Imperial College of Science, Technology and Medicine
Department of Electrical and Electronic Engineering

Embedded Deep Learning Systems for Robot Visual Intelligence

Alexandros Kouris

Supervised by Prof. Christos-Savvas Bouganis

Submitted in part of fulfilment of the requirements for the degree of
Doctor of Philosophy in Electrical and Electronic Engineering of Imperial College
and the Diploma of Imperial College, April 2023

To my beloved grandpa Nikos,
grandma Thaleia,
and aunt Nia.

Abstract

Over the past decade, the breakthrough of deep learning (DL) has revolutionised myriads of machine vision applications in the wild. As such, the Artificial Intelligence (AI) disruption has boosted the deployment of mobile robot systems, typically equipped with visual sensors, in real-world environments. However, the tremendous advancement of Deep Neural Networks (DNNs), which lead to their state-of-the-art accuracy, comes with constantly increasing computational and memory requirements. This poses significant challenges to their deployment in real-time applications, especially those related to mobile robots and Unmanned Aerial Vehicles (UAVs), that constitute inherently resource-constrained platforms due to their limited payload and power envelopes. Often confined by a strict latency budget and limited network availability upon deployment, off-loading computation to remote servers on the cloud is usually not a feasible option. This translates to sole reliance on the limited compute-capability devices available on-board, such as embedded Graphic Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs). Therefore, alleviating the workload burden of modern DNNs is of primary importance towards their efficient deployment in the embedded space. Additionally, taking into account the important interplay between the model (software) and architecture (hardware) design, efficiency can arise from incorporating knowledge about the one to the design process of the other (i.e. hardware-aware model design, or model-aware hardware design) or even co-optimisation (model-hardware co-design), instead of studying the two in isolation.

This thesis addresses the challenge of efficient DNN deployment, focusing on mobile robot and UAV applications. The main vehicle employed for efficiency is *application-specific customisation* (either at the hardware or model level), materialised through *approximation methodologies* applied all-across the deployment stack of DNNs. This spans from the (lowest) computation-level approaches (e.g. approximating DNN calculations by adopting low-precision and custom hardware accelerators), through the model and data layers to the (highest) application-level (e.g. approximating unseen camera views of an object through DNNs). The contributions of this work include: a multi-precision model cascade for efficient input-dependent classification; a progressive-refinement based approximate inference methodology for autonomous driving; a region-selection methodology for efficient UAV-based detection exploiting prior domain knowledge and multi-modal sensory inputs to dynamically reduce the workload; a self-supervised DL-based approach for indoor autonomous navigation through spatio-temporal representation learning; and an object-level depth inpainting pipeline for efficient occlusion-free 3D reconstruction.

Each of the above works exploits approximation and/or application-specific customisation methodologies to introduce and exploit a performance-accuracy trade-off at different abstraction levels, pushing the limits of efficient deployment of DNNs on-board mobile robot platforms, towards robot visual intelligence.

Copyright Declaration

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial-No Derivatives 4.0 International Licence (CC BY-NC-ND).

Under this licence, you may copy and redistribute the material in any medium or format on the condition that; you credit the author, do not use it for commercial purposes and do not distribute modified versions of the work.

When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text.

Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Statement of Originality

I declare that this thesis is entirely my own work and describes my own original research, except where otherwise indicated through appropriate references.

Acknowledgements

Ευχαριστώ...

...is a wonderful Greek word, that means "thank you". At this stage, I feel the need to say a huge thank you to a lot of people, who contributed in their own way, towards me reaching this important and exciting milestone of my career and life.

First and foremost, I would like to express my deepest gratitude to my advisor, Prof. Christos Bouganis. I have no words to describe how your continued guidance and wealth of support shaped me as a researcher and as a person. I feel lucky to work with you and sincerely thank you for all the trust (and patience) that are showing me, our countless technical and personal discussions, and our small daily interactions that made such a huge difference in my academic journey.

I am also grateful to Prof. Yiannis Demiris and Prof. Geoff Merrett for serving as the examiners of this thesis. Thank you for all your insightful feedback and genuine interest on my work, that made my Viva such a memorable experience.

During the early stages of my studies, I was fortunate to be mentored by Prof. Efstratios Gallopoulos at the University of Patras. Thank you for being the first person believing in me academically, and offering me so many opportunity to explore the world of research during my undergrad studies, cultivating my skills and ultimately sparking my interest to pursue this career path. I wouldn't have reached anywhere this point, if it wasn't for you.

While at University of Patras, I was also honoured to work with Prof. Nikolaos Aspragathos. Thank you for all your tangible support and guidance during these early days. Through our collaboration I was exposed to the wonderful field of robotics, and developed my technical and research skills at a pace that would not have been possible otherwise.

Throughout my undergrad and Ph.D. studies I was fortunate to have amazing colleagues. Two people that have been in more senior positions in the respective groups, who guided and supported me all these years have been Dr. Fotios Dimeas and Dr. Stylianos Venieris. Today I consider them to be two of my closest friends, but also two of the most talented people that I know and get inspired from. Thank you for everything that you learned me. Particularly in the context of this thesis, Stelios informally acted as a co-advisor to me, sharing countless hours of fruitful discussions, without which my work, skills and way of thinking would not have reached their current state.

Although a Ph.D. is meant to be a lonely journey, I was fortunate to be part of the Imperial College family. Throughout my studies, I was lucky to interact with amazing people, including Dr. Konstantinos Boikos, Dr. Andrea Nicastro, Prof. George Constantinides, Prof. Wayne Lyk and Dr. Stefan Leutenegger, with whom we shared exciting discussions, offering me plenty of ideas and great motivation to work harder. Thank you for all your time and excitement.

I would also like to thank Dr. Christos Kyrkou from the University of Cyprus for our numerous research conversations and collaboration, as well as sharing my vision for the creation of real-flight datasets and actively joining me in this endeavour. Alongside, I am grateful to every person in the Intelligent Digital Systems Lab (iDSL), Circuits and Systems (CAS) group, High Performance Embedded and Distributed Systems (HiPEDS) CDT, Spatial ML network and Imperial Robotics forum for all our interactions.

During my Ph.D. studies I was given the amazing opportunity to join Arm Research and Samsung AI Center in Cambridge for two internships. I feel blessed to have worked in such amazing environments and grateful to my line-managers at the time, Mr. Vasilios Laganakos and Dr. Ilias Leontiadis for their guidance and support; and close collaborators Michalis Korakakis, Stefanos Laskaridis, Mario Almeida, Lukasz Dudziak, Mohamed Abdelfattah, Hongxiang Fan, Rui Li, Mihai Anca, Shell Xu Hu, and Prof. Timothy Hospedales for our countless technical (and fun) discussions in Cambridge. I would also like to offer my gratitude to Prof. Nicholas Lane, who leads the Samsung AI Center in Cambridge, where I am currently based as a Research Scientist. Your support and positivity while I have been trying to balance my work duties and thesis write up has been invaluable, as well as your guidance during my first steps in industry. It is an honour to work with you, and have the benefit of your advice at all levels.

I also feel the need to thank all my friends for their support throughout these years. Sharing all these laughs together gave me the strength I needed to stay committed on this thesis, the hardest thing I have achieved so far. Especially, I would like to thank my partner Alexia for all her love, understanding, and patience supporting me wholeheartedly in all the ups and downs of this endeavour.

Finally, I feel the need to express my love to my parents Stefanos and Christiana and brother Iasonas for their full-hearted support and unconditional encouragement. Furthermore, I would like express my gratitude and deepest love to my grandpa Nikos, grandma Thaleia, and aunt Nia (and our lovely dog Lara, that has a photo of her hidden somewhere in the figures of this thesis) who I sadly lost while on this journey. This thesis is dedicated to you.

The support of the EPSRC Centre of Doctoral Training in High Performance Embedded and Distributed Systems (HiPEDS, Grant Reference EP/L016796/1) is gratefully acknowledged.

Contents

List of Figures **xiii**

List of Tables **xv**

List of Abbreviations **xix**

1 Introduction **1**

1.1 Motivation . . . 2

1.1.1 The Emergence of Deep Learning . . . 2

1.1.2 Deep Learning Systems . . . 3

1.1.3 Deep Learning and Robot Vision: Opportunities and Challenges . . . 6

1.2 Thesis Aims and Overview . . . 9

1.2.1 Towards Efficient Robot Visual Intelligence . . . 9

1.2.2 Thesis Overview . . . 11

1.2.3 Research Contributions . . . 13

1.3 Publications . . . 14

2 Background **17**

2.1 Overview . . . 18

2.2 The Deep Learning Era of Computer Vision . . . 19

2.2.1 The Computer Vision (R)evolution . . . 19

2.2.2 Deep Learning Principles . . . 25

2.2.3 Re-formulating Visual Perception through Deep Learning . . . 32

2.3 Embedded Systems . . . 40

2.3.1 Application-Specific Integrated Circuits (ASICs) . . . 41

2.3.2 General Purpose Graphics Processing Units (GPGPUs) . . . 41

2.3.3 Field-Programmable Gate Arrays (FPGAs) . . . 43

2.4 Efficient On-device Inference . . . 45

2.4.1 Key Performance Indicators, Beyond Accuracy . . . 46

2.4.2 Efficient Model Design . . . 47

2.4.3 Model Approximations . . . 48

2.4.4 Dynamic Inference . . . 51

2.5 Robot Visual Intelligence . . . 53

3 CascadeCNN **55**
3.1 Overview . . . 56
3.2 Motivation . . . 56
3.3 Related Work . . . 59
3.3.1 Approximate Computation in Deep Learning Inference . . . 59
3.3.2 Conditional Computation in Machine Learning . . . 61
3.4 Methodology . . . 64
3.4.1 Overview . . . 64
3.4.2 Quantisation . . . 65
3.4.3 Confidence Estimation . . . 67
3.4.4 Hardware Architecture . . . 69
3.4.5 Design Space Exploration (DSE) . . . 72
3.4.6 Latency-Aware DSE . . . 78
3.5 Evaluation . . . 82
3.5.1 Experimental Setup . . . 82
3.5.2 Architectural Performance and Accuracy . . . 82
3.5.3 Throughput-Optimised Cascade Evaluation . . . 84
3.5.4 Latency-aware Cascade Evaluation . . . 87
3.5.5 Performance Modelling Accuracy . . . 89
3.5.6 Qualitative Analysis on Scene Recognition . . . 90
3.6 Discussion . . . 92

4 Approximate LSTMs **95**
4.1 Overview . . . 96
4.2 Motivation . . . 96
4.3 Related Work . . . 99
4.3.1 Efficient LSTM Deployment . . . 99
4.3.2 Progressive Deep Learning Inference . . . 103
4.4 Methodology . . . 103
4.4.1 Overview . . . 103
4.4.2 LSTM Workload Analysis . . . 104
4.4.3 LSTM Approximations . . . 106
4.4.4 Hardware Architecture . . . 110
4.4.5 Design Space Exploration . . . 114
4.4.6 Model-Hardware Co-design . . . 115
4.5 Evaluation: A Case study on Autonomous Driving . . . 116
4.5.1 Autonomous Driving Background . . . 116
4.5.2 Experimental Setup . . . 117
4.5.3 Architectural Performance . . . 119
4.5.4 End-to-end Evaluation . . . 120
4.5.5 Qualitative Analysis . . . 125
4.6 Discussion . . . 126

5 Efficient UAV-based Detection **129**
5.1 Overview . . . 130
5.2 Motivation . . . 131
5.3 Related Work . . . 134
5.3.1 Convolutional Object Detectors . . . 134
5.3.2 Efficient UAV-based Object Detection . . . 136
5.3.3 Vehicle Detection Datasets . . . 138
5.4 The CyCAR Dataset . . . 138
5.4.1 Context . . . 139
5.4.2 Data and Annotations . . . 139
5.5 Methodology . . . 141
5.5.1 Overview . . . 141
5.5.2 Informed UAV-based Region Selection . . . 142
5.5.3 Region Selection Criteria for Vehicle Detection . . . 144
5.5.4 GPU Deployment Aspects . . . 151
5.6 Evaluation . . . 152
5.6.1 Experimental Setup . . . 152
5.6.2 Speed/Accuracy Trade-offs in Object Detectors . . . 153
5.6.3 End-to-end Evaluation . . . 155
5.6.4 Ablation on Region Selection Criteria . . . 158
5.6.5 Qualitative Analysis . . . 159
5.7 Discussion . . . 160

6 Self-Supervised UAV Navigation **163**
6.1 Overview . . . 164
6.2 Motivation . . . 165
6.3 Related Work . . . 168
6.3.1 Data Collection and Annotation for UAV Navigation . . . 168
6.3.2 DNN-based Approaches for UAV Navigation . . . 169
6.4 Methodology . . . 172
6.4.1 Overview . . . 172
6.4.2 Self-Supervised Data Collection and Annotation . . . 172
6.4.3 The *SelfNav* Dataset . . . 175
6.4.4 CNN Architecture . . . 177
6.4.5 Model Training and Optimisations . . . 180
6.4.6 Local Motion Planner . . . 182
6.5 Evaluation . . . 185
6.5.1 Experimental Setup . . . 185
6.5.2 Model Design and Prediction Accuracy . . . 186
6.5.3 Training- and Inference-time Optimisation Ablations . . . 189
6.5.4 End-to-end UAV Navigation Comparison . . . 191
6.5.5 Qualitative Analysis . . . 192
6.6 Discussion . . . 193

7 Occlusion-free 3D reconstruction 195
7.1 Overview . . . 196
7.2 Motivation . . . 197
7.3 Related Work . . . 202
7.3.1 Visual Simultaneous Localisation and Mapping . . . 202
7.3.2 Learnable 3D Reconstruction . . . 203
7.3.3 Shape Completion and Occlusion Handling . . . 204
7.4 Methodology . . . 210
7.4.1 Overview . . . 210
7.4.2 3D Shape Representation through Vis-à-Vis Camera Pairs . . . 213
7.4.3 Occlusion Handling Model Design and Training . . . 217
7.4.4 The *UnScene* Dataset . . . 222
7.4.5 3D Reconstruction and Multi-view Fusion . . . 224
7.5 Evaluation . . . 230
7.5.1 Experimental Setup . . . 231
7.5.2 Model Accuracy . . . 232
7.5.3 Time-to-Accuracy Evaluation in 3D Reconstruction . . . 234
7.5.4 Qualitative Analysis . . . 237
7.5.5 Case study on Robot Grasping . . . 243
7.6 Discussion . . . 245

8 Conclusions and Future Work 247
8.1 Does *general purpose* mean "*good for nothing*"? . . . 248
8.1.1 Computation-level Customisation . . . 248
8.1.2 Model-level Customisation . . . 250
8.1.3 Data-level Customisation . . . 251
8.1.4 Task-level Customisation . . . 252
8.2 Discussion . . . 254
8.2.1 Dynamic Inference . . . 254
8.2.2 Optimisation Objectives . . . 255
8.2.3 Customisation Interplay . . . 256
8.2.4 Design Automation . . . 257
8.2.5 Target Tasks/Models . . . 258
8.3 Closing Remarks . . . 259

Bibliography 261

List of Figures

1.1 Evolution of DNN workloads and available computation power over the years. . . 4
1.2 Development flow of Deep Learning models. 5
1.3 Landscape of Robot Visual Intelligence. 7
1.4 Deployment Stack of Deep Learning solutions. 10

2.1 Perspective and Orthographic Camera Projection models. 21
2.2 Evolution of the Computer Vision pipeline. 24
2.3 Illustration of a Convolutional Neural Network. 27
2.4 Illustration of a Long Short-Term Memory model. 30
2.5 Architecture of LeNet, AlexNet and VGG. 34
2.6 Architecture of GoogleNet. 35
2.7 Architecture of ResNet and MobileNet. 36
2.8 Hardware structure of CPUs, GPUs and FPGAs. 42

3.1 Automated Toolflow of CascadeCNN. 63
3.2 System architecture of CascadeCNN. 65
3.3 Block Floating-Point quantisation scheme. 65
3.4 Illustration of CascadeCNN's quantisation search space. 66
3.5 Comparison between quantisation approaches. 67
3.6 Intuition behind the proposed confidence evaluation metric (gBvSB). 68
3.7 Demonstration of gBvSB across predictions on ImageNet validation data. 69
3.8 Hardware architecture of CascadeCNN's engine. 72
3.9 The roofline performance model. 73
3.10 Different deployment settings of CascadeCNN. 77
3.11 Processing timeline of samples under differently optimised systems. 78
3.12 Evaluation of the proposed hardware architecture and quantisation scheme. 83
3.13 End-to-end evaluation of CascadeCNN's performance-error trade-off. 85
3.14 Qualitative analysis on Places365 dataset for scene recognition. 91

4.1 Conventional vs Progressive inference. 98

4.2 Automated Toolflow of Approximate LSTM. 104
4.3 Computational Structure of an LSTM model. 106
4.4 Hardware architecture of Approximate LSTM accelerator. 112
4.5 Performance evaluation of proposed hardware architecture. 121
4.6 End-to-end evaluation of Approximate LSTM's performance-accuracy trade-off. . 122
4.7 Qualitative analysis of performance-accuracy trade-off. 125
4.8 Qualitative analysis of QoR in a real-world application setting. 126

5.1 Cloud and on-device processing settings on UAVs. 132
5.2 Sample data from the newly-created *CyCAR* dataset. 140
5.3 The proposed Region Selection module within a detection model. 144
5.4 Illustration of region selection scale and density criteria. 146
5.5 Horizontal vs Essential (Oriented) bounding boxes. 147
5.6 Region proposals on real-world data. 149
5.7 Road Segmentation based on HSV-filtering, at different altitudes. 151
5.8 Impact of region proposal cardinality on the latency-accuracy trade-off. 156
5.9 Qualitative analysis of proposed Region Selection approach. 161

6.1 Sensor placement for self-supervised data collection. 174
6.2 Navigation policies for self-supervised data collection. 175
6.3 Sample data from the newly-created *SelfNav* dataset. 176
6.4 Architecture of the proposed 2-stream regression CNN. 178
6.5 Inference process of proposed navigation approach. 180
6.6 Illustration of robot/task specific tunable parameters. 183
6.7 Accuracy evaluation of distance predictions. 187
6.8 Qualitative case study on Autonomous Navigation. 192

7.1 Summary of occlusion types and image-space 3D representations. 200
7.2 Proposed occlusion-handling pipeline for 3D reconstruction. 212
7.3 Proposed vis-á-vis camera representation. 214
7.4 Encoder-decoder CNN architecture for occlusion-handling. 218
7.5 Loss calculation for the proposed occlusion-handling formulation. 221
7.6 Sample data from the newly-created *UnScene* dataset. 223
7.7 Time-to-accuracy evaluation in 3D reconstruction. 236
7.8 Qualitative analysis of occlusion-handling model's predictions. 238
7.9 (contd.) Qualitative analysis of occlusion-handling model's predictions. 239
7.10 Failure cases of proposed occlusion-handling model. 240
7.11 Qualitative analysis of single-view 3D reconstruction. 241
7.12 Qualitative comparison of multi-view 3D reconstruction approaches. 242
7.13 Effectiveness of proposed methodology in robot grasping. 244

List of Tables

1.1 Summary of the focus of each Chapter in this Thesis. 16

2.1 Technical Specifications of employed Nvidia GPU platforms. 43
2.2 Available resources of employed Xilinx FPGA platforms. 45

3.1 Performance Evaluation of Throughput-optimised CascadeCNN. 86
3.2 Performance Evaluation of Latency-aware CascadeCNN. 88
3.3 Accuracy Evaluation of Analytical Performance Model for CascadeCNN. 89

4.1 Performance/Efficiency Evaluation of Approximate LSTM approach. 124

5.1 Latency-Accuracy trade-off between Object Detectors. 154
5.2 End-to-end Evaluation of proposed Region Selection approach. 158
5.3 Ablation between proposed Region-Selection criteria. 159

6.1 Ablation between different Model Design choices. 188
6.2 Ablation of proposed Training and Inference Optimisations. 190
6.3 End-to-end Comparison on Autonomous Navigation. 191

7.1 Evaluation (and Ablation) of proposed Occlusion-handling Model. 234
7.2 Inference Latency and Throughput of each module in the proposed pipeline. . . . 237

List of Algorithms

1 Computation of a CNN layer expressed as a tiled matrix multiplication. 74
2 Approximate LSTM computation via iterative refinement. 110
3 Object detection through a region-based approach (and proposed modifications). . 142
4 Scale-based region selection criterion for vehicle detection. 146
5 Density-based region selection criterion for vehicle detection. 148
6 Obstacle avoidance using distance predictions. 184

List of Abbreviations

AGV . . . . . Automated Guided Vehicles

AI . . . . . Artificial Intelligence

ANN . . . . . Artificial Neural Network

API . . . . . Application Programming Interface

AR . . . . . Augmented Reality

ASIC . . . . . Application-Specific Integrated Circuit

BNN . . . . . Binary Neural Network

BRAM . . . . Block Random Access Memory

BW . . . . . Bandwidth

CAD . . . . . Computer-Aided Design

CE . . . . . Cross Entropy

CEU . . . . . Confidence Evaluation Unit

CNN . . . . . Convolutional Neural Network

CONV . . . . . Convolutional (Layer)

CPU . . . . . Central Processing Unit

CUDA . . . . . Compute Unified Device Architecture

CV . . . . . Computer Vision

DL . . . . . Deep Learning

DMA . . . . . Direct Memory Access

DNN . . . . . Deep Neural Network

DoF . . . . . Degree of Freedom

DRN . . . . . Dilated Residual Network

DSE . . . . . Design Space Exploration

DSP . . . . . Digital Signal Processor

FC . . . . . Fully-Connected (Layer)

FE . . . . . Feature Extractor

FIFO . . . . . First-In-First-Out

FLOPs . . . . Floating Point Operations

FN . . . . . False Negative

FoV . . . . . Field of View

FP . . . . . False Positive

FPGA . . . . . Field Programmable Gate Array

FPN . . . . . Feature Pyramid Network

fps . . . . . Frames per Second

gBvSB . . . . . generalised Best-versus-Second-Best (metric)

GPS . . . . . Global Positioning System

GPU . . . . . Graphics Processing Unit

GRU . . . . . Gated Recurrent Unit

HBB . . . . . Horizontal Bounding Box

HDL . . . . . Hardware Description Language

HLS . . . . . High-Level Synthesis

HPU . . . . . High-Precision Unit

HW . . . . . Hardware

IC . . . . . Integrated Circuit

ICP . . . . . Iterative Closest Point

IMU . . . . . Inertial Measurement Unit

IOB . . . . . Input/Output Block

IoT . . . . . Internet of Things

IoU . . . . . Intersection Over Union

IR . . . . . Infra-Red

KL . . . . . Kullback–Leibler (divergence) (metric)

KPI . . . . . Key Performance Indicator

LDI . . . . . Layered Depth Image

LPU . . . . . Low-Precision Unit

LRCN . . . . . Long-term Recurrent Convolutional Network

LRN . . . . . Local Response Normalisation

LSTM . . . . . Long Short-Term Memory (Model)

LUT . . . . . Look-Up Table

MACC . . . . Multiply-And-Accumulate

MAE . . . . . Mean Absolute Error

mAP . . . . . mean Average Precision

MAV . . . . . Micro Aerial Vehicle

mIoU . . . . . mean Intersection-over-Union

ML . . . . . Machine Learning

MM . . . . . Matrix Multiply

MSE . . . . . Mean Square Error

NAS . . . . . Neural Architecture Search

NLP . . . . . Natural Language Processing

NMS . . . . . Non-Maximum Suppression

NN . . . . . Neural Network

NPU . . . . . Neural Processing Unit

OBB . . . . . Oriented Bounding Box

OOD . . . . . Out of Distribution

p.p. . . . . . Percentage Points

PR . . . . . Precision-Recall

QoR . . . . . Quality of Result

RAM . . . . . Random Access Memory

ReLU . . . . . Rectified Linear Unit

RGB . . . . . Red-Green-Blue (Image)

RGB-D . . . . Red-Green-Blue-Depth (Image)

RL . . . . . Reinforcement Learning

RMSE . . . . . Root Mean Square Error

RNN . . . . . Recurrent Neural Network

ROM . . . . . Read Only Memory

RPN . . . . . Region Proposal Network

SDF . . . . . Signed Distance Function

SDK . . . . . Software Development Kit

SfM . . . . . Structure from Motion

SGD . . . . . Stochastic Gradient Descent

SIMD . . . . . Single-Instruction Multiple-Data

SLAM . . . . . Simultaneous Localisation and Mapping

SoC . . . . . System-on-Chip

SSD . . . . . Single Shot Detector

SVD . . . . . Singular Value Decomposition

SW . . . . . Software

TDP . . . . . Thermal Design Power

TP . . . . . True Positive

TPU . . . . . Tensor Processing Unit

TSDF . . . . . Truncated Signed Distance Function

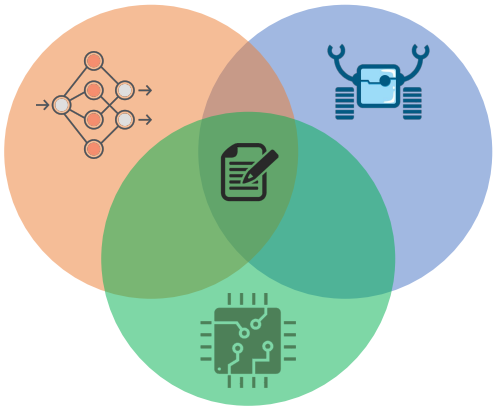
TTA . . . . . Time to Accuracy (metric)

UAV . . . . . Unmanned Aerial Vehicle

VIO . . . . . Visual Inertial Odometry

VO . . . . . Visual Odometry

VR . . . . . Virtual Reality



1 Introduction

Contents

1.1 Motivation 2
1.1.1 The Emergence of Deep Learning 2
1.1.2 Deep Learning Systems 3
1.1.3 Deep Learning and Robot Vision: Opportunities and Challenges 6
1.2 Thesis Aims and Overview 9
1.2.1 Towards Efficient Robot Visual Intelligence 9
1.2.2 Thesis Overview 11
1.2.3 Research Contributions 13
1.3 Publications 14

Keywords: *Deep Learning, Embedded Systems, Mobile Robots, Unmanned Aerial Vehicles, Visual Intelligence, Deployment Stack, Efficiency, Approximation, Application-specific Customisation*

This chapter is partly based on a position paper co-authored with Dr. Stylianos I. Venieris and Prof. Christos-Savvas Bouganis: (ISVLSI 2019); and a relevant workshop presentation (EMDL 2018).

1.1 Motivation

1.1.1 The Emergence of Deep Learning

Over the past decade, Artificial Intelligence (AI) has demonstrated tremendous advancement through the emergence of data-driven approaches known as Deep Learning (DL) [1][2]. Among other applications, deep learning has revolutionised the field of Computer Vision (CV), with Deep Neural Networks (DNNs) quickly dominating the state-of-the-art across all primitive CV tasks, including Image Classification [3], Object Recognition [4][5] and Semantic Segmentation [6][7].

This success of DNN-based approaches can be attributed to their ability to extract effective high-level representations (features) from raw sensor inputs, based on experience obtained through statistical learning [8] over a large volume of data [9], instead of hand-crafted analytical algorithms [10]. The term neural network (NN) was coined driven by our understanding of how the human brain works, loosely inspiring the development of the main computation element of NNs, called a neuron. As such neural networks essentially comprise the brain-inspired subcategory of Machine Learning (ML) defined as "the field of study that gives computers the ability to learn without being explicitly programmed" by Arthur Samuel back in 1959 [11].

The development of ML computer programs, also referred to as Software 2.0 [12][13], typically comprises two stages. Initially, the values of all learnable parameters (weights) of a model are determined in view of a target dataset, through an iterative optimisation process called *training*. Training is typically an offline process, performed once over a period of days, weeks or even longer, powered by datacentre-grade hardware [14]. Subsequently, in the online phase called *inference*, the model is deployed on the target task where it computes an output in view of new input samples, using its trained weights. This deployment increasingly takes place in the edge device landscape [15], with a timespan in the order of milliseconds per inference.

Although the concept of (deep) neural networks has been around for decades [16][17][18], two main driving factors can be identified behind their first breakthroughs in speech [19] and image recognition [20] during the early 2010s: (i) The creation of large-scale open-source *datasets* of real-world data [21][22] and (ii) the availability of unprecedented *computational power*, driven by the re-purposing of Graphics Processing Units (GPUs) [23] as Tensor Processing accelerators [24][25].

As a result, today DNNs empower a vast range of real-world applications, some with enormous societal impact, including autonomous driving [26], cancer detection [27], drug discovery [28], language translation [29], speech recognition [30], complex planning [31] and robotics [32].

1.1.2 Deep Learning Systems

The remarkable evolution of deep learning models however, came at the cost of rapidly escalating computational and memory requirements [33]. This can be attributed to the growing size and complexity of the proposed DNN topologies, scaling their underlying computational demands and number of trainable parameters over the years, in a quest to increase the expressive power of their representations [34]. As a result neural networks have been getting deeper [35] and wider [36] and feature more complex architectures [37][38].

Over the past decade, this increase in computational demand had been accommodated by a corresponding increase in the available computational power. However, extrapolating forward the slow-down of Moore's law [39][40] along with the recently observed exponential growth in workload [41], progress in the field will quickly become technically and economically unsustainable [42].

This is illustrated in Figure 1.1, where it is evident that the rate at which commercial devices are gaining computational power[1], including server/edge-grade Graphics Processing Units (GPUs) [23], mobile Neural Processing Units (NPUs) [43] and high-end Tensor Processors Units (TPUs) [44]), is overshadowed by the increase in computational demand[2] of state-of-the-art Computer Vision (CV) and Natural Language Processing (NLP) models [45]. With the failure of Dennard scaling [46] expected to further limit the pace of multi-core system scaling [47], the hardware community does not have a direct answer to remedy this challenge.

In recent years, significant efforts to bridge this gap have emerged both in the hardware and software communities, forming what is illustrated in Figure 1.1 as the Embedded ML space [48]. Primary focus of these efforts is to produce DNN models with reduced computational and memory footprint, suitable for efficient inference outside of the datacentre, on mobile/embedded devices [49] or at the "edge" [15]. Efficient hand-crafted [50][51] or Neural Architecture Search

[1] *Computational power* is described by the theoretical peak number of Operations per second (OP/s) each device can support.

[2] *Computational demand* is expressed by the number of Floating Point Operations (FLOPs) the DNN model requires per inference.

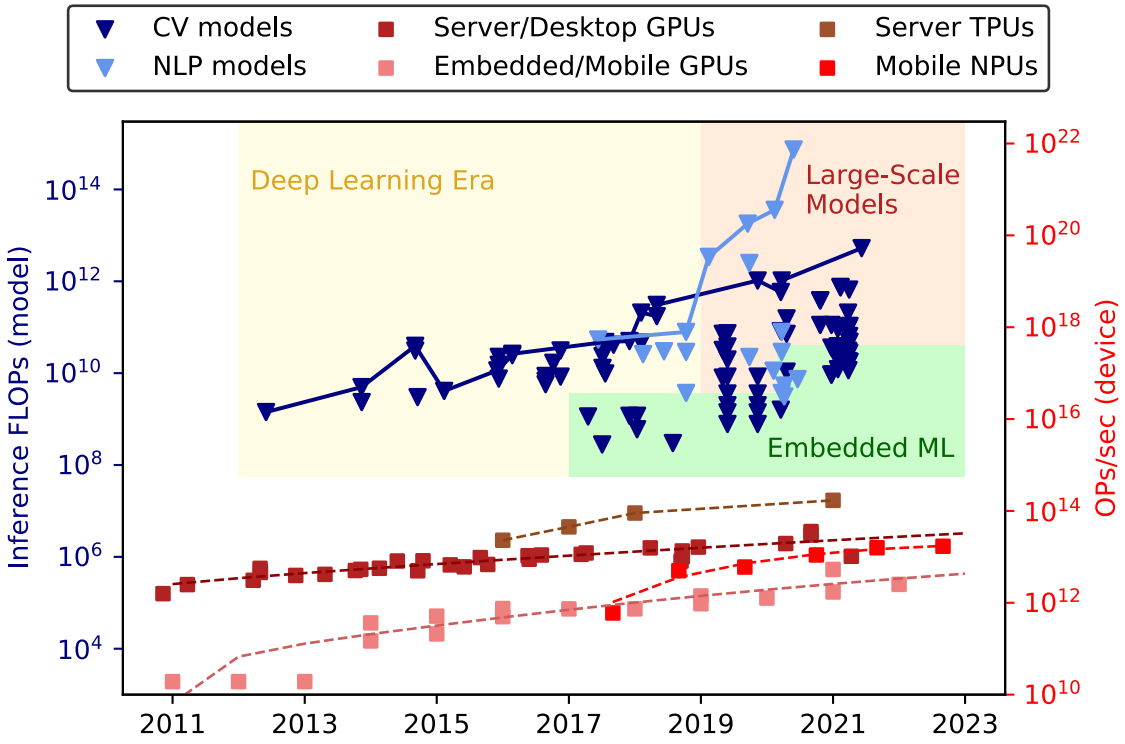


Figure 1.1: Evolution of DNN computational demands (FLOPs) in CV and NLP tasks; and hardware computational power (OP/s) in the Cloud, Embedded and Mobile landscape, over the years[3]. Dashed lines indicate tendency, while solid lines indicate the succession of state-of-the-art models (accuracy-wise).

(NAS) [52][53] -based model design has played an important role in this direction, proposing new modules and diverging from the traditional CNN architectures.

However, recent studies have shown that DNNs are over-parametrised "by design", as this redundancy essentially offers enhanced flexibility during the training process, leading to their remarkable generalisation capabilities [54]. Along these lines, several approximate inference methodologies have been introduced, including model quantisation [55], parameter pruning [56], low-rank compression [57] and knowledge distillation [58], aiming to exploit this redundancy and alleviate the computational burden post-training, when it is no longer necessary.

The typical development flow of such approaches is abstracted in Figure 1.2a. Ordinarily, optimisations have been taking place from an algorithmic perspective, employing a proxy metric for inference speed [50]. *Model size* (number of parameters) and *workload* (number of operations) form the most commonly used optimisation objectives, accompanying *accuracy* in this stream of works. As a next step, development tools are used to map the resulting model on the target deployment platform, where performance gains can be realised. This conventional development flow however,

[3]Based on an augmented version of the data released by [59].

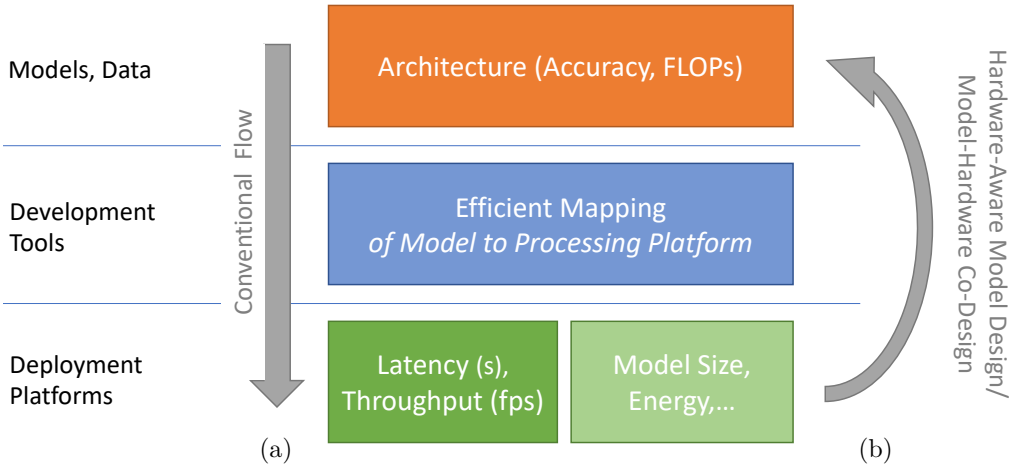


Figure 1.2: Development flow of Deep Learning models: (a) Conventional; (b) Hardware in-the-loop.

often demonstrates underwhelming actual speed-ups, as the proxy metrics fail to capture some characteristics of the model topology that affect its attainable performance on the platform at hand. For example the regularity and uniformity of the computation within and across layers can vastly affect the model's deployment performance [60], owing to inefficiencies and imbalances of the mapping causing underutilisation of the computational resources of the target device.

From a model perspective, recent work aims to see through this challenge by closing the loop between model design and deployment (Fig. 1.2b). This is enabled by the use of *hardware-aware* model design methodologies, explicitly evaluating model candidates on the target platform in each design iteration [61]. Such methodologies typically consider different metrics of interest such as latency, throughput but also model size, power/energy etc, or estimate the attainable performance of each topology through the use of data-driven methodologies [62].

From a hardware perspective, aiming to alleviate the inefficient mapping of models evident when targeting conventional GPUs [63], *domain-specific* hardware accelerators have emerged [64]. The development of accelerators tailored to a specific model (or a family of models), termed model-aware hardware design, has been enabled by the creation of automated toolflows that analyse the computation and efficiently parameterise the hardware to yield an efficient mapping [65]. Such systems typically take the form of Application Specific Integrated Circuits (ASICs) [66], or employ reconfigurable fabric through Field Programmable Gate Arrays (FPGAs) [67]. Driven by the fact that applying extreme specialisation on hardware while leaving the model intact demonstrates diminishing returns [68], recent efforts took a step further, showing great potential from holistically *co-designing* the model and corresponding hardware [69].

Commonly these approaches offer generalised solutions for improving inference efficiency, by statically exploiting the speed-accuracy trade-off in a *uniform* manner across input samples, while being *agnostic* to the target application. Conversely, this thesis explores more dynamic, *input-dependent* or *application-specific* optimisation approaches, offering *highly-customised solutions* to the examined use-case, pushing the limits of efficiency in deep learning inference and deployment aware of the Key Performance Indicator (KPI) requirements of the target application.

1.1.3 Deep Learning and Robot Vision: Opportunities and Challenges

In this context, this thesis revolves around different applications of (mobile) robotics, that have been fundamentally reformed by the emergence of deep learning. With robots being active agents that operate in and interact with complex real-world environments based on their partially observable state, data-driven perception (and planning) is becoming increasingly popular, witnessing a paradigm-shift from model-based approaches [70].

The deep learning revolution in computer vision tasks, has empowered mobile robots with powerful perception modules guiding robot vision through a vast pace of progress [71]. As a result, over the past 5 years the robotics community has been continuously adopting the latest advancements of deep learning, drastically increasing the reliability and applicability of the proposed systems in real-world applications. Additionally, the effectiveness and affordability of monocular cameras has eliminated the reliance of commercially available Automated Guided Vehicles (AGVs) to heavy-weight and expensive sensors, allowing for low-cost service and entertainment vision-based robots to reach consumer homes and the public sector.

Among others, tasks such as autonomous robot navigation [72], object tracking [73], autonomous driving [26] and driving assistant systems [74], object grasping [75], behaviour planning [76] and 3D reconstruction [77], have experienced unprecedented progress by incorporating learning-based methodologies.

Along the same lines, significant progress has been made in the field of aerial robotics, driven by the rapid development of inexpensive Unmanned Aerial Vehicles (UAVs), and especially commercial Micro Aerial Vehicles (MAVs) with reduced weight (typically less than 2kg) and form factor (diameter up to 1m) [78], such as quadcopters (drones), offering enhanced versatility and ease of use. Typically equipped with one or more visual sensors, UAVs are employed in a wide range of civilian applications, spanning from search and rescue operations [79] and

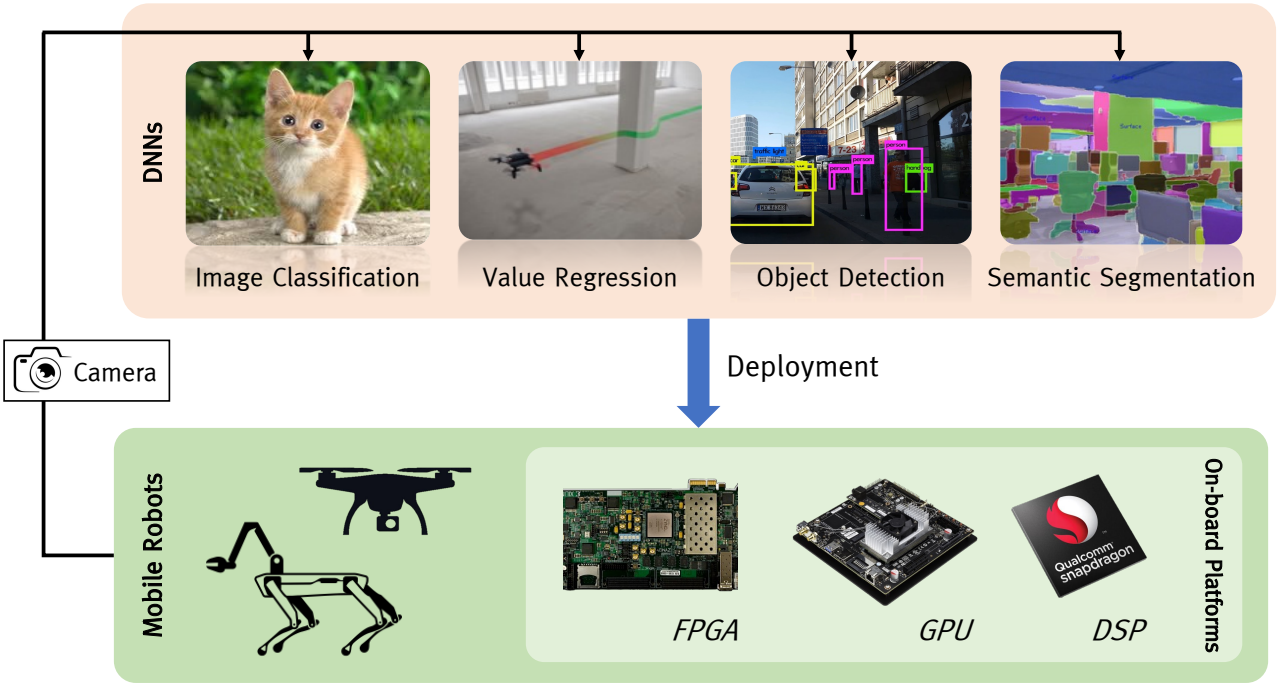


Figure 1.3: Landscape of Robot Visual Intelligence.

wildfire tracking [80] to precision agriculture [81], infrastructure inspection [82] and smart city monitoring [83] and remote sensing [84].

In robot vision, however, prediction errors may lead to wrong planning decisions and subsequently actions that can cause catastrophic failures. Such situations hinder the success of the robot mission, while also potentially endanger humans that share the same environment. Despite their state-of-the-art accuracy, deep learning models still exhibit unpredictable generalisation errors [85] and subpar results on more challenging situations [86][87]. Unfortunately, paving the last mile towards highly-accurate models appears to be the most computationally expensive part of the process. Recent studies [42] showed that gaining a 5% accuracy improvement on ImageNet state-of-the-art ([88] as of 2020) is estimated to require 10,0000× more computation. This computational complexity will soon be constraining for several real-world applications of deep learning.

Moreover, in many use-cases where perception results are directly involved in planning decisions, sustaining a real-time inference speed is also of upmost importance [89]. This confines their deployment on-board the robot platform, as communication with cloud infrastructure would incur prohibitive costs [90] and may be affected by network availability, jitter and coverage [91]. At the same time, mobile robots, and more so MAVs, are additionally challenged by their limited size, payload capability and power budget. These constraints severely limit the availability of

computational resources on-board, which often comprise *embedded* GPUs [92] or FPGAs [93] fitting within a typical power envelope of up to 30W [94], while offering an order of magnitude lower attainable performance than their desktop-grade counterparts [95]. Additionally, the modular pipeline of robotic applications leads to resource-sharing between different workloads [96][97], while the embedded compute platform may be subject to thermal management techniques [98], dynamically affecting the availability of computational power on device.

Conclusively, the challenges of deploying DNNs on the landscape of mobile robots and UAVs (Fig. 1.3) are significantly aggravated compared to other use-cases. This entanglement is caused by multiple conflicting deployment requirements such as: (i) *high accuracy* leading to increased computational complexity; (ii) *high performance* in terms of low latency and high-throughput, to enable mission-critical decision making in real-time while remaining able to cope with the data generation rate of on-board sensors, respectively; as well as (iii) limitations in the *compute capability* available on-board. In response, this thesis follows an *application-specific customisation* approach to enable the efficient deployment of DNNs on the compute platforms available on-board mobile robots, by traversing the underlying performance-accuracy trade-off in an informed way, aware of the requirements of the end task.

1.2 Thesis Aims and Overview

1.2.1 Towards Efficient Robot Visual Intelligence

This thesis stratifies the study of the deployment process of deep learning solutions in real-world applications, through the stack introduced in Figure 1.4.

The top layer of this stack, termed the **task layer**, captures the end-goal of the target application, independently of the adopted solution. In the context of robotics, this could be grasping an object [99], navigating autonomously within a space [100] or constructing a 3D representation of the robot's environment [101]. At this level, application-specific metrics are incorporated to evaluate the effectiveness of different design choices in an end-to-end manner. These include task completion *speed*, task-level *accuracy*, mean time between *failures* etc.

Beneath the task layer on the stack lies the **data layer**. This considers the input and output representations of AI models contributing as modules to the target application (and corresponding adaptations on the training process), while the models themselves are still viewed as black-boxes. Design choices at this level determine the *type* (i.e. RGB, depth [102], spatio-temporal features [103], multiple frames [104] and others, or combinations of the above) and *format* (i.e. resolution, frequency, pre-processing and others [105]) of input and output modalities of each model. The evaluation at this stage is commonly focused on accuracy-centric benchmark datasets [21], exploring the underlying trade-offs between alternatives.

Moving one level lower on the stack, the **model layer** focuses on the topology of the deployed DNNs. Here the structure, depth and width of each model is configured manually [106] or in an automated way [107], at design time or through post-training adaptation [108]. At this level, both accuracy and computational/memory demands are considered through standardised benchmarks [109][110] aiming to reflect on application-specific requirements.

At the bottom of the stack one meets the **computation layer**, concerning the underlying workload of each model (i.e. the actual calculations and representation of the operands [111] [112] that are required during inference), as well as their mapping to the target computational platform [113] and even the hardware design of the computation engine itself (architecture, memory hierarchy etc.) [65]. This stage typically considers primitive numerical metrics such as absolute or relative Root Mean Square Error (RMSE), and raw performance measurements such as latency or FLOPs per second/area/power, to verify the validity and evaluate design choices of the system respectively.

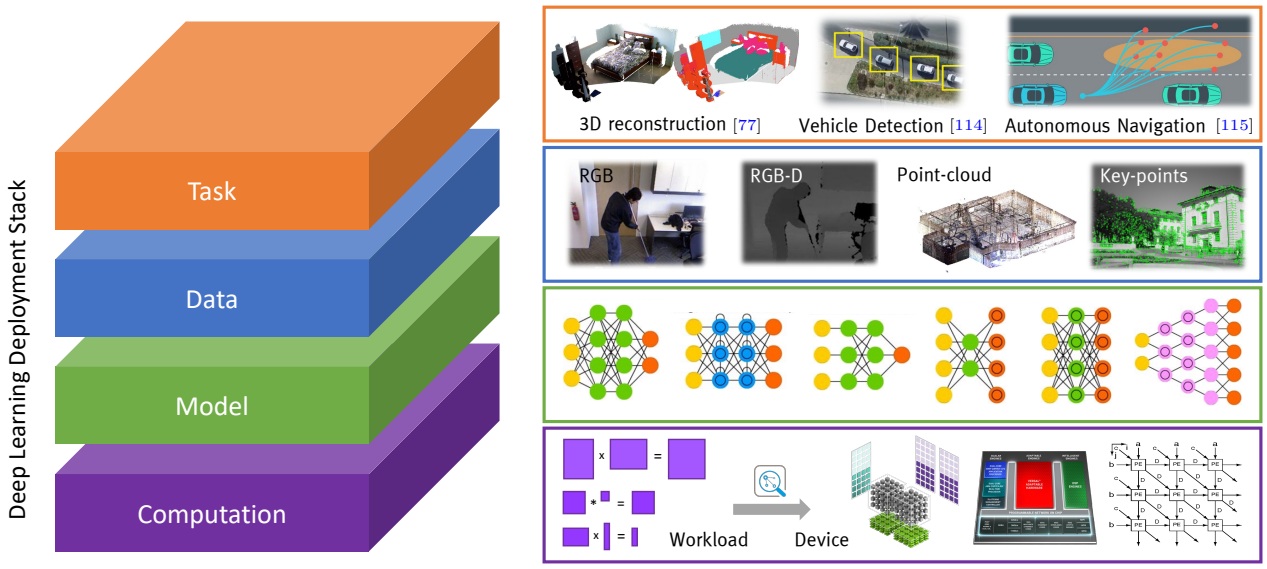


Figure 1.4: Deployment Stack of Deep Learning solutions.

It is noteworthy that lower layers of the stack are naturally more generic and application-agnostic compared to higher layers that become more specialised on a given task. This fact is also reflected on the examined metrics of each layer. As a result, a strong interplay exists between all layers, with speed-accuracy requirements being populated in a top-down manner; while the impact of any optimisations performed at lower levels typically propagates upwards.

Thesis Aim

The aim of this thesis is to address some of the key deployment challenges of vision-based robot applications, taking advantage of the unique *efficiency* opportunities that arise from the respective use-case. Towards this goal, this thesis follows a holistic approach, considering the all layers of the proposed DNN deployment stack (Fig. 1.4) and visiting different AI-enabled robot vision applications, in the context of inference-time efficiency. Across the whole stack, the main vehicle adopted for efficiency is *customisation*, taking the form of novel *approximation* methodologies. While approximations typically introduce a speed-accuracy trade-off at the layer they are applied, by projecting the impact of each approximation to the task layer and incorporating *application-specific* customisations, the provided solution is able to optimise for inference efficiency in view of the requirements/constraints of the end-task itself. As a result, the penalty in accuracy at the examined layer can often be mitigated, resulting to a corresponding speed-generality trade-off at the task layer instead.

Notably, although throughout the chapters of this thesis the goal remains the same: *efficiency*, the range of applicable optimisation methods differs significantly between different layers of the deployment stack: from approximating the calculations of DNN inference towards *computational efficiency* at the lowest layer; to employing a DNN to approximate the output of an otherwise time-consuming process towards *task efficiency* at the top of the stack; with efficient model design and data/representation efficiency in between. In all cases, this thesis follows the approach of tailoring the proposed solutions to the needs of the target application through domain-specific and/or input-dependent approximations, offering highly-customised systems that exploit the underlying trade-offs introduced by approximation.

1.2.2 Thesis Overview

More specifically, this thesis is organised into eight chapters. Initially, the motivation and contributions of the presented works are discussed in **Chapter 1**; followed by a summary of background information focusing on Deep Learning and Systems aspects that will be visited later in the manuscript in **Chapter 2**.

Chapter 3 studies the *computation layer* of the deployment stack focusing on the *image classification* task, which is widely used in robotics (e.g. for place recognition). Exploiting the fact that not all input samples are equally challenging to classify, the aim of this chapter is to approximate the inference computations through quantisation, employing a different compression level for each sample, in an *input-dependent* manner. Towards this goal a multi-precision cascade of CNNs is introduced, where each sample is processed by progressively higher-precision variants of the same CNN, with a novel confidence evaluation unit in-between dynamically determining where each sample should exit. To realise the performance gains of quantisation, a scalable FPGA-based *CNN accelerator* is designed along with an automated toolflow that jointly optimises the cascade structure and hardware architecture to the target CNN-FPGA pair, in view of application-specific requirements in throughput, latency and accuracy.

Chapter 4 spreads across the *computation and model layers* focusing on *LSTM* models for latency-critical use-cases, such as *autonomous driving*. To address the unique deployment challenges of LSTMs, a novel methodology coupling low-rank approximation with parameter pruning is introduced; enabling the iterative refinement of the model predictions through a *progressive inference* paradigm. Alongside, a configurable FPGA-based *LSTM accelerator* is designed, with its

configuration being jointly optimised with the approximation scheme in a co-design approach that maximises efficiency, while satisfying application-specific requirements in latency and accuracy.

Chapter 5 remains targeted to the *model layer* focusing on *detection CNNs* for *UAV-based vehicle detection*. Exploiting the additional sensor information from the UAV, along with prior-domain knowledge, a novel region selection approach is introduced aiming to identify false-positives region proposals at an early-stage of the computation, guiding a novel approximate inference methodology. As a result, inference latency is dynamically reduced in an *input-dependent* manner, without any compromise in accuracy. Alongside, a new dataset for vehicle detection in aerial imagery, named *CyCAR*, is created and shared with the community.

Chapter 6 visits the *data layer* focusing on *regression CNNs* for *autonomous navigation*. A self-supervised data collection and annotation methodology is introduced to facilitate the learning process of data-driven navigation approaches. Using the proposed method, a new large-scale UAV navigation dataset termed *SelfNav* is created and shared with the community. Alongside, a specially trained two-stream CNN is designed, with the ability to extract implicit spatio-temporal representations from the UAV camera stream, and optimised to achieve comparable inference latency to its single-stream counterparts. The proposed model is able to accurately approximate distance sensor readings from visual data, enhancing the applicability of autonomous navigation to low-cost platforms. Subsequently, a novel motion planner relying on the predictions of this model is proposed, demonstrating enhanced flexibility and autonomy in indoor environments compared to other learnable approaches.

Chapter 7 reaches to the *task layer* focusing on *dense-prediction CNNs* for *semantic 3D reconstruction*. A data-driven methodology is introduced to approximate a novel 3D representation by predicting depth views from artificial object-centric camera poses, given an RGB-D input from a cluttered scene. This task, enabled by a newly-created occlusion-handling dataset termed *UnScene*, allows for inpainting of occluded areas on the depth cue, while integration to a multi-view pipeline via a novel informed fusion scheme considering both the observed and predicted information, offers a progressive refinement of the 3D representation as more views become available. As a result the proposed method improves task-level efficiency, in terms of providing a significantly improved time-to-accuracy trade-off in 3D reconstruction, facilitating several down-stream applications.

Finally, **Chapter 8** concludes with the key findings and contributions of this thesis, and discusses the limitations of the proposed approaches, suggesting potential directions for future work.

1.2.3 Research Contributions

This thesis presents a number of original contributions, summarised below:

(Ch.3) The first multi-precision CNN cascade in the literature, combining the performance benefits of quantisation and dynamic (input-dependent) inference. Coupled with a novel FPGA-based CNN accelerator that is able to scale its performance as the wordlength decreases, as well as a newly formulated multi-objective Design Space Exploration methodology tailoring the hardware architecture to the target CNN-FPGA pair, the proposed approach is able to improve inference efficiency, while considering application-specific requirements.

(Ch.4) A novel iterative approximation approach for LSTMs, combining low-rank approximation and parameter pruning with progressive inference. Supported by a novel FPGA-based LSTM accelerator and an automated co-design framework that jointly optimises the algorithmic and hardware configuration of the system to the target LSTM-FPGA pair, the proposed method improves inference efficiency, while considering application-specific requirements.

(Ch.5) A novel region-selection methodology for UAV-based vehicle detection, incorporating prior domain knowledge and additional sensory information to dynamically reduce inference latency in an altitude-aware input-dependent manner. The development of this methodology was enabled by a new UAV-based vehicle detection dataset, named *CyCAR*, offering rich sensory data to enable the development of data-driven approaches in the field.

(Ch.6) A novel vision-based robot navigation methodology, based on a data-driven representation approximating distance sensor values by implicitly extracting and exploiting spatio-temporal information, able to improve navigational accuracy and remain adaptable without any compromise in inference latency. The development of this method relied on a novel self-supervised data collection and annotation methodology, minimising human effort; leading to a new large-scale UAV navigation dataset for indoor environments, named *SelfNav*.

(Ch.7) A novel data-driven methodology for occlusion handling in depth and semantic data, based on approximating artificial object-centric views, enabled by a new hybrid dataset named *UnScene*. In combination with a novel multi-view reconstruction pipeline, facilitating progressive refinement through the informed fusion of measured and predicted information to a persistent representation, the proposed method improves task efficiency in 3D reconstruction.

1.3 Publications

The original contributions of this thesis have been published in the following peer reviewed journals, conference and workshop proceedings:

Journals:

(CEM 2020) Alexandros Kouris, Stylianos I. Venieris, Michail Rizakis and Christos Bouganis, 2020. Approximate LSTMs for Time-Constrained Inference: Enabling Fast Reaction in Self-Driving Cars. IEEE Consumer Electronics Magazine, 9(4); [116].

(CSUR 2018) Stylianos I. Venieris, Alexandros Kouris and Christos-Savvas Bouganis, 2018. Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions. ACM Computing Surveys, 51(3); [117].

Conference Proceedings:

(DATE 2020) Alexandros Kouris, Stylianos I. Venieris and Christos-Savvas Bouganis, 2020. *A throughput-latency co-optimised cascade of convolutional neural network classifiers.* In Design, Automation & Test in Europe Conference & Exhibition, IEEE; [118].

(IROS 2019) Alexandros Kouris, Christos Kyrkou and Christos-Savvas Bouganis, 2019. *Informed region selection for efficient UAV-based object detectors: altitude-aware vehicle detection with CyCar dataset.* In IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE; [119].

(ISVLSI 2019) Alexandros Kouris, Stylianos I. Venieris and Christos-Savvas Bouganis, 2019. *Towards Efficient On-Board Deployment of DNNs on Intelligent Autonomous Systems.* In IEEE Computer Society Annual Symposium on VLSI, IEEE; [120].

(FPL 2018) Alexandros Kouris, Stylianos I. Venieris and Christos-Savvas Bouganis, 2018. *CascadeCNN: Pushing the Performance Limits of Quantisation in Convolutional Neural Networks.* In International Conference on Field Programmable Logic and Applications, IEEE; [121].

(IROS 2018) Alexandros Kouris and Christos-Savvas Bouganis, 2018. *Learning to fly by myself: A self-supervised CNN-based approach for autonomous navigation.* In IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE; [122].

(SysML 2018) Alexandros Kouris, Stylianos I. Venieris and Christos-Savvas Bouganis, 2018. *CascadeCNN: Pushing the Performance Limits of Quantisation.* In Conference on Systems and Machine Learning; [123].

(ARC 2018) Michail Rizakis, Stylianos I. Venieris, Alexandros Kouris and Christos-Savvas Bouganis, 2018. *Approximate FPGA-based LSTMs under computation time constraints.* In International Symposium on Applied Reconfigurable Computing, Springer; [124]. [**Best paper award candidate.**]

Workshop Proceedings:

(EMDL 2018) Stylianos I. Venieris, Alexandros Kouris, and Christos-Savvas Bouganis, 2018. *Deploying Deep Neural Networks in the Embedded Space.* In International Workshop on Embedded and Mobile Deep Learning, MobiSys; [125].

Contributed Datasets:

(SelfNav) Indoor Robot/UAV Navigation Dataset, supporting (IROS 2018).

(CyCAR) UAV-based Vehicle Detection Dataset, supporting (IROS 2019).

(UnScene) Occlusion-free extension of YCB-Video Dataset's [126] annotations (to be released).

While not described directly in this thesis, the following publications reflect to work conducted during the same period, that has inspired many of the presented discussions:

(ECCV 2022) Alexandros Kouris, Stylianos I. Venieris, Stefanos Laskaridis, and Nicholas D. Lane, 2022. *Multi-Exit Semantic Segmentation Networks.* In European Conference on Computer Vision, Springer; [127].

(EMDL 2021) Stefanos Laskaridis, Alexandros Kouris, and Nicholas D. Lane, 2021. *Adaptive inference through early-exit networks: Design, challenges and directions.* In International Workshop on Embedded and Mobile Deep Learning, MobiSys; [128].

Finally, Table 1.1 provides an overview of this thesis, indicating the focus of each chapter in terms of the targeted application and underlying workload, examined models, datasets and deployment platforms; and their mapping to the publications listed above.

Table 1.1: Summary of the focus of each Chapter in this Thesis.

Ch.	Focus	Deep Learning Task	Relevant Application	Examined Models	Examined Datasets	Target Platform	Related Publications
1	— *Introduction* —						(EMDL 2018),(ISVLSI 2019)
2	— *Background* —						(CSUR 2018)
3	Computation Layer	Image Classification	Scene Recognition	CNNs	ImageNet, Places365	SoC FPGA	(FPL 2018),(DATE 2020)
4	Comp. and Model Layer	Video Classification	Autonomous Driving	LSTMs	BDDV	SoC FPGA	(ARC 2018),(CEM 2020)
5	Model Layer	Object Detection	UAV-based Vehicle detection	Faster R-CNN	DOTA, CyCAR*(new)*	Embedded GPU	(IROS 2019)
6	Data Layer	Image Regression	Robot/UAV Navigation	Two-stream CNNs	SelfNav*(new)*	Embedded GPU	(IROS 2018)
7	Task Layer	Depth Completion	3D Reconstruction	Encoder-Decoder CNN	YCB-Video, UnScene*(new)*	Desktop GPU	In preparation
8	— *Conclusion* —						

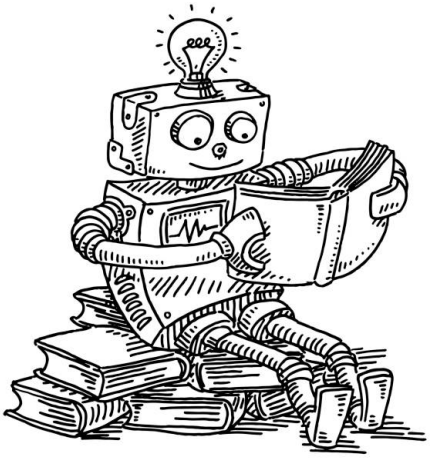


Image credits: https://www.istockphoto.com/.

2 Background

Contents

2.1 Overview 18
2.2 The Deep Learning Era of Computer Vision 19
2.2.1 The Computer Vision (R)evolution 19
2.2.2 Deep Learning Principles 25
2.2.3 Re-formulating Visual Perception through Deep Learning 32
2.3 Embedded Systems 40
2.3.1 Application-Specific Integrated Circuits (ASICs) 41
2.3.2 General Purpose Graphics Processing Units (GPGPUs) 41
2.3.3 Field-Programmable Gate Arrays (FPGAs) 43
2.4 Efficient On-device Inference 45
2.4.1 Key Performance Indicators, Beyond Accuracy 46
2.4.2 Efficient Model Design 47
2.4.3 Model Approximations 48
2.4.4 Dynamic Inference 51
2.5 Robot Visual Intelligence 53

Keywords: *Computer Vision, 3D Geometry, Deep Learning, CNNs, LSTMs, Classification, Regression, Detection, Segmentation, Embedded Systems, System-On-Chip, GPUs, FPGAs, Efficient Inference, Quantisation, Pruning, Compression, Dynamic Inference, Robot Vision*

2.1 Overview

This chapter summarises the necessary background information for the succeeding chapters of this thesis, focusing on the areas of Computer Vision, Machine/Deep Learning and Efficient Inference, Embedded Systems, and Robot Vision. The provided information is by no means exhaustive; instead it is highly-selective, focusing on a subset of topics that will be visited in later chapters of this thesis. The aim of this chapter is to render this manuscript as self-contained as possible to the reader. The provided references offer a more in-depth analysis of the respective subjects.

In more detail, across all chapters of this thesis, the problem setting assumes the availability of visual inputs from the target domain (sometimes accompanied by other modalities). The technical contributions of this thesis, propose (or employ) processing pipelines based mostly on deep learning models and 3D geometry, that consume such visual inputs to generate predictions for different perception tasks. As such, Sec. 2.2 presents some background information on visual sensing and 2D/3D representations, followed by a brief history of computer vision and its evolution towards data-driven approaches. Subsequently, the internal workings and underlying workloads of popular deep learning models and architectures that are adopted in the thesis are introduced, along with the training and evaluation process for the downstream tasks that are later explored, such as classification, regression, detection and segmentation.

The aim of this thesis is to bridge the gap between highly-accurate models or algorithms that pose significant computational requirements, and resource-constrained platforms available for inference in the embedded space. In this direction, Sec. 2.3 provides more details about the compute platforms most commonly met in this landscape, that will also be used for deployment of the proposed approaches. Additionally, important metrics of interest to characterise deep learning systems beyond accuracy in the embedded space, and relevant techniques applied for efficient model inference are briefly reviewed in Sec. 2.4.

Finally, the main motivation behind all chapters of this thesis is to address the unique challenges of efficient deployment of deep learning models on mobile robot applications. Additionally, some of the contributions exploit the unique opportunities for efficiency through customisation offered in robot-related usecases. The above are discussed in Sec. 2.5.

This chapter is partly based on a survey paper co-authored with Dr. Stylianos I. Venieris and Prof. Christos-Savvas Bouganis: (CSUR 2018).

2.2 The Deep Learning Era of Computer Vision

2.2.1 The Computer Vision (R)evolution

Computer Vision (CV) focuses on capturing and *understanding* the world through visual (image or video) data, a task also known as *visual perception* [129]. Similarly to human vision, two main components are required in a machine vision system: a *sensing device* mimicking the human eye and an *algorithm* mimicking the brain's visual function.

Sensing Devices and Visual Data

The monocular camera is the primary and most commonly used visual sensor. A camera C essentially provides a mapping (projection) from the 3D world (object space) to a 2D image plane. *Image formation* refers to a set of radiometric and geometric processes that facilitate this projection. In its simplest version, the output of this process is a *grayscale* image that can be represented as a function $\mathbf{I}(x, y) : \mathbb{R}^2 \rightarrow \mathbb{R}$, essentially representing a discrete Cartesian grid of $H \times W$ discrete *pixel* values, with each picture element (pixel) denoting the *light intensity* of the corresponding location in the image.

In computer vision, different geometric projection models are formulated, offering a trade-off between simplicity and geometric consistency of the resulting representation. Assuming a 3D point expressed in homogeneous coordinates[1] as $\mathbf{p} = [X, Y, Z, 1]^\mathsf{T}$ in the real-world, its projection $\pi(\cdot)$ on the image plane $\mathbf{u} = [x, y, 1]^\mathsf{T}$ relies on the camera *intrinsic parameters*[2]. As such, assuming no occlusions, the light intensity of the point $\mathbf{p}$, will be captured in the image at pixel coordinates $\mathbf{u}$, following the transformation:

$$\mathbf{u} = \pi(\mathbf{p}) = \frac{1}{Z} \cdot \mathbf{K} \cdot \mathbf{p} \tag{2.1}$$

where $\mathbf{K} \in \mathbb{R}^{3\times3}$ denotes the camera intrinsic matrix (a fourth column of zeros is appended to $\mathbf{K}$ to facilitate the adoption of homogeneous coordinates). Under the *perspective projection* model (Fig. 2.1a), $\mathbf{K}$ is defined as:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.2}$$

[1] Homogeneous coordinates comprise a redundant representation, with one additional dimension, that is commonly employed in 3D geometry in place of the Cartesian representation due to its flexibility and properties.

[2] This assumes that both $\mathbf{u}$ and $\mathbf{p}$ are expressed with respect to the same *camera* coordinate frame.

where f_x, f_y denote the focal length of the camera and c_x, c_y the coordinates of the principle point c on the image plane, assuming a left-hand coordinate system. The Field-of-View (FoV) of the camera, describing the angular extent of the scene captured by the camera (in degrees), can be derived for the vertical direction as $FoV_x = 2 \cdot \arctan(\frac{d_x}{2 \cdot f_x})$, where d_x is the physical height of the camera sensor, and accordingly for the width. These parameters are usually provided by the camera manufacturer, but can also be estimated through a process called *calibration* [130]. Simplifications of this model include expressing f_y with respect to f_x given the sensor *aspect-ratio* a as $f_y = a \cdot f_x$, or defining the optical centre at the center of the image $(c_x, c_y) = (W/2, H/2)$.

A more simplified projection model, termed *orthographic* $\pi_o(\cdot)$ (Fig. 2.1b), faithfully preserves the spatial components of 3D points while dropping their depth $(X, Y) \rightarrow (x, y)$. This can be expressed as a corner case of the perspective model, where the camera has disproportionate focal length to the object size, making all projection rays parallel to each other, also called the *camera-at-infinity* model. In this case, the projection is simplified as:

$$\mathbf{u} = \pi_o(\mathbf{p}) = \mathbf{K}_o \cdot \mathbf{p} \tag{2.3}$$

where the matrix $\mathbf{K}_o$ is solely parametrised by a uniform scaling factor s across height and width, that can be optionally adopted (or set to 1) as:

$$\mathbf{K}_o = \begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.4}$$

Although in the context of this thesis the internals of camera sensors and photometric methodologies (broadly discussed in [131]) are not visited, from a programming perspective each pixel intensity is represented by an unsigned *int8* value $\in [0, 255]$. Consequently, a grayscale image can be represented as a 2D matrix of the pixel values, the dimensionality of which is referred to as *image resolution*.

For colour images, each pixel is usually represented by three such matrices named *channels*. In the most commonly used *RGB system*, each channel represents the intensity of a specific colour in each location, namely Red, Green, Blue (RGB): $\mathbf{I}(x, y) = [\mathbf{R}(x, y), \mathbf{G}(x, y), \mathbf{B}(x, y)]$; while other colour systems exits such as HSV employing a Hue, Saturation and Lightness representation instead.

During this projection the depth dimension is lost; hence, monocular cameras are not able to measure *pixel distance* i.e. the distance between corresponding points on the camera plane and on

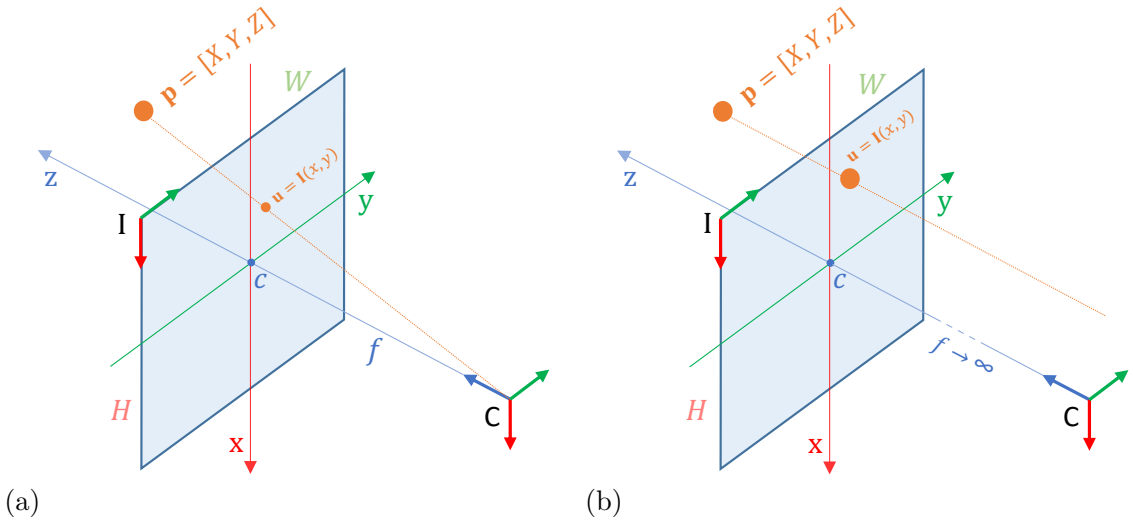


Figure 2.1: Camera Projections, through the: (a) Perspective and (b) Orthographic model.

objects in the scene. More complex sensing devices can be used to enable (visual) *spatial perception*. Among others, RGB-D cameras are the mostly widely adopted to date, due to their ability to provide a dense (per-pixel) depth measurements. The dominant RGB-D camera technology relies on *structured light projection*, where the sensor emits an Infra-Red (IR) light pattern, with each reflection on objects being measured by an IR camera, filtering out other light frequencies. As a result, in RGB-D cameras a fourth channel which captures the depth of each pixel at metric scale (called a *depth map* $\mathbf{D}(x, y) : \mathbb{R}^2 \rightarrow \mathbb{R}$) and is pixel-wise aligned to the RGB, is appended to the output image. This is also referred to as a 2.5D representation, due to its partial ability to capture 3D information. Depth data are usually represented by larger-granularity 16-bit unsigned data types. The encapsulation of the depth measurements, allows the definition of the back-projection operation $\pi^{-1}(\cdot)$, transforming 2D points on the image-plane to 3D points in the real-world space as:

$$\mathbf{p} = \pi^{-1}(\mathbf{u}) = [\mathbf{D}(\mathbf{u}) \cdot \mathbf{K}^{-1} \cdot \mathbf{u},\ \mathbf{D}(\mathbf{u})] \quad \text{and} \quad \mathbf{p} = \pi_o^{-1}(\mathbf{u}) = [\mathbf{K}_o^{-1} \cdot \mathbf{u},\ \mathbf{D}(\mathbf{u})] \tag{2.5}$$

for the perspective and orthographic case, respectively.

The obtained depth data can be represented in different 3D representations [132], according to the needs of the target application [133]. Among others, these include:

- Point-clouds: represent structure as an unordered set of points in 3D space, each defined by its $[X, Y, Z]$ coordinates denoting its physical location w.r.t. a reference frame. Point-clouds are trivial to generate, fuse multi-view data and process. However, they lack regularity in their representation often resulting to computational challenges.

- Voxel-grids: consist of discrete volumetric elements (voxels), organised in a 3D occupancy grid. Voxels can be considered the 3D equivalent of pixels in an image. Due to their discretisation, voxel-grids sacrifice a level of shape detail in favour of providing a highly-structured (regular and uniform) representation.

- Octrees: form a more memory efficient voxelised representation, where voxel-resolution is dynamically adjusted across the volume, by employing a hierarchical tree-structure. Octrees progressively divide each occupied voxel to eight octants: higher-resolution sub-voxels recursively instantiated to best capture object structure; while empty/similar areas are represented by lower-resolution (larger) voxels, leading as a result to increased representation complexity in favour of memory efficiency.

- (Truncated) Signed Distance Functions (TSDFs): form a volumetric representation where each voxel does not denote occupancy in terms of a discrete {0,1} value, rather encodes the signed distance to its nearest surface (usually truncated above a certain value). This allows for sub-voxel resolution in the representation of 3D structures, with negative values signifying voxels that lie within the shape, striking a balanced shape detail/efficiency trade-off.

- Meshes: are a surface-based representation, decomposing 3D structures to a combination surface-elements (surfels). Surfels are typically polygon faces (and most commonly triangles) stitched together to reconstruct the external cell of each object. Meshes comprise a less efficient representation both compute- and memory-wise, offering however the greatest level of detail and are often used for visualisation purposes.

Such metric 3D representations can effectively accumulate information from multiple camera views, through fusion processes adhering to the principles of 3D geometry [134]. When the camera C is moving in the scene, 3D points are best described by a static *world* coordinate frame W, in which case they are denoted as ${}^{W}\mathbf{p}$. To calculate their projection to the current camera frame, an extra transformation ${}^{C}\mathbf{T}_{W}$ is required, capturing the translation $\mathbf{t}$ and rotation $\mathbf{R}$ between the camera C and world W coordinate frames. This transform is commonly known as extrinsic parameters, and can be modelled (among others) in a matrix form within the Special Euclidean group as:

$$SE(3) = \left\{ \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^{\mathsf{T}} & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} | \mathbf{R}^{3\times3} \in SO(3), \mathbf{t} \in \mathbb{R}^{3} \right\} \text{[3]} \tag{2.6}$$

[3] SO(3) denotes the Special Orthogonal group, a subset of orthogonal matrix group O(3) capturing all possible proper rotations in 3D space, through a rotation matrix $\mathbf{R}$ as $SO(3) = \left\{ \mathbf{R} \in \mathbb{R}^{3\times3} | \mathbf{R}\mathbf{R}^{\mathsf{T}} = \mathbf{I}, det(\mathbf{R}) = 1 \right\}$.

The forward projection model is then generalised to:

$$\mathbf{u} = \frac{1}{Z} \cdot \mathbf{K} \cdot {}^{C}\mathbf{T}_{W} \cdot {}^{W}\mathbf{p} \tag{2.7}$$

and considers both the camera intrinsic and extrinsic parameters. In contrast to intrinsics, extrinsic parameters vary over time due to robot motion, and are estimated on-the-fly according to the setting. Back-projection transformations and the orthographic projection counterparts can be equivalently derived.

Computer Vision Algorithm Pipeline

After *sensing* (i.e. acquiring visual data), *perception* (i.e. the task of understanding the visual input) takes place. With the emergence of AI, the visual perception pipeline has rapidly evolved over the past decades.

Input data are commonly processed and analysed by a sequence of modules, starting from a *pre-processing* algorithm to standardise the input image (i.e. resize, normalise, apply geometric and colour transforms etc). Standardisation plays a key-role in the effectiveness of subsequent steps, by reducing the diversity of the input data and hence the complexity of the problem.

The standardised image is then fed to a Feature Extraction module. *Features* are measurable properties or characteristics that are evident in or can be extracted from the input, such as specific shapes, colours or patterns. Adhering to the *no free lunch theorem* [135], there is no single set of features that works best across all tasks. Conversely, a different set of features should be selected, aiming to maximise the descriptiveness of the representation in respect of the target application.

In traditional CV pipelines (Fig. 2.2a), human experts with domain knowledge manually select and engineer the features for each application according to their intuition. General guidelines suggest that the selected features need to be easy to identify, track and compare, and demonstrate consistency across different lighting conditions, view-angles, scales and affine transformations. Along these lines, a wide variety of handcrafted features have been proposed in the literature over the years, including: HOG [136], SIFT [137] and SURF [138].

Subsequently, a task-specific algorithm receives the extracted feature representations and processes them to yield a final prediction. For example, in image recognition, a *classification* algorithm is employed to assign a *class label* from a predefined set to the input image, by examining

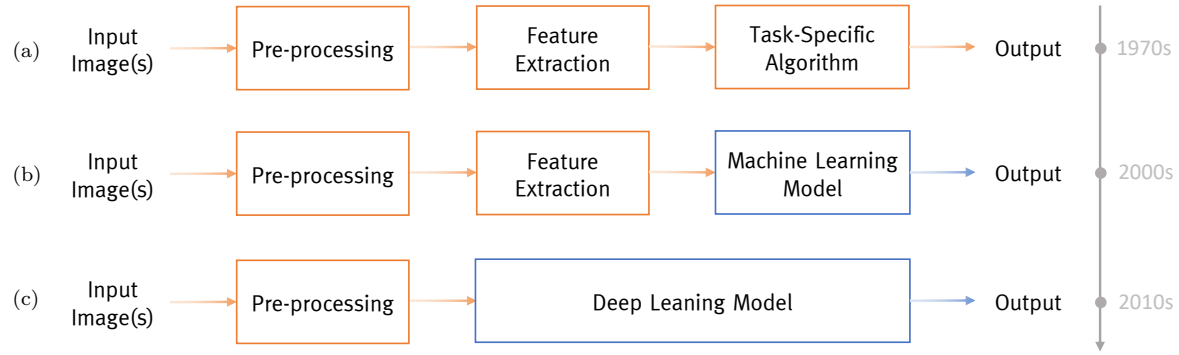


Figure 2.2: Evolution of the Computer Vision pipeline: From traditional to Deep Learning.

the extracted feature vector. To achieve that, the selected features need to be descriptive enough to facilitate the distinction between different classes.

Manually developing such task-specific algorithms with the ability to process highly-dimensional feature vectors can be tedious and lead to sub-optimal solutions. Nonetheless, performing classification based on computerised data representations is far from intuitive for humans. To remedy this challenge, during 1990s Machine Learning[4] was injected in the pipeline (Fig. 2.2b) to replace the explicit task algorithm with a learnable model, such as a Support Vector Machine [139], Decision Tree [140] or Adaboost [141]. The main contribution of these modules was to automatically weight the contribution of each hand-crafted feature to yield a final prediction [142] for the end-task.

During the ML era, feature selection remained a manual process, until the revolution of deep learning [143] in early 2010s. Deep learning replaces both the feature extraction and task-specific modules with an *end-to-end* learnable model (Fig. 2.2c), typically a Deep Neural Network [18]. Consequently, no handcrafted feature selection and engineering is required. Instead, the neural network automatically extracts descriptive features through a paradigm termed *representation learning* [144], and jointly learns their importance to the end task. This is not to say that the notion of a feature extractor and a classifier is completely abolished. Instead, these units still distinctively exits [145], but are tight together within a deep learning model that can be holistically optimised by supervising its final task output.

[4] Also referred to as Statistical Inference or Statistical Learning back then [8].

2.2.2 Deep Learning Principles

Artificial Neural Networks (ANNs)

Loosely inspired from the human brain, neural networks consist of a large set of *neurons* being organised into *layers* and interconnected with each other through *connections* forming a *network*, where information can flow from the input to the output layers [146]. All intermediate layers, between the input and output ones, are named *hidden layers*. Given that the input layer essentially detects patterns on the input data, hidden layers are responsible to detect patterns within these detected patterns. This gives neural networks the ability to extract complex features by hierarchically combining simpler ones.

Although their name may falsely suggest otherwise, neural networks are not aimed to directly emulate the human brain, rather mimic just some limited aspects of the human understanding about its structure and functionality [147]. Instead, the study of bio-inspired AI systems branches out focusing on spiking neural networks [148] and spiking computing [149]. Under this paradigm, each neuron is activated independently, firing a spiky pulse signal that is propagated to all its immediately connected neurons, which are consequently activated triggering a flow of signals across the network.

In contrast, machine learning adopts a more computerised flavor of *artificial neural networks* (ANNs), where computation is more structured and information uniformly propagates across the layers. As depicted in Figure 2.3b, each neuron receives a number of inputs in a vectorised representation $\mathbf{x}$ through direct connections with the neurons of its previous layer, and computes a weighted summation to produce a single real-valued output y. This dense connectivity between the neurons of subsequent layers led to the adoption of the term *Fully-Connected* (FC) layers [150], which is widely used today. The employed weights $\mathbf{w}$ and bias term b form learnable parameters corresponding to each connection and neuron, respectively. As such the computation taking place within a neuron with N_{in} incoming connections can be formalised as a dot product [151]:

$$y = \sum_{i=1}^{N_{in}} \mathbf{w}_i \cdot \mathbf{x}_i + b. \tag{2.8}$$

A nonlinear activation function is commonly applied on the output of each neuron, with popular options being:

$$\text{sigmoid}(y) = \frac{1}{1+e^{-y}} \in [0,1], \qquad \text{or} \qquad \tanh(y) = \frac{e^{y}-e^{-y}}{e^{y}+e^{-y}} \in [-1,1]. \tag{2.9}$$

Introducing non-linearity between layers is critical for the representation power of the model, as otherwise the sequence of linear (weighted sum) operations would become equivalent to a single-layer linear model. As such, the computation for the whole j-th layer comprising N_{out} neurons can be described as:

$$y_j = \text{nonlin}(\sum_{i=1}^{N_{in}} \mathbf{W}_{i,j} \cdot \mathbf{x}_i + \mathbf{b}_j), \qquad \forall j \in [1, N_{out}] \tag{2.10}$$

It is noteworthy that in the vanilla version of NNs, connections between neurons of the same layer are not established. Neural Networks with a large numbers of hidden layers (i.e. more than four) are given the term *Deep Neural Networks* [152]. The rapid evolution of such models, offering an unprecedented in representation learning, gave birth to the field of deep learning [143].

Convolutional Neural Networks (CNNs)

The most prominent deep learning models to date are *Convolutional Neural Networks* (CNNs) [18][153], which will be widely used across this thesis. CNNs comprise deep neural networks with neurons organised in a three-dimensional topology within each layer, to facilitate the extraction of powerful representations from spatial data, such as images[5]. The weights of each CNN layer demonstrate considerably lower spatial dimensions compared to the respective input data, and thus the same weight are re-used across different locations. This is achieved by striding though the image exploiting the locality in the input data to recognise patterns in a translation-invariant manner.

The main building block of CNNs is the *convolutional layer* which acts as a "feature finder" that slides over the input image [129]. Convolutional layers essentially use learnable weights to provide a transformation between the input and output *feature volumes*. Several convolutional layers are stacked successively to extract more complex features, similar to ANNs. The output of each convolutional layer, however, is a 3D feature volume consisting of a N_{out} 2D *feature maps* or *activations*. Each activation map is generated by convolving[6] the feature volume of the previous layer with a three-dimensional convolutional filter, also called a *kernel*, consisting of learnable parameters (weights). Stacking multiple kernels together generates several output channels, that are appended to form the 3D output of the layer.

[5] Although images can be flattened to 1D vectors and processed by ANNs, this would lead to a loss of spatial information (i.e. the structure of visual patterns across the 2D plane).

[6] The operation implemented by most Deep Learning libraries is essentially cross-correlation, as the kernel is not flipped relatively to the input in the way that the original convolution suggests. However the term "convolution" is dominantly used by the ML community for this operation.

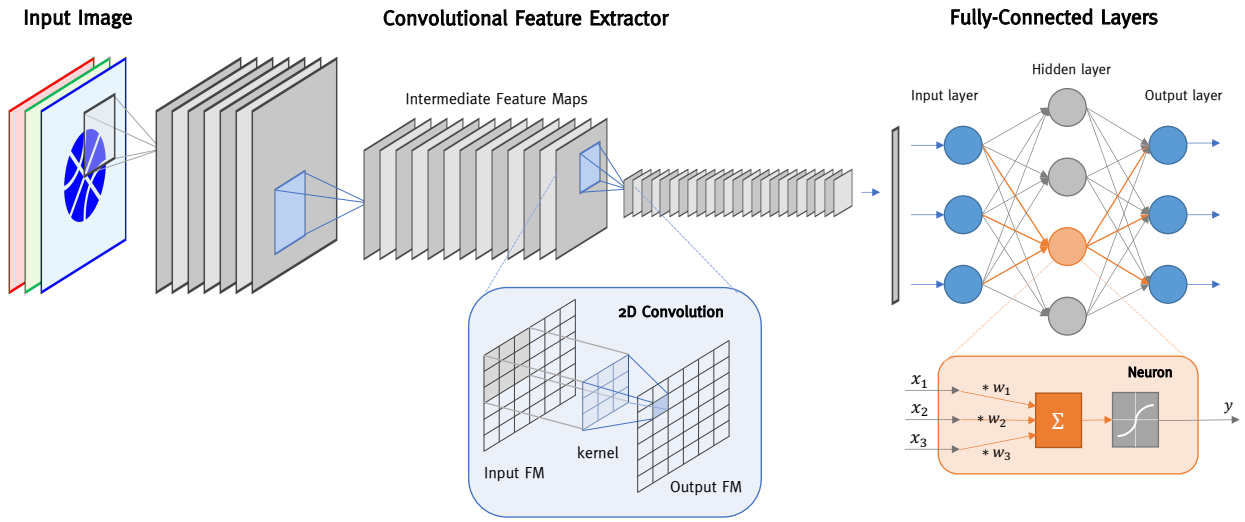


Figure 2.3: (a) Convolutional Feature extractor and (b) Fully-Connected layers; forming a CNN.

By flattening the filter and corresponding feature map window, the computation in each convolutional location is equivalent to that of Eq. 2.8. This operation is performed in a "sliding window" manner, usually covering all spatial locations on the input volume with resolution $H \times W$, with a pre-determined *stride* size controlling the potential overlap between neighbouring windows. Each convolutional layer is further parametrised by the number of channels of its input feature volume N_{in} (also determining the number of channels for each filter), the number of output channels N_{out} (determining the number of filters applied), as well as the kernel size $K_H \times K_W$ (defining the spatial dimensionality of each kernel weight matrix), the convolution strides $S_H \times S_W$ (determining the amount by which each filter slides over the input feature maps) and padding size Z (determining the cardinality of additional zero elements appended around the border of the input volume, most commonly used to preserve the spatial dimensionality between the input and output volumes).

Overall, the computation of a convolutional layer can be summarised as:

$$\mathbf{Y}_j = \sum_{i=1}^{N_{in}} (\mathbf{X}_i * \mathbf{K}_{i,j}) + \mathbf{b}_j, \qquad \forall j \in [1, N_{out}] \tag{2.11}$$

where $*$ denotes the 2D convolution operation, $\mathbf{X}_i \in \mathbb{R}^{H \times W}$ denotes the i-th input feature map, $\mathbf{Y}_j$ the j-th output activation map, $\mathbf{K}_{i,j} \in \mathbb{R}^{K_H \times K_W}$ the filter kernel corresponding to the i-th input and j-th output channels, and $\mathbf{b}_j \in \mathbb{R}^{N_{out}}$ the layer's bias vector.

As in ANNs, to stop the convolution operations of subsequent layers from collapsing to an equivalent single linear operation severely limiting the descriptive capability of the feature

extractor, nonlinear *activation functions* are introduced after each convolution. For CNNs, the most common option to date is the Rectified Linear Unit (ReLU) [154], which is performed in an element-wise manner on each output activation:

$$\text{ReLU}(y) = \max(0, y) \in [0, +\infty). \tag{2.12}$$

Going deeper in the network, the spatial dimensions of the feature volume typically shrink, while the number of activation/filter channels increases (Fig. 2.3a). This can be attributed to the hierarchical representation extracted by CNNs: the filters of shallow layers are looking for more general, primitive features, such as lines, curves and edges. If such features are found in the input image, their locations are reported by activating the corresponding pixels of the respective output feature maps. Subsequent layers combine these primitive features to identify more complex/abstract patterns such as specific shapes, corners and blobs, which become increasing complex and high-level such as text, faces, object parts and even objects[7].

This channel-wise inflation can be achieved by employing a larger number of filters in a convolutional layer, while spatial down-sampling is traditionally performed by *pooling layers*. These layers can resize the feature volume by applying summary statistical functions such as $\max(\cdot)$ or $\text{average}(\cdot)$ along its spatial dimensions, in a non-overlapping windowed fashion. Apart from reducing the computational complexity of subsequent layers, pooling exposes a larger portion of the input image to the filters of deeper layers, essentially increasing the *receptive field*. This enables the extraction of global context and more abstract representations, and facilitates scale/translation invariance on the learning process. Alternatively, the feature volume can be shrunk by selecting larger stride values in a convolutional layer, leading to a simpler and more computationally-uniform (and thus hardware-friendly) solution [155].

The output of the convolutional feature extractor is flattened to a feature vector and fed to a feed-forward (or recurrent) neural network, that is responsible to execute the downstream task at hand (Fig. 2.3b). The most common choice for establishing a feed-forward model is to append a few fully-connected layers, described earlier in this section.

[7]CNN feature maps are not necessarily mapped to explicitly human-recognisable features. They rather encapsulate more abstract but meaningful 2D representations, which enable the projection of the input image to a space that facilitates the target recognition task through separating the target representations.

Computationally, the feature extractor is mainly *compute-bound* dominating the workload of the CNN, while the task-specific head at the end is *memory-bound*, due to the large number and minimum re-use of its weights.

Long Short-Term Memory Networks (LSTMs)

Recurrent Neural Networks (RNNs), on the other hand, are artificial neural networks with the ability to handle sequential data by capturing long-term dependencies (i.e. information for earlier input timesteps can contribute to later predictions). In that sense RNNs are known to be *stateful*.

RNN training, however, has been prone to the vanishing/exploding gradient problems. *Long Short-Term Memory* (LSTM) Networks [156] handle this challenge by using gates to control the flow of information in and out of the model's *state*. Gates are implemented as nonlinear sigmoid$(\cdot)$ functions used to determine if the extracted information at each timestep needs to be preserved or forgotten, while ensuring that the gradients maintain within adequate scale even in long data sequences.

The core of the LSTM is the *cell state* $\mathbf{c}^{(t)}$, at which information is added or removed at each timestep through three *gates*: input, forget and output. LSTMs have a chain structure, temporally repeating the same module with different inputs while propagating the internal state, as illustrated in Figure 2.4.

The *output gate* $\mathbf{o}^{(t)}$ is responsible to determine the LSTM's prediction $\mathbf{h}^{(t)}$ for each timestep t. A tanh$(\cdot)$ operation is initially performed on the current cell state of the LSTM $\mathbf{c}^{(t)}$, which is subsequently scaled element-wise by a vector determined by the output gate. This vector is the result of matrix-vector multiplication operations between the gate's weights $\mathbf{W}_o$, the current input $\mathbf{x}^{(t)}$ and the LSTM output during the previous timestep $\mathbf{h}^{(t-1)}$. This process can be formalised as:

$$\begin{aligned} \mathbf{o}^{(t)} &= \sigma(\mathbf{W}_o^x \cdot \mathbf{x}^{(t)} + \mathbf{W}_o^h \cdot \mathbf{h}^{(t-1)} + \mathbf{b}_o) \\ \mathbf{h}^{(t)} &= \mathbf{o}^{(t)} \odot \tanh(\mathbf{c}^{(t)}) \end{aligned} \tag{2.13}$$

where $\odot$ denotes the element-wise multiplication operation (a.k.a. Hadamard product).

The *forget gate* $\mathbf{f}^{(t)}$ determines what information from previous timesteps should be dismissed from the cell state, by scaling each element of $\mathbf{c}^{(t-1)}$ with the vector output of the similar matrix-vector products, also considering the current input $\mathbf{x}^{(t)}$ and previous output $\mathbf{h}^{(t-1)}$ of the LSTM, along with the gate weights $\mathbf{W}_f$. Accordingly, the *input gate* $\mathbf{i}^{(t)}$ determines which cell state

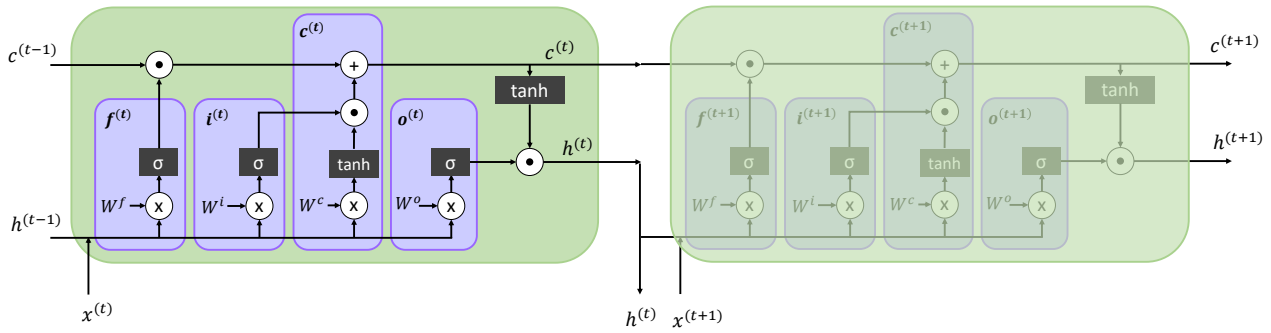


Figure 2.4: LSTM chain (and internal workings) across two timesteps of an input sequence.

values should be updated with the newly obtained information at timestep t; whereas a $\tanh(\cdot)$ layer is used to formulate the update values, considering the same two terms and corresponding weights. The update is filtered/scaled by the output of $\mathbf{i}^{(t)}$ before being accumulated to the cell state. The above steps can be formalised as follows:

$$\begin{aligned}\mathbf{f}^{(t)} &= \sigma(\mathbf{W}_f^x \cdot \mathbf{x}^{(t)} + \mathbf{W}_f^h \cdot \mathbf{h}^{(t-1)} + \mathbf{b}_f) \\ \mathbf{i}^{(t)} &= \sigma(\mathbf{W}_i^x \cdot \mathbf{x}^{(t)} + \mathbf{W}_i^h \cdot \mathbf{h}^{(t-1)} + \mathbf{b}_i) \\ \mathbf{c}^{(t)} &= \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \odot \tanh(\mathbf{W}_c^x \cdot \mathbf{x}^{(t)} + \mathbf{W}_c^h \cdot \mathbf{h}^{(t-1)} + \mathbf{b}_c)\end{aligned} \tag{2.14}$$

LSTMs can be formulated to tackle regression and classification models. In their vanilla implementation, LSTMs receive a one-dimensional input sequence. To employ LSTMs on visual recognition tasks (e.g. for video data), convolutional variants termed ConvLSTM [157], which are able to capture the underlying spatial features in multi-dimensional data such as images, have been explored in the literature; as well as CNN-LSTM combinations, where the preceding CNN acts as a feature extractor, providing a flattened *visual embedding* (i.e. feature representation of an image, mapping the visual content to a compact and dense feature vector suitable for ML tasks, by capturing meaningful information in a summarised way) from each frame to the LSTM [158].

Computationally, LSTMs are also challenged by being memory bounded due to the limited weight re-use and parallelisation opportunities, which are hindered by the complex data dependencies across timesteps.

Deep Neural Network Training and Inference

The life cycle of deep learning model evolves in two stages: *training* and *inference*.

Inference is the feed-forward process during which a deep learning model f parametrised by static weights (and biases) jointly denoted here as θ, can make a prediction y by progressively

processing previously unseen input data $\mathbf{x}$ from the input to the output layers. Inference is an online process relying on the generalisation ability of the model obtained through exposure on a vast amount of data during training, leading to summarisation/abstraction capabilities. As such, inference can take place on the data-centre; however, with the emergence of deep learning in low-latency and privacy-aware applications, this paradigm is increasingly shifting towards edge and embedded/mobile device deployment [159]. In this setting inference is mainly latency-optimised, however throughput and energy (co-)optimisation are also playing an important role.

Training, preceding inference, is the process of determining the values of learnable parameters θ for a model f. Under the *supervised learning* paradigm, which is studied in this thesis, training requires an annotated dataset D defined as set of labelled data tuples (examples) from the same domain as the target application:

$$D = \{< \mathbf{x}_1, \hat{y}_1 >, < \mathbf{x}_2, \hat{y}_2 >, ..., < \mathbf{x}_N, \hat{y}_N >\} \tag{2.15}$$

The model follows the forward inference process to yield a prediction $y_i = f_\theta(\mathbf{x}_i)$. Then a cost (error) function, can used to determine the discrepancy between the predicted and desired value, termed *loss*: $l = cost(y_i, \hat{y}_i)$. Iterating through the dataset D to find the best set of weights can be formulated as an optimisation problem:

$$\theta^* = \arg\min_\theta \sum_{<\mathbf{x},y>\in D} cost(f_\theta(\mathbf{x}), \hat{y}) \tag{2.16}$$

Although solving this problem though traditional optimisation algorithms is not computationally tractable due to the large number of free parameters, the *Gradient Descent* algorithm can be used to provide an iterative approximation of the optimal weights. This would require iterating over the dataset K times, named *epochs*, and computing the cost gradient to provide an iterative update on the weight values as:

$$\theta_{k+1} = \theta_k - \alpha \cdot \nabla_\theta l_k \tag{2.17}$$

where α is termed *learning rate* and determines the magnitude of the update (also known as *step size*). However, calculating the gradient for the whole dataset also becomes intractable given the scale of datasets employed in deep learning. Instead, mini-batch Stochastic Gradient Descent (SGD) [160] adopts an approximation that calculates the gradient of smaller subsets of the training set for each update step, named *batches* and denoted as $\mathbf{b}$:

$$\theta_k^{b+1} = \theta_k^b - \alpha \cdot \nabla_\theta l_k^b \tag{2.18}$$

Each weight update (based on a subset of the training data) is termed a training *iteration*. Several improvements have been proposed to stabilise the convergence of SGD, considering for previous [161] and/or (estimations of) future [162] gradient steps in each iteration, dynamically adjusting the learning [163] according to the gradient morphology and other criteria [164]. Additionally several alternative optimisers have been proposed in the literature, with most prominent example being the Adam optimiser [165].

Given that ground-truth labels are only provided for the output of the last layers, another computation-tractability problem arises for calculating the gradient of weights in intermediate layers, especially in the era of deep learning. *Backpropagation* [166][167][168] is employed to remedy this issue by progressively propagating the cost backwards from the output to the input layers.

When training deep neural networks, several other techniques have been proven effective and are thus commonly used [169] to aid generalisation and convergence. These include the addition of *dropout layers* [170], *batch normalisation* [171] and *weight decay* [172].

To date, training is most-often an offline throughput-oriented process, typically taking place in the datacentre and exploiting the profusion of computational power and storage available [63].

2.2.3 Re-formulating Visual Perception through Deep Learning

This section discusses the formulation of the primitive computer vision tasks and how deep learning methods approach them, achieving state-of-the-art solutions [173].

Image Classification

Image classification, also known as *image recognition*, aims to categorise an input image as a whole into a discrete predefined set of possible classes.

To facilitate this, the output dimensionality of the last Fully-Connected layer equates the number of classes of the target task N_{class}. Additionally, an extra layer termed softmax is appended at the end. Softmax is essentially a generalisation of the sigmoid$(\cdot)$ non-linear function for multi-class classification problems. Hence, softmax forces the output "logits" $(\mathbf{y})$ of the last layer to sum to 1, reassembling a probability distribution $\mathbf{p}$ across candidate classes:

$$\mathbf{p}_i = \text{softmax}(\mathbf{y}_i) = \frac{e^{\mathbf{y}_i}}{\sum_{j=1}^{N_{class}} e^{\mathbf{y}_j}} \tag{2.19}$$

The most common error function for classification $\mathcal{L}_{cls}$ is the *cross-entropy* loss, which is used to quantify the difference between two probability distributions. After expressing the ground-truth label as a one-hot target distribution, cross-entropy can be calculated as:

$$\text{CE}(\mathbf{p}, \hat{\mathbf{y}}) = -\sum_{i=1}^{N_{class}} \hat{\mathbf{y}}_i \cdot \log(\mathbf{p}_i) \tag{2.20}$$

Top-K accuracy is the most commonly employed metric for classification. Assuming the existence of a labelled test set $D_{test} = \{< \mathbf{x}_1, \hat{\mathbf{y}}_1 >, < \mathbf{x}_2, \hat{\mathbf{y}}_2 >, ..., < \mathbf{x}_N, \hat{\mathbf{y}}_N >\}$ and corresponding set of model predictions $P = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_N\}$ each in the form of a probability distribution across classes, then top-K accuracy is defined as:

$$\text{topK}(D_{test}, P) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(\hat{\mathbf{y}}_i \in \arg\,\text{top}_K(\mathbf{p}_i)) \tag{2.21}$$

where $\mathbb{1}(\cdot)$ denotes the *indicator* function returning 1 if the condition in the parenthesis is satisfied and 0 otherwise; and $\arg\,\text{top}_K(\cdot)$ a function returning the indices of the K largest values in the input distribution. The most frequently used variants in the literature are top-1 and top-5 accuracy, indicating whether the correct label for an image is the highest, or within the five highest prediction probabilities.

At inference time, since the output of the model remains a distribution across classes, different metrics can be used to implicitly capture the *confidence* (or uncertainty) of each prediction. The most commonly met formulation is to treat the softmax output as a probability distribution, thus interpreting the $\max(\mathbf{p}_i)$ as the probability the model assigned to its most probable class-prediction [174]. Accordingly, other works [175] study the use of entropy:

$$\text{H}(\mathbf{p}_i) = -\sum_{i=1}^{N_{class}} \mathbf{p}_i \cdot \log(\mathbf{p}_i) \tag{2.22}$$

or other hand-crafted metrics, examining the distribution provided by the model [176].

Classification is the most broadly studied task in computer vision, which has driven the rapid progress in the field since the emergence of the first CNN deployment in a practical application in the 1990s [177]. This pioneering model, named **LeNet-5**, comprised three convolutional and two fully-connected layers (Fig. 2.5a), targeting the MNIST [178] task of hand-written digit recognition in low-resolution grayscale images.

More than a decade of slow progress later, **AlexNet** [20] was the first network to achieve state-of-the-art accuracy on large-scale image classification, surpassing classical computer vision approaches

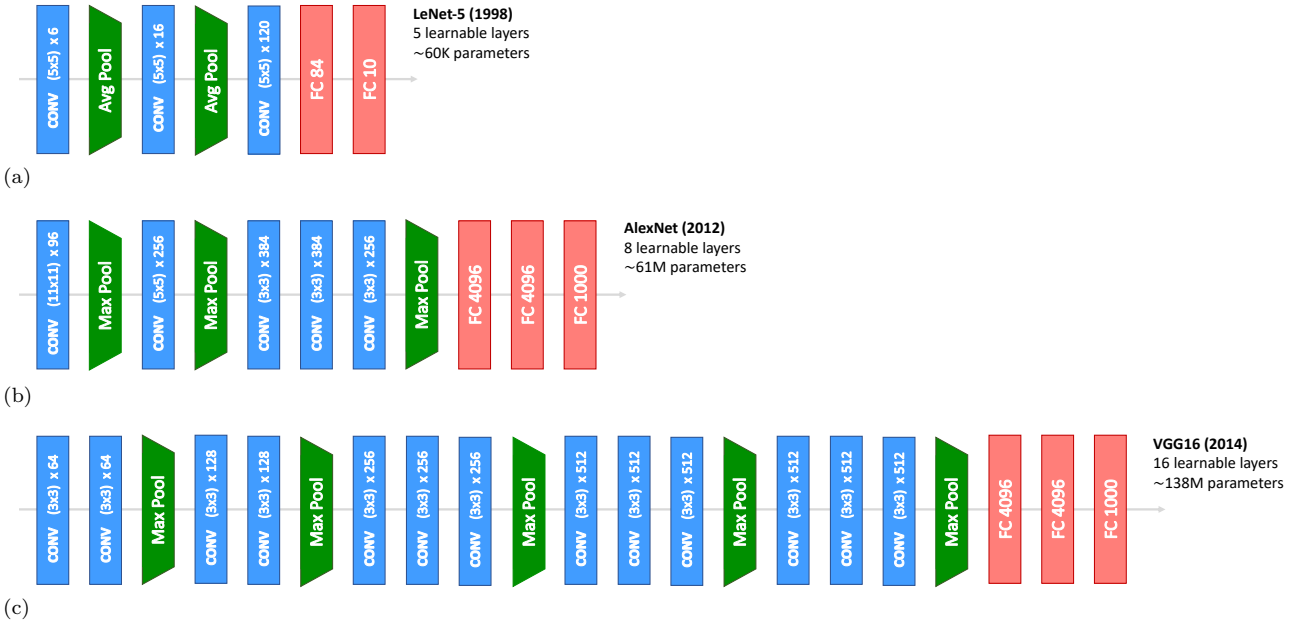


Figure 2.5: Progression of early "milestone" CNN architectures for image classification: (a) LeNet-5; (b) AlexNet; (c) VGG-16.

for the first time on the 2012 ILSVRC ImageNet challenge [21]. Conversely to MNIST, ImageNet comprises higher-resolution colour images of 1,000 object classes[8]. AlexNet was not fundamentally different from LeNet-5, although it was a deeper model comprising five convolutional and three FC layers (all with a larger number of channels), featuring as a result three orders of magnitude more trainable parameters (Fig. 2.5b). Additionally, AlexNet paved the way for popularising CNNs in computer vision, instigating a plethora of techniques used until today such as the adoption of the ReLU activation function partly mitigating vanishing gradients, dropout and weight decay to avoid overfitting, Local Response Normalisation (LRN; an ancestor of Batch Normalisation for speeding up convergence), and data augmentation techniques to aid generalisation capability.

Aiming to tightly fit in the memory of the GPUs of the date (GTX 580), one thing to note about AlexNet is the non-uniformity of its kernels, featuring diverse spatial dimensions and arbitrary channel numbers. Two years later, **VGG-16** [36] doubles up the number of parameters while expanding its depth to 13 convolutional and 3 FC layers (Fig. 2.5c), favouring regularity by adopting uniform 3x3 kernels and restricting channel width to powers of two.

[8]This competition only focuses on a subset of the ImageNet dataset, which comprises almost 22,000 classes, organised in a lexical hierarchy based on WordNet [179]. ImageNet training data contain more than 14 million annotated images, collected from online resources and labelled by humans through Amazon Mechanical Turk.

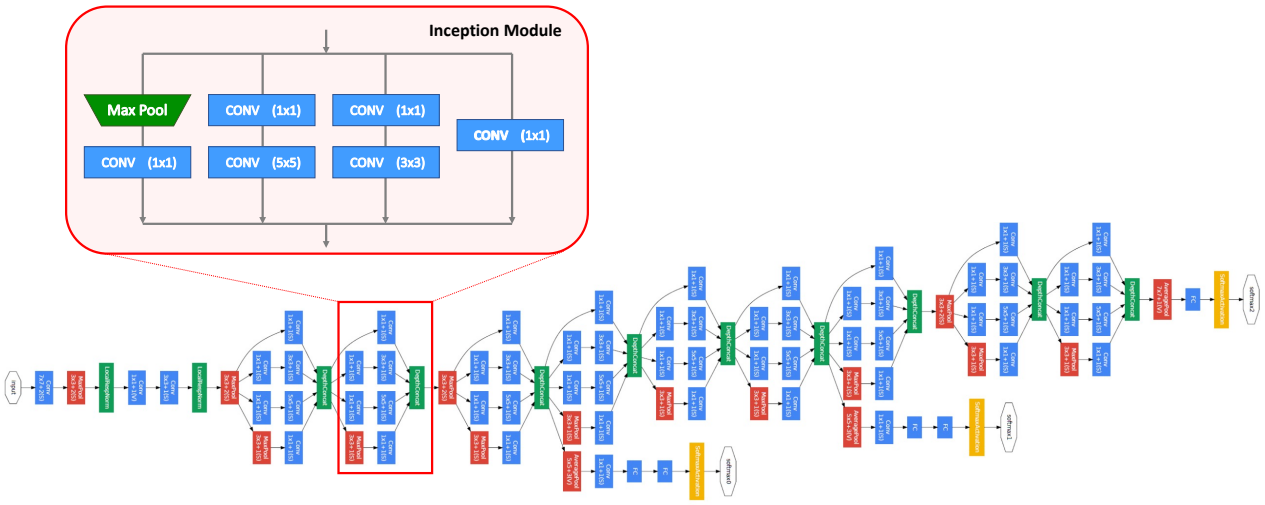


Figure 2.6: GoogleNet CNN architecture, comprising numerous Inception modules. Figure sourced and modified from original paper [180].

As can be deduced from Fig. 2.5, the early evolution trajectory of CNNs, i.e. becoming increasingly deeper and wider, comes at a non-scalable price of inflated computational and memory demands, hindering their deployability. To remedy this challenge, more complex architectures started to emerge, aiming to facilitate equally powerful feature extraction capabilities with fewer parameters.

For example, **GoogleNet** [37] features 57 convolutional and a single fully-connected layer; but reduces the number of parameters up to 20× compared to VGG-16 by introducing a multi-path layer structure termed the Inception module (Fig. 2.6). Instead of manually selecting kernel dimensions for each layer, Inception applies several operation options in parallel and concatenates their outputs. GoogleNet stacks up several Inception modules along with pooling layers and reaches almost human-level performance on the ILSVRC competition. Additionally, during training intermediate classifiers are added and jointly trained (by summing their loss terms), to alleviate the vanishing gradient issue and regularise the model.

Soon after, Residual Networks (**ResNet**) [3] were developed following a different design philosophy. By re-visiting the deeper-is-better school of thought, ResNets feature skip connections ($y(x) = L(x) + x$; where $L(x)$ denotes a block of layers of the model), offering a shortcut for gradients to directly propagate back to early layers without vanishing. This allowed training CNNs with up to 150 layers, being the first to actually surpass human-level accuracy on ImageNet. To mitigate the otherwise excessive computational overhead of that revelation, ResNets adopted

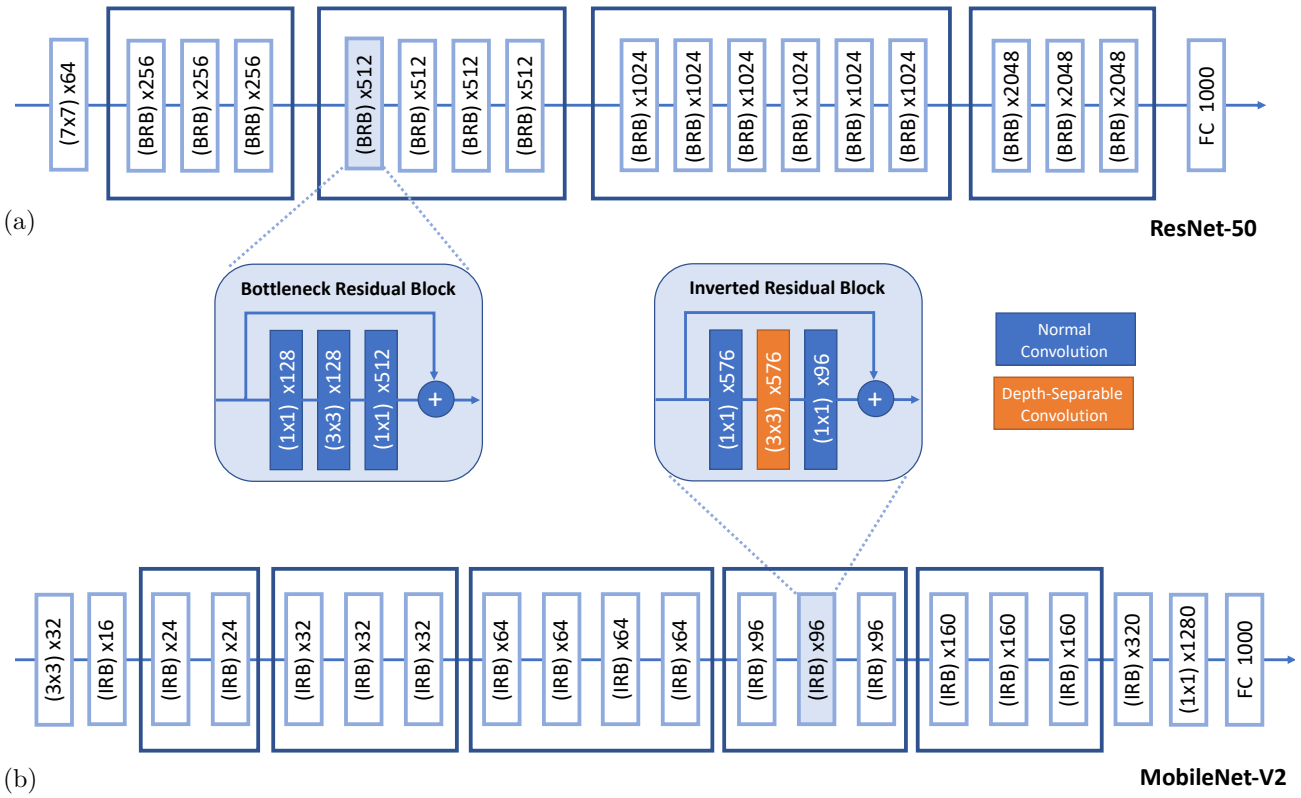


Figure 2.7: Architecture of: (a) ResNet-50 and (b) MobileNet-V2 CNNs.

the Bottleneck Residual blocks as their main component (Fig. 2.7a). Each block, begins with a 1x1 convolution radically reducing the channel dimensionality [181], before proceeding to a more computationally expensive 3x3 convolution. At the end of each block a 1x1 convolution is used to restore the activation volume dimensionality. Finally, ResNets preserve the VGG-16's regularity of kernel and channel dimensions.

Soon after, the two latter concepts were merged forming **Inception-ResNet** [180], using residual connections within the multi-scale Inception block. This allowed the GoogleNet architecture to grow deeper and enhance its learning capacity, yielding yet another accuracy boost on the ILSVRC.

Aiming to facilitate CNN deployment in the embedded and mobile landscape, the **MobileNet** [50][182] line of works was introduced. The idea of ResNet's residual connections is adopted, however MobileNet's modules named Inverted Residual Blocks apply depth-wise separable convolution to reduce the number of parameters and computational burden (Fig. 2.7b).

Since then, the development of automated design methodologies [183], and collection of further mass-scale private datasets [184], has led to the continuous advancement of CNN architectures, pushing their attainable accuracy [185] with an unprecedented pace.

Image Regression

Image regression is a computer vision task aiming to predict a task-specific continuous value from an input image. Its key difference from image classification is the domain of the output, that becomes a real number instead of a categorical class label. In that sense, regression is a more challenging problem and some works aim to re-formulate it as a classification problem by binning the continuous values into discrete ranges [186], at the cost of introducing a rounding error on the prediction.

In the original regression formulation, CNN classification architectures are re-used for regression, replacing the final classifier with a single-neuron fully connected layer, while employing a $\tanh(\cdot)$ or $\text{sigmoid}(\cdot)$ non-linearity. Regressed values are therefore usually squeezed to [-1,1] or [0,1], reducing the risk of exploding gradients.

Mean Squared Error (MSE) is a common option for regression loss:

$$\text{MSE}(y_i, \hat{y}_i) = (\hat{y}_i - y_i)^2 \tag{2.23}$$

along with Huber Loss, that is found to be less sensitive to outliers:

$$\text{Hurber}_\delta(y_i, \hat{y}_i) = \begin{cases} 1/2 \cdot (\hat{y}_i - y_i)^2 & , \text{ if } |\hat{y}_i - y_i| \leq \delta \\ \delta \cdot (|\hat{y}_i - y_i| - 1/2 \cdot \delta) & , \text{ otherwise.} \end{cases} \tag{2.24}$$

Accordingly, these error functions can be used as accuracy metrics, along with Mean Average Error $\text{MAE}(y_i, \hat{y}_i) = |\hat{y}_i - y_i|$ and Root Mean Square Error $\text{RMSE}(y_i, \hat{y}_i) = \sqrt{\text{MSE}(y_i, \hat{y}_i)}$, which are more interpretable by adopting the same units as the target task.

Due to the aggravated difficulty and cost of data annotation arising from the continuous nature of the labels in regression tasks, it is common to employ ImageNet pre-trained classification backbones as an initialisation for training regression models. This facilitates faster convergence with fewer training data, as during the first training epoch on the target task, the feature extractor is already able to provide meaningful feature representations from the inputs. This technique, known as *transfer learning* [187], essentially aims to migrate the knowledge learned from a source dataset to the target task by re-using some of its learned parameter values, either as a *frozen* feature extractor (mostly applicable if the domains are very similar), or as a weight initialisation before training (fine-tuning) on the target domain. Transfer learning is also commonly employed on the detection and segmentation tasks, described below.

Object Detection

Contrary to image classification that predicts a label for each input image as a whole, object detection supports the existence of several objects of interest in a single frame and aims to simultaneously recognise and localise them. As a result, the output of object detection models comprises a list of bounding boxes $B = \{\mathbf{b}_n, \mathbf{p}_n\}_{n=1}^{N_{obj}}$, each denoting the coordinates of a detected object n in the input frame as $\mathbf{b}_n = [x, y, height, width]$ (multi-variate regression), while being associated with a classification-like probability distribution $\mathbf{p}_n$ (classification).

As a result, a multi-task loss function is commonly employed to train detection models. Initially, a challenge arises by having numerous predictions, as well as numerous ground-truth labels. This is addressed by employing a *matching* method, such as the Hungarian algorithm [188] to provide prediction-label pairs. Prediction summarisation methodologies, such as Non-Maximum Suppression (NMS) [189], are also used to merge duplicate detections of the same object into a single one. Intersection-over-Union (IoU) is commonly used as a distance metric between two bounding boxes $\mathbf{b}_A$, $\mathbf{b}_B$, to facilitate this matching:

$$\text{IoU}(\mathbf{b}_A, \mathbf{b}_B) = \frac{\mathbf{b}_A \cap \mathbf{b}_B}{\mathbf{b}_A \cup \mathbf{b}_B} \qquad \in [0, 1] \tag{2.25}$$

Subsequently the loss consists of a classification term across $N_{class} + 1$ categories, with the extra term accounting for a background (no-class) option (indexed as class 0 below); and a box regression loss applied only on predictions corresponding to valid objects in the ground-truth data:

$$\mathcal{L}(B, \hat{B}) = \sum_{n=1}^{N_{obj}} \mathcal{L}_{cls}(\mathbf{p}_n, \hat{\mathbf{p}}_n) + \mathbb{I}(\arg\max \hat{\mathbf{p}}_n \neq 0) \cdot \mathcal{L}_{reg}(\mathbf{b}_n, \hat{\mathbf{b}}_n) \tag{2.26}$$

By matching bounding boxes into pairs and calculating and thresholding their IoU score, detections can be categorised to valid True Positive (TP) or non-valid False Positive (FP), as well as missing False Negative (FN). This redefines accuracy in the context of object detection and establishes other useful metrics:

$$\text{Accuracy} = \frac{TP}{TP + FP + FN}, \quad \text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN} \tag{2.27}$$

A less intuitive metric, most commonly employed in competition settings such as the Microsoft Common Objects in Context (MS-COCO) benchmark [190], is mean average Precision (mAP). By tuning the confidence threshold, a trade-off between False Negatives and False Positives is explored. This trade-off can by captured by plotting the 2-axis Precision-Recall (PR) curve.

Average Precision (AP) is obtained by calculating the Area-under-Curve in the PR plot; while doing so individually for each class and calculating the mean defines the mAP metric.

Numerous different approaches have been proposed for the object detection task. Most of them employ transfer learning by re-using classification-based backbones as feature extractors, which are enhanced by tailored detection modules appended in place of the classifier.

Faster R-CNN [4] constitutes a two-stage detection model, where a region-proposal network initially identifies image regions that could potentially contain objects of interest, which are then extracted, reshaped and fed independently to a classifier and box regression units to yield the final predictions.

Single-Shot Detector (**SSD**) [191] and **YOLO** model family [5][192][193] form alternative single-stage detection models which directly predict boxes and probability distributions from input images. In these models, a fixed cardinality of predictions (being uniformly distributed across classes in SSD or image cells in YOLO) is provided through a single forward pass, which are subsequently filtered through NMS and IoU thresholding.

Image Segmentation / Dense Prediction Tasks

Semantic Segmentation is the most fine-grained visual understanding task, aiming to assign a class label from a predefined set to *every pixel* of the input image. In that sense, segmentation is a dense prediction task with the output *mask* $\mathbf{M} \in \mathbb{R}^{H \times W \times N_{class}}$ featuring the same spatial dimensions as the input, where a probability distribution $\mathbf{p}_i$ across classes in inferred for each pixel.

Training-wise, segmentation is viewed as a large set of independent classification tasks, calculating a cross-entropy loss for each pixel and reducing them to a single value through summation/averaging, or by employing a weighting/balancing algorithms, such as the focal loss [194]:

$$\text{Focal}(\mathbf{p}_i, \hat{\mathbf{y}}_i) = -\sum_{n=1}^{N_{class}} \hat{\mathbf{y}}_i[n] \cdot (1 - \mathbf{p}_i[n])^{\gamma} \cdot \log(\mathbf{p}_i[n]) \tag{2.28}$$

mitigating the dominance of background pixels in some images (and generally class imbalance), parametrised by a tunable parameter γ.

Evaluation-wise, the most broadly used metrics are: *pixel accuracy* that measures the ratio of correctly classified pixels over the total number of pixels in the image; and *mean Intersection over Union* (mIoU) that captures the average IoU across all different classes.

Typically segmentation CNNs are fully-convolutional [195], featuring an encoder (feature extraction) and decoder (reconstruction) part. The decoder typically employs de-convolution [196] or up-sampling operations to restore spatial dimensions (e.g. **SegNet** [6]), while often incorporating skip-connections from the encoder and feature concatenation to guide the spatial reconstruction process (e.g. **U-Net** [197]). Alternatively, by replacing traditional pooling in the encoder with dilated (atrous) convolutions [7], the receptive field of each layer can be equally increased without any loss of spatial information (e.g. Dilated Residual Networks (DRN) [198]), eliminating the need for the decoder at a significantly increased computational cost.

Similarly to semantic segmentation, Depth Estimation also comprises an every-pixel task, which is however formulated as a regression problem. The aim of depth estimation is to predict a depth value for each pixel of a RGB input image, re-assembling a RGB-D representation without the need to use specialised sensors. Commonly, depth estimation employs similar model architectures and summarisation techniques (across pixels) to segmentation to calculate corresponding regression loss and accuracy terms [199].

Instance Segmentation is a more complex Computer Vision task that involves both object detection and semantic segmentation. The goal is to individually label every-pixel in the input image, while distinguishing between different *instances* of the same object class.

Intersection-over-Union (IoU) or Dice loss [200][201], defined as:

$$\text{Dice}(\mathbf{M}_i, \hat{\mathbf{M}}_i) = 1 - \frac{2 \cdot |\mathbf{M}_i \cap \hat{\mathbf{M}}_i|}{|\mathbf{M}_i| + |\hat{\mathbf{M}}_i|} \tag{2.29}$$

between a predicted $\mathbf{M}_i$ and ground-truth $\hat{\mathbf{M}}_i$ instance masks are commonly used for training instance segmentation models along with cross-entropy.

The most frequently met instance segmentation models are built on top of Faster-RCNN detection pipeline, adding a fully-convolutional branch as in **Mask-RCNN** [202]; or employing multi-scale Feature Pyramid Networks (FPN) [203] as in **YOLACT** [204]. At the same time, evaluation comprises a fusion of detection (mAP) and semantic segmentation (mIoU) metrics.

2.3 Embedded Systems

The impressive predictive accuracy of deep learning models has led to their adoption in a wide range of CV applications. Many of these impose low-latency, privacy and security constraints, restricting

the model inference near the sensor (i.e. on the same platform). Consequently, deployment resorts to the embedded/mobile landscape where the available computational resources, memory capacity and power envelope are significantly limited, compared to the datacentre.

With a constantly increasing number of Internet-of-Things (IoT) devices emerging in every-day life, the market of embedded computer systems is one of the fastest growing portions of the tech world [205]. With the computational power of embedded Central Processing Units (CPUs) being limited for some demanding visual intelligence applications [206], a dominating trend in the embedded landscape of deep learning systems is the integration of a generic processor cores with application-specific hardware Integrated Circuits (ICs) on the same chip, referred as Systems-on-Chip (SoC). Nowadays, different communities follow fundamentally different schools of thought on the design of such application-specific accelerators, which are briefly introduced in this section.

2.3.1 Application-Specific Integrated Circuits (ASICs)

On the one end of the spectrum, Application-Specific Integrated Circuits (ASICs) first appearing in 1980s, have emerged in the domain of machine learning [66][207][208], offering tremendous customisation capabilities with minimum area and power consumption. ASICs however, lack flexibility upon deployment as their hardware and memory structure is tailored to specific workloads (i.e. model families), introducing excessive non-recurring engineering costs. An additional deployment challenge arises from the long development cycles and turnaround times of ASICs [209], that can only be alleviated if production takes place in sizeable scale, thus limiting the ability to quickly adapt and iterate designs. This is a crucial downside in the ever changing domain of deep learning applications [210], with new model architectures emerging continuously.

2.3.2 General Purpose Graphics Processing Units (GPGPUs)

On the other end, Graphic Processing Units (GPUs), the design of which has been shaped by the video game industry, offer massive Single-Instruction-Multiple-Data (SIMD) parallelism capability. Over the past decade, GPUs have been progressively re-purposed as General Purpose accelerators for linear algebra and *tensor*[9] processing operations, offering throughput-optimised kernel execution [212]. As illustrated in Fig. 2.8a and b, compared to multi-core CPUs, GPUs are more spatial in the sense of comprising a much larger number of (significantly simpler) processing

[9]Tensors is another term for multi-dimensional arrays.

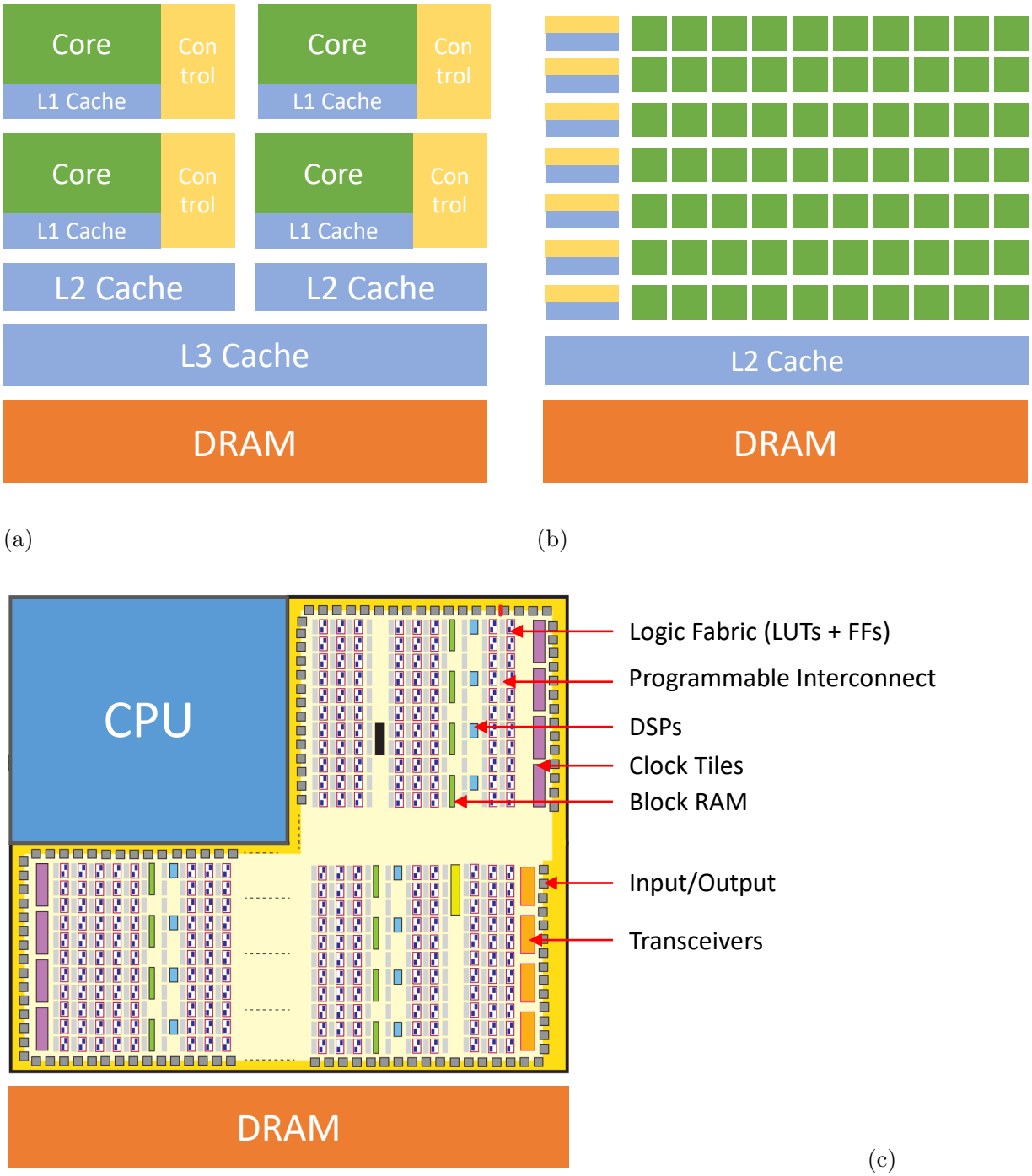


Figure 2.8: Conceptual illustration of hardware structure within: (a) CPUs, (b) GPUs; (c) FPGAs. Subfig. (c) constitutes a modified version of a figure sourced from [211].

Table 2.1: Technical Specifications of employed Nvidia GPU platforms.

Device	Architecture	CUDA Cores	DRAM	Mem. BW	Clock	TDP
Jetson TX2	Pascal (16nm)	256	8GB	59.7 GB/s	1.30 GHz	15 W
AGX Xavier	Volta (12nm)	512	16GB	136.5 GB/s	1.37 GHz	30 W
GTX1080	Pascal (16nm)	2560	8GB	320.3 GB/s	1.73 GHz	180 W
RTX2080-Ti	Turing (12nm)	4352	11GB	616.0 GB/s	1.54 GHz	250 W

cores [213], paving the way to many-core architectures [214]. Recently, GPU accelerators have appeared in the embedded space [215], fitting in a competitive power envelope that enables their applicability on mobile robot applications.

GPUs generally favour flexibility by adopting a fixed hardware structure that is programmable at the software level. As a consequence, GPUs rely on the regularity and uniformity of the target workload to yield high performance, and thus are less suitable than ASICs to realise latency speed-ups from some optimised DNN architectures. However, in the context of deep learning, GPUs can be programmed by several Application Programming Interfaces (APIs) such as CUDA[10], cuDNN[11] and Software Development Kits (SDKs) such as TensorRT[12] offering a fine-granularity trade-off between low-level control and development time; while the wide support of most deep learning frameworks [216][217][218] enables even their out-of-the-box deployment on the target use-case. Some technical specifications for the GPU boards used in this thesis are listed in Table 2.1.

2.3.3 Field-Programmable Gate Arrays (FPGAs)

In the middle of this flexibility-customisation trade-off lie SoC Field-programmable gate arrays (FPGAs), typically consisting of one or more processors alongside reconfigurable fabric [219]. The processor is responsible for executing non-critical code and coordinating the operation of the overall system. The reconfigurable fabric can be customised at the hardware level, allowing the on-chip computational and memory resource allocation to be optimised to match the particular workload and consequently the performance needs of the target application and its underlying implementation.

FPGAs maintain a notable proportion of the customisation capability and power efficiency of ASICs, while remaining programmable and flexible. Additionally, the reconfigurability of

[10] https://developer.nvidia.com/cuda-toolkit
[11] https://developer.nvidia.com/cudnn
[12] https://developer.nvidia.com/tensorrt

FPGAs allows for hardware architecture adaptation to take place even at runtime, at significantly lower cost and iteration speed than ASICs, offering an optimised mapping of diverse workloads [220]. Because of this versatility, FPGAs have found their way to the deep learning systems domain, both in the cloud [221] and the embedded space [222], offering significantly reduced operating cost due to their pronounced power-efficiency.

Of course, this adaptability does not come for free; as FPGAs tend to suffer from considerably reduced clock speeds [210] and memory bandwidth due to their complex configurable interconnection network. This challenge is progressively addressed by the addition of hardened specialised logic blocks [223] next to the configurable fabric, which are able to meet the clock at a considerably increased frequency. Nonetheless, designing and verifying custom hardware accelerators through Hardware Description Languages (HDL) remains a tedious and time-consuming process, that requires in-depth hardware expertise and prolonged development cycles compared to GPUs. This challenge is partly remedied by the emergence of High-Level Synthesis (HLS) tools [224][225], offering the ability to automatically translate C/C++ level code to optimised hardware designs, introducing an abstraction layer at the cost of reduced control that results in partial performance degradation. To re-gain this performance, hardware experience needs to be re-introduced in the process, calling for specially authored C/C++ functionality descriptions that aim to guide the resulting design (without explicitly specifying it) [226]. In all cases, integration with commonplace Deep Learning frameworks used by AI practitioners is out of the question, apart from the development of domain-specific toolflows for mapping DNNs to custom hardware [117][65] from the research community.

Diving deeper in the internal structure of SoC FPGAs (Fig. 2.8c), an embedded CPU is tightly interconnected with a Programmable Logic fabric, co-habitating in the same chip. The Programmable Logic features two main types of computational components: Look-Up Tables and Digital Signal Processors [211].

Look-Up Tables (LUTs) are primary logic elements forming a flexible resource that can emulate a logic function with a limited number of inputs or a small memory structure (ROM, RAM or shift-register). LUTs are typically organised into regular groupings and connected by a programmable interconnect network, that enables the combination of several LUTs to implement more complex logic functions. LUTs are also typically accompanied by super-fast memory elements (Flip-Flops) implementing shift registers.

Table 2.2: Available resources of employed Xilinx FPGA platforms.

Board	Chip	LUTs	DSPs	On-chip Mem.	DRAM	Mem. BW
Zynq ZC706	z7045 (28nm)	218,600	900	2.40 MB	1 GB	4.4 GB/s
UltraScale+ ZCU102	zu7ev (16nm)	230,400	1,728	4.75 MB	4 GB	13.4 GB/s

Digital Signal Processors (DSPs) are more specialised computational units, able to implement high-speed arithmetic on inputs with larger wordlength. DSPs typically comprise a pre-added/substractor followed by a multiplier and a post-adder with logic and accumulation capability. As with LUTs, DSPs can be combined through programmable interconnect to implement more complex arithmetic functions.

Also, distributed across the chip are Block RAM modules, forming dedicated and optimised memory elements that can implement different structures (RAM, ROM and FIFO buffers). Additionally, FPGAs feature different types of Input/Output Blocks (IOBs), as well as clocking functions facilitating the operational and interface requirements of the SoC.

The available resources of the SoC FPGA boards used in this thesis are listed in Table 2.2.

2.4 Efficient On-device Inference

Deep neural networks are inherently redundant by being over-parametrised at design time. This property has demonstrated accuracy gains at training time [42], leading to a stream of accuracy-centric approaches that yields constantly increasing predictive capabilities at a dramatic increase in computational cost. Manual and automated approaches have emerged in the embedded deep learning era, evaluating the impact of different design choices in accuracy and inference speed in a quest to design more efficient models [227].

Additionally, real-world applications in the embedded space introduce further applicability requirements (e.g. in model size or energy consumption), which are discussed in this section. Towards meeting such constraints, both at design time and upon deployment, a large portion of this redundancy can be eliminated without incurring a considerable accuracy penalty. This property has been exploited by numerous model and inference approximation methodologies [228][229], also summarised below later in this section.

2.4.1 Key Performance Indicators, Beyond Accuracy

They key point to be made in this section is that there are many more considerations apart from accuracy in the deployment of DNNs to real-world applications [230][106]. Some of these Key Performance Indicators (KPIs) and design objectives are summarised below:

- *Latency* (typically reported in seconds (s)) is used to measure the time spent in the whole inference process of a single sample (from the moment the data arrive to the system until the output is generated). Indicatively real-time applications often impose a stringent constraint in latency (in the order of 30ms or less), which is slightly relaxed in near real-time settings (ranging from 100ms to 1s, according to the use-case).

- *Throughput* (typically reported in operations per second (op/s) or frames per second (fps)) is used to measure the amount of input data that can be consumed and processed (run inference) by the system within a given time period. Although throughput and latency can be viewed as directly inversely proportional, several optimisations such as *batching* offer a wide trade-off between them (even dating back to queuing theory [231]). Throughput-optimisation is typically central to non real-time applications. However, continuous vision tasks in latency-critical domains such as robotics usually co-optimise and balance latency and throughput, to be able to consume all the generated data (e.g. a 30 fps camera stream), and mitigate False Negatives caused by "dropping" frames.

- *Energy* (typically reported in Joules (J)) is a measure of the total amount of work being performed. In the context of deep learning inference, attention is paid to the relevant metric of *Energy Efficiency* (measured in frames/J) used to indicate the amount of input data that can be processed for a given unit of energy. This metric becomes increasingly important in battery-operated platforms, where it is optimised in a lower-is-better fashion to maximise the operational time of the system within a battery cycle.

- *Power* (typically measured in Watts (W)) is a measure of the rate at which energy is being consumed. Apart for being a key limiting factor for the computational resources available on mobile agents (frequently capping Thermal Design Power (TDP)[13] at most to 30W), *Power Efficiency* (ops/s/W) is frequently used in embedded systems as a normalisation factor in the comparison across different devices.

[13]TDP refers to the power that the cooling system of the processing device is designed to be able to dissipate.

- *Memory Footprint* (typically measured in *bytes*) is a measure of the model size and memory-size requirements during the inference process. It is most commonly considered in the mobile end of the spectrum, where devices are equipped with limited storage and memory capacity [159]. Minimising the model size and required memory accesses also plays a key role in energy efficiency [106].

- *Privacy*: The majority of data generated on mobile/embedded devices are often sensitive [232] (e.g. collected within a user's home [233]) and need to stay and be processed locally. Thus inference systems usually need to protect data security by shifting to an edge inference paradigm [90].

- *Availability*: Except from privacy, continuous service requirements, that can be hindered by network availability issues and communication overheads, form another constraint that restricts the cloud-processing of data in many remote applications.

2.4.2 Efficient Model Design

This section focuses on approaches that can be used at DNN *design time* to improve the efficiency of the resulting model.

Neural Network architectures are parametrised by a large set of design choices such as the number of layers, the type of employed operations, their size and shape, and cross-layer organisation determining the dataflow. Efficient model design is all about tuning and trading these different aspects aiming to make deep learning models more computationally efficient, rather than simply optimise their accuracy [234].

Early efforts in this direction involved manual design of efficient layer structures, undertaken by human experts. The most frequently used approaches in this line of works focus on: (i) reducing the spatial dimensions of the employed filters [235], (ii) reducing the number of channels of the feature volume though summarisation operations [51], (iii) reducing the number of filters employed in each layer [236], or (iv) introducing more efficient forms of the convolution operation [237] and layer structures [238].

Although effective, manual network design remains a tedious and slow process, due to its trial-and-error nature with costly training of models being interjected between design iterations. Automated approaches searching for efficient network architectures have emerged under the term

Neural Architecture Search (NAS). In this setting, an optimisation algorithm (including random search [239], evolutionary [240], reinforcement learning [241], Bayesian optimisation [242] or gradient decent [243]) undertakes the design of each model iteration (which can also happen in batches) by sampling architectures from a large-scale predefined search space [244][245] by considering the results of the previous optimisation round.

The optimisation objective and benchmarking practises in the above works play a significant role on the efficiency of the resulting architectures. Experience has shown that reducing the number of parameters or theoretical computational complexity does not always translate to latency speed-ups or energy savings. This is due to numerous additional factors, such as resource utilisation, cost of memory transactions and scheduling overheads coming into effect at deployment time. As a result, performance modelling [246], hardware in-the-loop [61], and model-hardware co-design [69] methodologies have emerged, leading to improved inference efficiency at a system level.

Special training methodologies have also been developed aiming to distil (transfer) knowledge from large over-parametrised model architectures to more efficient ones, in a teacher-student paradigm termed knowledge distillation [58]. These methodologies aim to partially account for the limited representation-learning capability of smaller models, by training them to directly imitate the predictions (or intermediate representations and visual embeddings [247]) provided by larger trained models. The intuition behind this approach is two-fold. First, the probability distribution between classes is a more information-rich signal compared to the one-hot labels, offering the student model more insightful correlations at training time. Additionally, distillation implicitly provides a cleaned-up version of training data, by eliminating the inflation of loss from conflicting or ambiguous cases that even the teacher network was not able to recognise correctly.

Notably, all the above approaches require unimpeded access to training data and corresponding computational resources; assumptions that often do not come true due to privacy or memory limitations, as well as constraints by the available infrastructure.

2.4.3 Model Approximations

Instead, this section focuses on methodologies that can be applied on top of a pre-trained DNN, to eliminate redundancy *right before deployment*. Although many of these techniques can be combined with parameter fine-tuning to push the limits of approximation by restoring accuracy, the focus of this section is generally on post-training approximation methodologies.

Quantisation

Quantisation is an approximation technique that reduces the representation *precision* (also called *wordlength*) of the DNN's weights and/or activations. This approximation is beneficial in many ways, such as: (i) reducing the required *memory bandwidth* (and thus computation-to-communication ratio), consequently boosting the attainable performance (throughput) on memory-bound workloads, (ii) reducing the complexity and required *area* of computation units[14], while enables more parallelism and thus achieving higher performance, (iii) reducing the *memory storage* requirements of the model parameters, which is important on resource-constrained environments, and iv) reducing *energy consumption* which is dominated by data movement, that plays a key role in battery-operated platforms. However, not all models are equally resilient to reduced-precision calculations, thus quantisation can also notably impact accuracy. Therefore, the goal of post-training quantisation methodologies is to strike a balance between improving performance/efficiency with minimum or controlled compromise in accuracy.

Reducing the wordlength of a numeric representation from the default 32-bit format (fp32), essentially limits the number of unique values that can be attributed to it. Through this lens, Quantisation $Q(\cdot)$ determines the mapping from high (x) to low ($\tilde{x}$) -precision representations:

$$\tilde{x} = Q(x). \tag{2.30}$$

When deciding the quantisation level and mapping function, the most common optimisation objective is minimising the average quantisation error, defined as the average difference between the original and quantisation values:

$$E[(\mathbf{x} - \tilde{\mathbf{x}})^2] = \int_{x_o=x_{min}}^{x_{max}} (\tilde{\mathbf{x}} - x_o)^2 \cdot p_{\mathbf{x}}(x_o)\, dx_o \tag{2.31}$$

where $p_{\mathbf{x}}(\cdot)$ is the probability density function for $\mathbf{x}$. Therefore, the average quantisation error also depends on the data distribution. The range of the input data distribution (x_{max}/x_{min}) and the required granularity of representations determine the required wordlength and vice-versa. Different quantisation schemes have been proposed [249], such as *uniform quantisation* which employs a uniform spacing between the quantisation values across the spectrum and *non-uniform* where this spacing can vary, offering enhanced flexibility at the expense of requiring significantly more complex hardware arithmetic units.

[14] Convolutions are multiplication-heavy operations. Integrated Circuit Design theory suggest that area and energy of a multiplier scales quadratically ($O(n^2)$) with wordlength n [248].

When custom hardware accelerators (such as NPUs) are considered for deployment, quantisation is often combined with *fixed-point* representations [250]. In contrast to the industry-standard Floating Point arithmetic (IEEE 754), fixed-point representations use the much simpler integer (int) format, along with a group-wise (e.g. layer-wise) scaling factor to determine a static location of a decimal point for a set of values[15]. This optimisation offers significant efficiency gains by massively reducing the complexity of the required hardware units.

More aggressive quantisation approaches have explored reducing precision down to a single bit, with the resulting models named binary networks. That way multiplication can be turned to a bit-wise XNOR operation [112], leading to profound reduction of memory requirements and efficient hardware mapping. However, even after fine-tuning, binary networks have still not been able to achieve comparable accuracy to their higher-precision counterparts [253], limiting their applicability on critical decision making applications; unless highly-specialised model topologies are introduced [254]. Other approaches study variable-precision DNN quantisation, with different layers adopting different precision levels [255], that can be tuned in a hardware-aware manner [256] or even at runtime [257]. This however, usually over-complicates the hardware design, leading to diminishing or non-practically realisable returns with the current design practises [106].

Notably, although some benefits from quantisation can directly be realised as performance gains across deployment platforms (i.e. by reducing the required memory traffic), fully exploiting the efficiency potential of quantisation typically requires support from custom (reduced-precision) hardware blocks. Such units are increasingly becoming available in GPUs at a coarse precision granularity, however custom hardware solutions in the form of ASIC- or FPGA-based NPUs currently offer the enhanced flexibility required to fully exploit quantisation's efficiency potential.

Pruning

Pruning exploits an orthogonal direction towards reducing the redundancy of deep learning models, by imposing *sparsity* on the model parameters [258]. Data sparsity essentially refers to the existence of several zero values in the data. Sparsity can be translated into a reduction in the memory footprint and computational complexity of the inference process, as addition and multiplication operations by zero can be skipped (or replaced by fixed-value signals at the hardware level).

[15]This re-assembles the existence of a shared exponent across the whole group of numbers, also referred to in the literature as Block Floating Point [251] or Dynamic Fixed Point [252] representation.

Model pruning approaches aim to identify the weights that have the smallest impact on the model's predictions and remove them by essentially replacing them by zeros. For this process, an importance criterion needs to be determined to score the contribution of each weight. This can be magnitude based [259], or consider a projection of the impact of each weight to the model output [260]. Optionally, in *structured pruning* [261] the weights are grouped following a pre-determined structure [262]. This steps trades flexibility for regularity in the remaining weights, that can essentially guarantee a better mapping of the resulting computation to the target platform [263], leading to the realisation of proportional speed-ups. The groups (or weights in the case on *unstructured pruning*) are then ranked (globally or locally [264]) based on their score. This ranking, along with a predefined target sparsity level, determines which weights would be eliminated [56].

As such, pruning typically needs to be performed in a hardware-aware manner to offer proportional performance gains on general-purpose accelerators under the assumption of a highly-structured approach; which can partially be relaxed when the flexibility of custom-hardware accelerators is also exploited.

Compression

Several compression methodologies have also been applied to reduce the storage, memory traffic and computational requirements of pre-trained models, by summarising their parameters in a lower-dimensional representation [265]. Commonly, large tensors representing the weights of a layer, can be *decomposed* to a series of smaller tensors [266], that can additionally be compressed by *low-rank factorisation* methodologies [267], effectively reducing the number of parameters. Similar to quantisation, the memory traffic gains arising from compression are often directly realisable on generic hardware, whereas exploiting additional computation gains relies heavily on support from highly-customised hardware.

2.4.4 Dynamic Inference

This section paves the last mile towards inference efficiency, focusing on approximation methodologies that take place *upon deployment* (at runtime), to eliminate further redundancies and facilitate efficiency in a dynamic manner.

The design methodologies discussed in the previous sections statically optimise performance/efficiency, while aiming to preserve accuracy at the desired level for the target application. Accuracy

optimisation typically focuses on a significantly dominant percentile of the data distribution, as perturbing the model predictions for just a few samples is directly projected on the examined accuracy metric. However, not all input samples pose the same challenges or require the same amount of processing to yield a correct prediction. Additionally, the resource budget on the target platform may be dynamically affected due to resource sharing between multiple models/tasks [268], as well as thermal management techniques commonly applied in the embedded landscape [98].

Therefore, employing static approximations uniformly *across inputs* is essentially optimised to the sensitivity of the most difficult samples of the validation data, leaving plenty of unexploited redundancy still evident on easier (less ambiguous) inputs. Additionally, employing static approximations *over time* fails to capture the dynamicity in resource availability and allocation on the processing platform, being challenged by the concurrent and stochastic execution pattern of different tasks.

Dynamic inference aims to remedy the above challenges by dynamically adopting the computation path at runtime, in view of sample difficulty [269] or resource-availability [270] information. This commonly materialises in the form of *model selection* methodologies, where upon deployment an algorithm adaptively chooses between a set of models with diverse accuracy/latency characteristics that have been trained on the same task, aiming to guide each (batch of) input(s) through a different processing path, while meeting application-specific constraints.

These models can be arranged as parallel alternatives along with a preceding *difficulty-estimator* or *runtime-manager* matching each input image to a model [271]; or in the form of a cascade, featuring progressively more powerful but computationally expensive models [272], with the selection criterion relying on the prediction uncertainty of each stage [273], trainable components [274] or hybrid approaches [275].

Through this paradigm, the inference process is re-formulated to follow a conditional input-dependent paradigm, where the aim is to minimise the total amount of computation spent to process a volume of input data, while meeting application-specific constraints. This paradigm has been extended within the deep learning models, with dynamic DNN architectures [276] adapting their depth [175], width [277], input resolution [105] or computation path (i.e. by skipping layers [278][279]) at runtime, according to the estimated difficulty of the current input [128].

2.5 Robot Visual Intelligence

Robot Perception is increasingly becoming *vision-based* due to the abundance of geometric and semantic information captured by visual cues [280], as well as the tremendous advancement of the predictive capability of visual understanding approaches, predominantly driven by the emergence of deep learning.

Robot vision, however, demonstrates additional challenges and opportunities from traditional computer vision in several aspects, arising from the fact that *robot perception is only a part of a more complex, embodied and goal-driven system* [70].

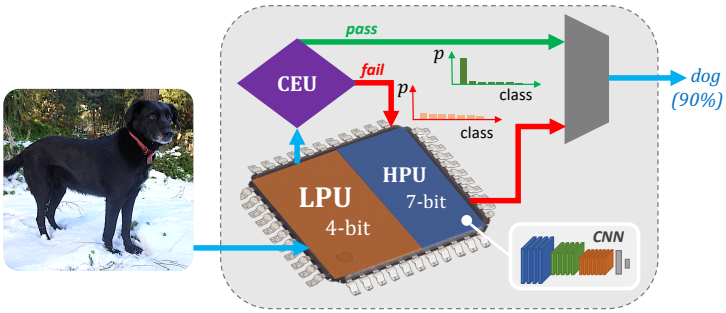
First and foremost, robot vision is both *continuous* and *real-time*. As a result inference is both *throughput* and *latency* sensitive. More specifically, instead of concerning individual frames, robot perception systems are tasked with processing a continuous stream of images. Due to the dynamic behaviour of the robot, as well as others agents/humans operating in the same environment, significant changes can arise at any time. Timely identifying such changes in the relative state of the robot and its environment is crucial for making mission-critical reactive decisions and requires low response time, as well as keeping up with the rate of incoming frames captured by the visual sensors. This jointly imposes high-accuracy, low-latency and high-throughput requirements on the inference process. At the same time, the embodyment of deep learning models on-board the robot platform, constrains the availability of computational power [281]. This situation leads to aggravated difficulty compared to other use-cases, where a trade-off is allowed between the aforementioned resource/accuracy/throughput/latency constraints/requirements to facilitate the deployment of solutions on the target domain.

Additionally, robot vision is *stateful*. Robot perception needs to encapsulate spatial and temporal information regarding the geometric and semantic properties of the environment. This already challenging task is hindered by the partial observability of the environment due to sensor limitations (such as limited field of view, sampling rate, sensitivity to light conditions etc), clutter and occlusion. Additionally, the temporal embodyment of robot vision in the real-world is further challenged by the lack of consistency that characterises real-world environment, due to the presence of dynamic objects and other actors operating in the scene.

Furthermore, robot vision is *multi-modal*. Robots are frequently equipped with multiple sensing modalities. Although this property is largely exploited by traditional approaches [282],

learning-based computer vision research is dominated by monocular modalities, as the limited spatial and temporal alignment between different sensor cues introduces further challenges in their incorporation on learning-based systems. Although recent literature has started to exploit multi-modal information in a quest to improve accuracy, there are several unexplored opportunities for exploiting additional sensory inputs towards efficiency.

Finally, outside the scope of this thesis, robot vision differentiates from computer vision in other aspects, such as being *active*, in the sense that the deployed system has the unique opportunity to control the camera pose, offering the ability to selectively collect more information and consequently improve the observability and understanding capability of the system. This concept is tightly coupled with the integration of *uncertainty* estimation in the inference process which, in the context of deep learning, poses its own reliability challenges while introducing excessive computational overhead through the current methodologies.



3

CascadeCNN:

Pushing the limits of quantisation in CNNs

Contents

3.1 Overview 56
3.2 Motivation 56
3.3 Related Work 59
3.3.1 Approximate Computation in Deep Learning Inference 59
3.3.2 Conditional Computation in Machine Learning 61
3.4 Methodology 64
3.4.1 Overview 64
3.4.2 Quantisation 65
3.4.3 Confidence Estimation 67
3.4.4 Hardware Architecture 69
3.4.5 Design Space Exploration (DSE) 72
3.4.6 Latency-Aware DSE 78
3.5 Evaluation 82
3.5.1 Experimental Setup 82
3.5.2 Architectural Performance and Accuracy 82
3.5.3 Throughput-Optimised Cascade Evaluation 84
3.5.4 Latency-aware Cascade Evaluation 87
3.5.5 Performance Modelling Accuracy 89
3.5.6 Qualitative Analysis on Scene Recognition 90
3.6 Discussion 92

Keywords: *CNNs, Quantisation, Image Classification, Input-dependent Inference, FPGA architecture, Automated toolflow, Uncertainty Estimation, Design Space Exploration*

3.1 Overview

*This chapter focuses on the **computation layer** of the deep learning deployment stack, bringing together the concepts of post-training quantisation and input-dependent inference. The proposed framework, CascadeCNN, is an automated toolflow that pushes the performance limits of CNN deployment by approximating the computations of a given CNN model through quantisation and efficiently mapping the underlying operations on custom hardware. The aim of the toolflow is to increase inference performance (throughput or latency) of pre-trained CNN models through quantisation and efficient FPGA-based deployment, without the need for any model fine-tuning or access to training data. Driven by the fact that not all inputs require the same amount of processing to yield a confident prediction, a two-stage architecture tailored for any given CNN-FPGA pair is generated, consisting of a low- and a high-precision unit, arranged in a cascade. A confidence evaluation unit in the middle, is employed to identify misclassified cases from the excessively low-precision unit at runtime, and forward them to the high-precision unit for re-processing. This allows to reduce the low-precision unit's precision to unprecedented levels, offering significant performance gains when deployed on the proposed custom hardware accelerator.*

This chapter is based on two conference papers co-authored with Dr. Stylianos I. Venieris and Prof. Christos-Savvas Bouganis: (FPL 2018), (DATE 2020).

3.2 Motivation

In recent years, Convolutional Neural Networks have become the most broadly adopted solution across a wide range of Machine Vision tasks, challenged to deal with problems of continuously increasing complexity. A definitive factor for this has been their unprecedented ability to learn robust representations, leading to high predictive accuracy. Nevertheless, the significant advancement of CNNs came with: (i) increased number of layers [36], (ii) increased number of kernels [283] and (iii) more complex architectures [3], which introduce substantial complexity costs, both in terms of computational and memory resources.

For scenarios that require the deployment of such AI algorithms in the embedded landscape, as in the case of autonomous mobile agents, these challenges are further aggravated by the imperative performance requirements imposed by real-time applications (e.g. for mission-critical

decision making), as well as the limited resource and power budget available on-board. Although the exact resource/power constraints are heavily dependent on the target use-case, while this chapter strives to achieve a more general solution, it is realistic to assume that mobile robot applications restrict the computation to embedded SoC platforms with a power envelope of up to 20W [284]. Hence, in order to deploy CNNs in the embedded space it is necessary that their high computation and memory burden is alleviated.

Towards this direction, from a software perspective, numerous model compression techniques have been proposed in the literature, aiming to exploit the inherent redundancy evident in most state-of-the-art CNN models. These involve: (i) model quantisation [55][285], (ii) parameter pruning [56], (iii) compression [286][287] and (iv) low-rank approximation [288]. The aim of these efforts is to uniformly alleviate the computational cost, with no (or minimum) compromise in accuracy. As a result, in these approaches, it is typical that the level of approximation is tightly bounded by the model's accuracy resilience to approximation, as captured on the target dataset.

Driven by the fact that not all inputs require the same amount of computation to yield a confident prediction, this chapter follows the paradigm of input-dependent computation, aiming to dynamically save computation on "easier" samples (i.e. those that appear to be more resilient to approximation). In this way, a greater performance boost can be achieved by dynamically pushing the approximation beyond what would be uniformly acceptable (across all samples) on easier inputs, while preserving the overall accuracy by using more a faithful variant of the original models on more challenging ones.

Nevertheless, this dynamicity on the processing path of each sample increases the complexity of the inference process and dismantles the structure of the computation. This, in turn, introduces deployment inefficiencies that can lead to diminishing returns in terms of actually realisable performance gains (e.g. in latency). From a hardware perspective, in order to mitigate this issue and maximise the attainable performance and efficiency, capitalising on such optimisations, the employed computational platforms have diverted from conventional general-purpose processors towards many-core accelerators (e.g. GPUs) and custom hardware solutions tailored to the needs of the target application (e.g. ASICs and FPGAs).

Field-Programmable Gate Arrays (FPGAs) form a cardinal platform for the implementation of approximate inference approaches [228] due to their enhanced flexibility and customisation capabilities, also empowered by the short development cycles enabled by the emerging High-Level

Synthesis (HLS) tools and automated toolflows developed by the community [117]. For the case of precision quantisation, such platforms enable the use of highly optimised low-precision arithmetic units to achieve performance gains. Existing FPGA-based accelerators have produced hardware designs that span from uniform 16-bit precision [65][289] or mixed 16-bit activations and 8-bit weights [289], with minimal effect on accuracy, down to very high-performance binarised networks [290], but at a significant accuracy loss. In this uniform approximation setting, given a fixed resource budget, the attainable performance for a given error tolerance is limited by the shortest wordlength that meets the error bound.

In order to obtain a better performance-accuracy trade-off, the majority of existing works apply a post-quantisation re-training step, fine-tuning the quantised model's fixed-point weights to restore its accuracy [291] [292]. However, this approach assumes access to the training data of the original model, which is often not pragmatic in real-world applications, frequently imposing strict privacy constraints [232] [293] [294]. Examples of such cases comprise datasets that are proprietary (e.g. collected by private organisations), contain sensitive personal information protected by confidentiality regulations (e.g. medical applications [295]) or even models trained in a distributed setting (e.g. federated learning [296]) with user data that are not accumulated in a single place to form a shareable dataset.

This chapter introduces *CascadeCNN*, a novel automated methodology for generating high-performance cascades of CNN classifiers, pushing the performance of precision quantised CNNs without the need to re-train the model or access its training data. *CascadeCNN* exploits the fact that not all inputs require the same level of precision to obtain a confident prediction and introduces approximate computation (in terms of precision quantisation) as a design dimension for the deployment of CNN cascades. In this respect, the proposed framework generates an excessively low-precision processing unit to obtain rapid (approximate) classification predictions together with a parametrised mechanism for estimating the prediction confidence. The samples that are detected as misclassified are recomputed on a high-precision unit to restore the application-level accuracy and meet the user-specified limits. *CascadeCNN*'s automated toolflow considers the application-specific error tolerance and the target CNN-FPGA pair to provide a highly-customised solution by selecting the quantisation scheme and configuring the cascaded low- and high-precision processing units and confidence evaluation mechanism accordingly.

The main contributions of this chapter are the following:

- The introduction of the first multi-precision CNN cascade in the literature that exploits the custom arithmetic potential of FPGAs to boost CNN inference efficiency. A tunable confidence evaluation unit is injected between the cascade processing units to estimate the confidence of classification predictions at run time. By considering the user-specified error tolerance for the target application, the confidence evaluation unit decides accordingly whether or not an input sample is to be recomputed by the second (higher-precision) stage of the cascade.

- A novel FPGA-based architecture that scales its performance as the wordlength of activations and weights decreases, by fully exploiting the available FPGA resources. The proposed architecture employs both DSP blocks and LUTs to map its compute units and demonstrates performance gains with each bit reduction. The architecture is able to execute both the convolutional and fully-connected layers of CNNs and is parametrised so that it can be configured at compile time.

- A multi-objective design space exploration methodology that takes into account the CNN-FPGA pair and a user-defined error tolerance for the target application and optimises the CNN cascade structure with respect to the quantisation scheme and the hardware configuration for each stage. For throughput-oriented designs *CascadeCNN*'s DSE employs full reconfiguration of the FPGA as a design option, to enable switching between designs highly optimised for each of the two computation stages, along with appropriate batch-processing to maintain high performance by alleviating the cost of reconfiguration. Alternatively, for latency-sensitive usecases, where batching and device reconfiguration costs can be prohibitive, a novel two-stage hardware architecture is introduced and explored by the proposed DSE methodology, replacing device reconfiguration with a resource-sharing approach, such that both processing units are statically mapped on the same device at once.

3.3 Related Work

3.3.1 Approximate Computation in Deep Learning Inference

Aiming to alleviate the computational and memory burden of state-of-the-art deep learning models, several model compression techniques have been proposed in the literature. These methodologies typically adopt "computational shortcuts", introducing different approximations during inference,

in place of a faithful implementation of the original model. With the embedded deployment of deep learning models becoming commonplace in myriads of real-world applications, efficient inference becomes a necessity leading to the widespread adoption of such approximation methodologies. Typical examples of these techniques employ model quantisation, parameter pruning, compression and low-rank factorisation. These approximations are typically applied uniformly across all input samples, exploiting the performance-accuracy trade-off.

Most relevant to this chapter are quantisation techniques, that have the potential to offer significant performance gains when reduced precision calculations are efficiently supported by the underlying hardware. In this direction, several quantisation-aware custom hardware accelerators have been introduced, as detailed below.

Precision Quantisation of CNN Models and Custom Hardware Accelerators

Particularly, low-precision quantisation is an actively researched and increasingly popular approach, having demonstrated the ability to provide significant performance gains while reducing the memory footprint and bandwidth requirements (and consequently energy) [297], taking advantage of the inherent redundancy of state-of-the art DL models.

FPGA-based CNN accelerators have used 16-bit fixed-point precision for both activations and weights with minimal accuracy penalty [289][298] with Suda et al. [299] employing 8-bit weights. Targeting applications where a considerable accuracy degradation can be tolerated, optimised FPGA mappings of binarised [290] and ternary networks [300] have also been proposed.

In between these two approaches, a large stream of works apply fixed-point quantisation to full-precision trained models and employ a retraining step as a mechanism of fine-tuning the network's fixed-point weights and compensating for accuracy losses. In this direction, Gysel et al. [291] proposed Ristretto, a framework that quantises both activations and weights under a block floating-point scheme with uniform wordlength and different scaling across layers. Angel-Eye [292] is a CNN-to-FPGA framework that comprises a quantisation method together with a CNN accelerator. Similarly to Ristretto, Angel-Eye employs uniform wordlength across layers and formulates the per-layer scaling search as an optimisation problem. Zhou et al. [55] presented an incremental scheme that focuses solely on weights and pushes state-of-the-art networks below 6 bits with no loss of accuracy. However, for this re-training to take place, access to the training set of the target application (that the original model was trained on) is required. With

privacy becoming a parameter of paramount importance often protected by regulation, and large proprietary datasets being valuable assets for private organisations, this assumption becomes constantly less realistic in real-world applications.

Following a different approach to the majority of existing quantisation methods, *CascadeCNN* is relieved of the need to have access to the whole application dataset, as it does not employ model retraining in its quantisation scheme. Instead, a two-stage cascade structure is generated using only a small subset of labelled samples (e.g. validation set), where the higher-precision processing stage is responsible for restoring accuracy to user-defined acceptable levels, following an conditional-computation inference setting described below.

3.3.2 Conditional Computation in Machine Learning

Uniform approximation in deep learning models can quickly reach its efficiency limits, due to the existence of challenging samples, being less resilient to approximation, causing accuracy degradation when computation is rigorously loosened. Driven by the fact that not all inputs require the same amount of computation, recent literature has studied *conditional computation* approaches, targeting to avoid computationally expensive workloads on inference, except where needed.

Karpathy et al. utilised an additional CNN model focusing on the centre of each frame (usually demonstrating higher concentration of information in classification scenarios), alongside the main model which acts on the entire image, to unevenly increase the model's capacity across the input for the task of video classification [301].

Dynamic Capacity Networks (DCN) [302] divert from the uniform workload distribution paradigm by using a attention-based mechanism to resort to variable-capacity sub-networks for different spatial locations of the input image, allocating higher workload to the regions identified as task-relevant at runtime. Applied on the task of Recurrent Neural Network (RNN)-based speech recognition, DeltaRNN [303] forms a temporal variant of this methodology, updating the output of each neuron only when a significant difference is evident compared to its previous activation.

Focusing on the task of image super-resolution, MobiSR [113] evaluates the up-scaling difficulty of low-resolution input images, in order to schedule their inference on heterogeneous compute platforms, incorporating models with variable workload-accuracy characteristics.

Finally, a different stream of works, introduce early-stop classifiers, such as Shallow-Deep Networks [304] and BranchyNet [175]. In this compute scheme, multiple "exit lanes" (branches)

are incorporated to the base model, yielding early classification decisions based on intermediate activations of the network. By evaluating the prediction entropy, more layers become available to harder samples, to maximise the likelihood of a correct prediction.

Cascade Machine Learning Systems

Cascades of classifiers form a widely studied design approach in machine learning [305][306]. Such algorithms aim to minimise the computation time per classification by exploiting the property that different inputs require different amount of computation to obtain a confident prediction. A cascade structure is typically formed by connecting classifiers of increasing complexity as a multi-stage architecture. At each stage, based on the confidence of its prediction, a classifier can either decide upon the classification of the current input sample and terminate the execution or pass it to the next stage. The deep learning community has mainly focused on the design of CNN cascades from an algorithmic perspective, targeting diverse tasks including the detection of faces [307], pedestrians [308] and objects [309]. In [310], for example, a general CNN-based classifier is responsible to route the more challenging input samples through cascaded "expert" classifiers. These classifiers are trained on subgroups of classes, formed based on the model's confusion matrix, targeting to boost prediction accuracy at an extra computation cost as required.

In a more systems-oriented approach, Kang et al. [311] proposed NoScope, an automated framework that generates CNN cascades optimised for high-throughput query-based video analysis on high-end GPUs, by searching the design space of both CNN models and cascade structures with the objective to perform binary classification tasks.

This chapter approaches CNN cascade design from a different perspective to the above works. Instead of designing one weaker and one more sophisticated model at the training phase, *CascadeCNN* takes as input a model already trained at a given precision and generates one weaker (but faster) low-precision model and one slower (but more accurate) high-precision model. To decide whether to pass each sample to the second-stage classifier, a novel tunable confidence evaluation unit is introduced, which estimates the classification confidence of the low-precision model. By taking into account the user-specified error tolerance, the two stages of the cascade, the corresponding hardware architectures and the confidence evaluation unit are co-optimised to maximise performance while lying within the acceptable user-defined error bounds.

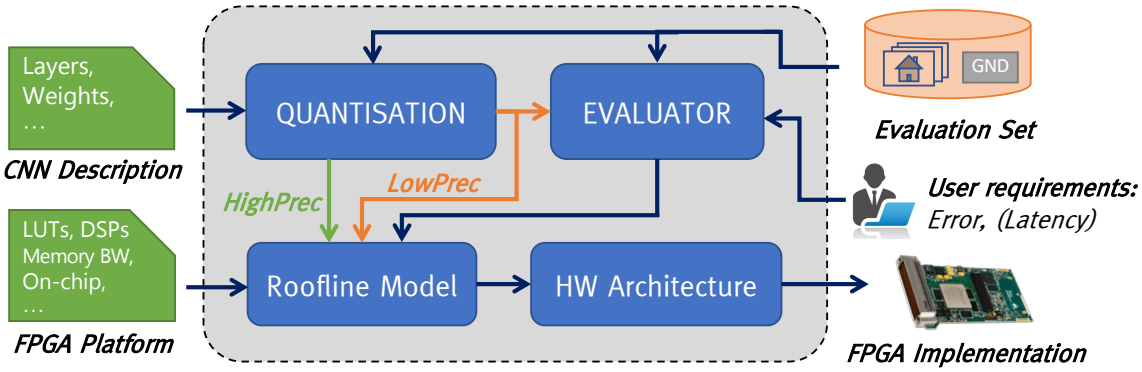


Figure 3.1: High-level view of *CascadeCNN*'s automated toolflow.

In parallel to this work, [312] examines an edge case of this precision-oriented cascade philosophy, introducing a heterogeneous cascade system featuring a fully binarised FPGA-based high-throughput CNN implementation, alongside a floating point CPU-mapped high-accuracy network, with a trainable Perceptron acting as a confidence evaluator on the BNN predictions.

All the aforementioned approaches exploit different approximation dimensions for input-dependent inference. However, in contrast to *CascadeCNN*, they still rely on training-time methodologies or model fine-tuning, continuing to assume access to the full training set of the target application. Only exception is the work of [274], that tackles input-dependent computation from a higher-level network selection perspective. In [274], a cascade is formed by adopting several pre-trained models with varying performance-accuracy characteristics from the literature (e.g AlexNet, GoogleNet and Resnet) and interconnecting them in a ascending order in terms of workload, interleaving trainable confidence evaluators between the models. This approach is not particularly suitable for embedded deployment, as edge device would severely struggle to execute the models from the latter cascade stages due to their excessive computational demands. Conversely, *CascadeCNN*'s precision-oriented approach offers significant efficiency gains, being directly realisable through the proposed custom hardware architecture. Furthermore, the proposed approach is orthogonal to accuracy-oriented model cascades and can be applied independently to each of their stages to improve their performance.

3.4 Methodology

3.4.1 Overview

This chapter approaches the challenge of efficient quantised inference from an input-dependent computation angle, though an automated toolflow that provides a holistic software-hardware solution, tailored to a user-specified CNN-FPGA pair, and application level requirements (in accuracy, latency etc), without the need to access the training data of the original model. A high-level view of *CascadeCNN*'s processing flow is depicted in Fig. 3.1.

More formally, *given*:

- A high-level description of a trained CNN model (i.e. Caffe model).
- The available computational and memory resources of the target FPGA platform (LUTs, DSPs, Memory bandwidth, on-chip and off-chip memory capacity)
- An application-level error tolerance in a user-defined metric (e.g. top1/top5 classification error), optionally accompanied by an inference latency constraint.
- A limited set of labelled data forming a calibration/evaluation set for the target application.

CascadeCNN searches the architectural design space and *provides* a two-stage hardware architecture, optimised for the particular CNN model and target device, along with a complete parametrisation of other tunable variables. The generated system for CNN inference (Fig. 3.2) consists of:

- A low-precision unit (LPU) which employs excessively low-precision arithmetic to trade lower accuracy with high throughput.
- A high-precision unit (HPU) which achieves the same accuracy level as the reference model.
- A tunable Confidence Evaluation Unit (CEU) that detects samples that were wrongly classified by the LPU and redirects them to HPU for re-processing.

The key idea behind the proposed approach is that the LPU will process the whole workload, while the HPU will only process a fraction of it based on the CEU's evaluation of classification confidence on LPU's predictions, reducing its compute requirements. At the same time, the accuracy loss that is induced due to the extreme model quantisation of the LPU is restored to meet the user-specified error threshold.

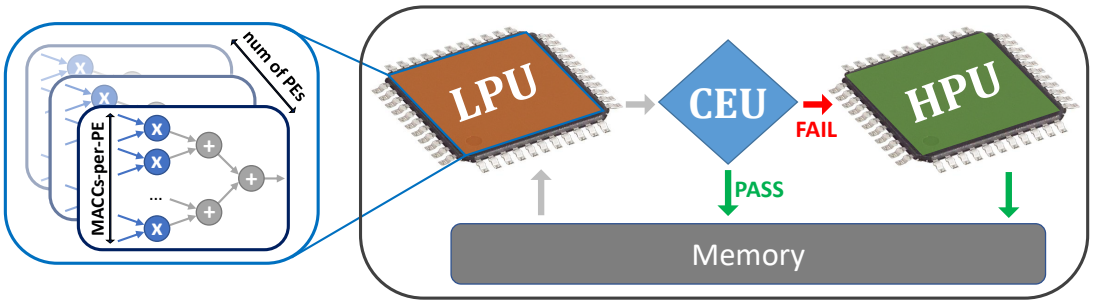


Figure 3.2: High-level view of *CascadeCNN*'s system architecture.

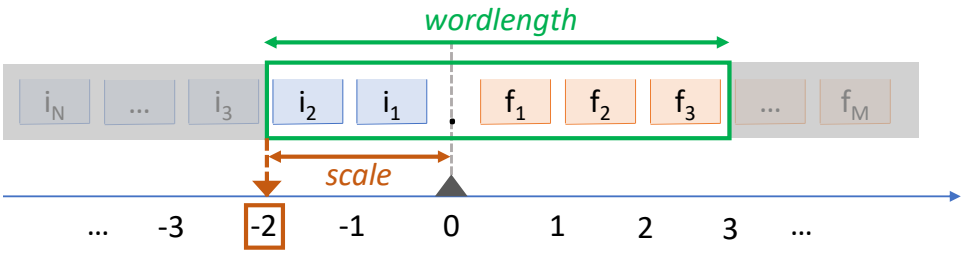


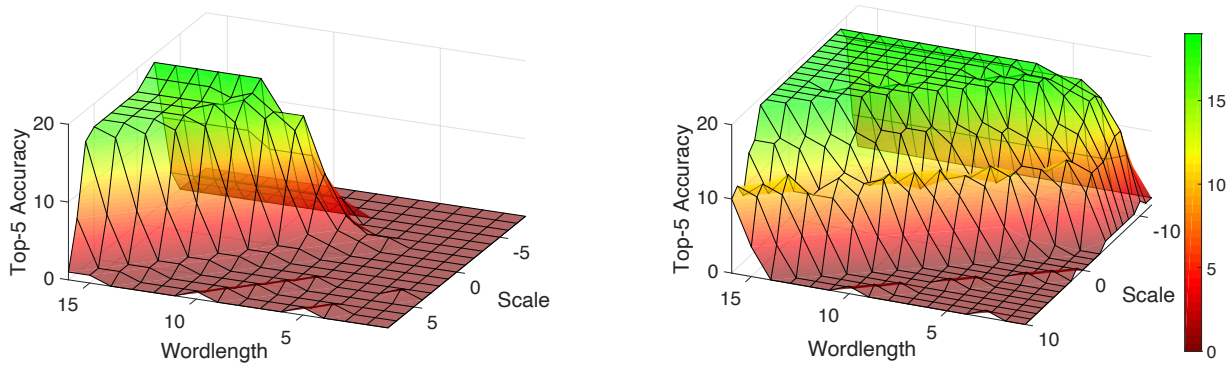
Figure 3.3: Parametrisation of *CascadeCNN*'s block floating-point quantisation scheme.

3.4.2 Quantisation

Arithmetic precision reduction is a widely studied technique that exploits the inherent redundancy of CNNs in order to reduce the memory footprint and bandwidth requirements, minimise power consumption and achieve higher performance. These gains are enlarged when fixed-point arithmetic is employed in place of its commonly used floating-point counterpart, leading to more resource-efficient hardware implementations and, in turn, higher parallelism.

CascadeCNN exploits a fine-grained search space across precision quantisation schemes for each of its processing units. This is formed by allowing different scaling factors for the weights and activations of each layer, which determine the ratio of integer and fractional bits, with a uniform wordlength per processing unit across all CNN layers (Fig. 3.3). The flexibility provided by this *block floating-point* approach, allowing non-uniform scaling factors across different CNN layers, enables smaller wordlengths to achieve high classification accuracy due to the different scaling factors in each layer ℓ, which would otherwise suffer significant accuracy loss if uniform scaling factors were employed across all layers. Moreover, employing a uniform wordlength W across all the layers of the target CNN allows the derivation of a single hardware architecture for each processing unit that can be time-shared between layers, mitigating the hardware overheads of multi-precision support. The selected quantisation scheme of each layer can be described by the following tuple:

$$q_\ell =< W^{(model)}, S_\ell^{(weights)}, S_\ell^{(activations)} > \tag{3.1}$$



(a) Uniform **(b)** Single Layer

Figure 3.4: Accuracy across CascadeCNN's quantisation space for VGG-16, examined across 20 randomly selected samples of the ImageNet validation set.

To determine the per-layer scaling factors for every explored wordlength of a given CNN, statistics regarding the quantisation impact of each layer on the application-level accuracy are extracted using a small user-provided evaluation set. Each layer's weights and activations are progressively quantised with decreasing wordlength values, examining a wide range of scaling factors, while computations and parameters for the rest of the network's layers employ floating-point representation. Fig. 3.4 illustrates snapshots of *CascadeCNN*'s quantisation search space. The achieved classification accuracy of VGG-16 is demonstrated across quantisation schemes on a calibration set of 20 samples, for uniform scaling factors across the network (Fig. 3.4a), and for a single layer (Fig. 3.4b). As observed in this figure, useful conclusions about the dynamic range of each layer can be extracted even from a very small evaluation dataset; while the adopted block-floating point representation offers enhanced flexibility compared to uniform quantisation schemes.

The extracted per-layer statistics are used to guide the exploration towards the range of scaling factors that achieves the highest accuracy for each wordlength. In more detail, all quantisation schemes that exceed an accuracy threshold for each layer, are combined to form a search space for the particular network, which is heuristically searched to determine the classification scheme that achieves the highest network-level accuracy for the examined wordlength. This fine-grained quantisation space pushes the accuracy limits attainable by each wordlength, introducing a more graceful accuracy degradation in the low-bit representation range, compared to more uniform approaches.

CascadeCNN selects for the LPU an excessively-reduced wordlength quantisation scheme that achieves intermediate application-level accuracy, but with significantly higher performance when mapped on its custom precision-optimised hardware units. At run time, all input samples

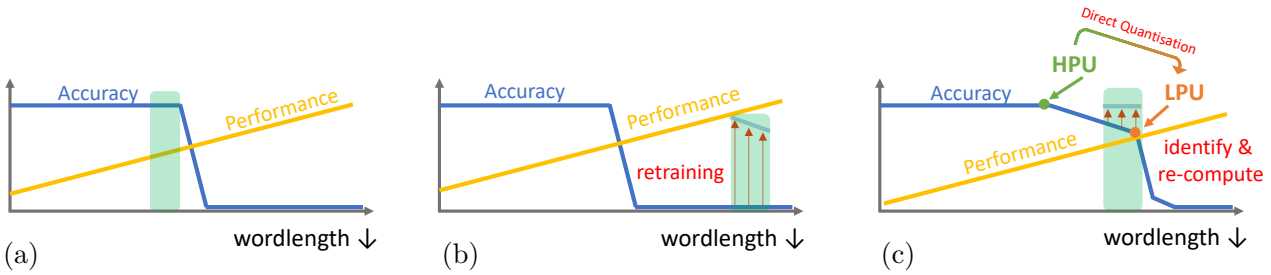


Figure 3.5: Illustrative comparison between Quantisation approaches: (a) Direct Quantisation, (b) Quantisation with Fine-tuning, (c) Quantisation in CascadeCNN. Green highlight indicates the most common operational sweet-spot.

are processed by the LPU in order to obtain a rapid classification decision, which is then fed to the CEU. A wordlength that achieves an accuracy that complies with the user-specified error margins is selected for the HPU (Fig. 3.5c). This "identify and recompute" scheme for missclassifications of the quantised model significantly differs from traditional direct quantisation approaches uniformly employing a wordlength that does not impose any accuracy degradation (typically 16 or 8 bits) (Fig. 3.5a); or Quantisation with retraining where the impact of excessive quantisation (all the way to 1 bit) is partly amortised through model fine-tuning, assuming the availability of training data (Fig. 3.5b).

Moreover, by imposing relevant constraints on the quantisation space, the reduced-precision model employed by *CascadeCNN*'s LPU can optionally be derived from the HPU's CNN by direct quantisation (without retraining), at run-time. As a result of this weight-sharing approach, the model size and associated memory size requirements of the proposed cascade approach can remain the same as in the case of a single stage accelerator employing the precision selected for the HPU, for use-cases with strict memory space limitations.

3.4.3 Confidence Estimation

CascadeCNN allows the exploration of extreme quantisation schemes for the LPU, under the assumption of no possible re-training of the model, by aiming to identify potentially misclassified inputs based on the confidence of the LPU classification prediction. Low-confidence predictions, identified at run time by the CEU, are re-processed by the HPU to restore classification accuracy.

This section builds on the work of [176] by generalising the proposed Best-vs-Second-Best (BvSB) metric. BvSB was previously examining solely the distance between the two highest probability values, which is mostly applicable when solely considering top-1 accuracy and fails to

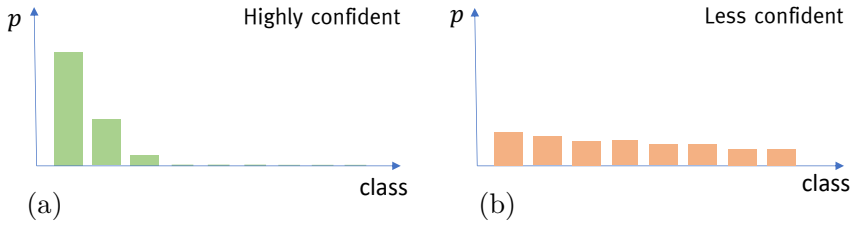


Figure 3.6: Indicative comparison between (sorted) probability distributions of (a) a confident and (b) an uncertain CNN prediction.

capture important characteristics of overall probability distribution. The proposed generalised BvSB (gBvSB) metric is defined as:

$$\text{gBvSB}_{<M,N>}(\mathbf{p}) = \sum_{i=1}^{M} \mathbf{p}_i - \sum_{j=M+1}^{M+N} \mathbf{p}_j \tag{3.2}$$

where $\mathbf{p}_i$ denotes the i-th largest element in the (sorted) probability distribution $\mathbf{p}$ across the examined N_{class} classes provided by the model, and M and N are tunable parameters of gBvSB.

Intuitively, the degree of uncertainty of the classification decision is determined by examining how "spiky" the sorted probability distribution of a CNN's prediction is (Fig. 3.6). Accordingly, a prediction from the LPU is considered confident, and thus the processing ends there, when:

$$\text{gBvSB}_{<M,N>}(\boldsymbol{p}) \geq th \tag{3.3}$$

where threshold th forms an additional tunable parameter, the value of which is automatically determined along with M and N by the proposed toolflow, using the provided evaluation set to meet the user-specified error tolerance.

An instance of this space is illustrated in Fig. 3.7, indicating the gBvSB for VGG-16 predictions (quantised to 5-bits with the methodology described in Sec. 3.4.2) across the validation set of ImageNet [21]. As illustrated, misclassified cases are uniformly distributed across the evaluation set. As such, a subset of as little as 200 samples was consistently proven adequate to provide an estimate of the percentage of misclassified cases that each CEU instance will fail to capture.

It should be noted that a trade-off exists between the number of samples that were correctly classified by the LPU but forwarded for re-processing to the HPU due to low confidence (false-negatives), and the number of misclassified samples that the CEU failed to identify due to the CNN's overconfident prediction (false-positives). The latter cases are responsible for the classification error that is introduced by *CascadeCNN*, and can be reduced by setting the CEU

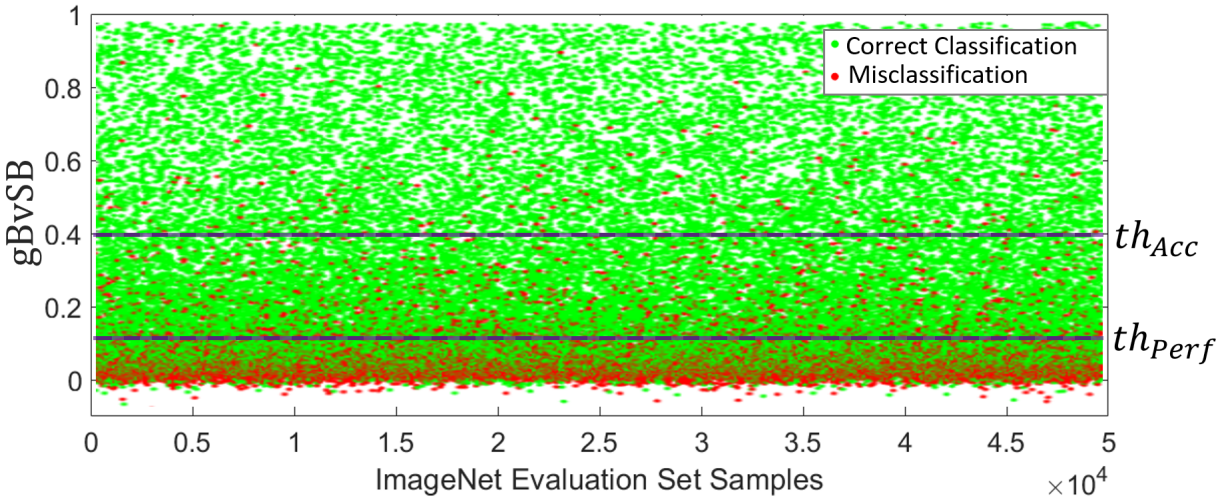


Figure 3.7: $\text{gBvSB}_{<5,5>}$ metric values across the ImageNet validation set for a VGG-16 5-bit LPU.

parameters to a configuration that terminates the computation only on remarkably confident predictions (th_{Acc}). However, this would significantly increase the number of correctly classified inputs that are re-processed, which comes with a performance penalty on the system. In contrast, a configuration that only suggests re-processing on the extremely under-confident inputs, maximises performance at the cost of degraded classification accuracy (th_{Perf}). This trade-off (depicted in Fig. 3.7) is exploited to achieve the highest performance satisfying the user-specified error tolerance.

3.4.4 Hardware Architecture

To effectively realise the performance gains of quantisation, a scalable, fine-grained hardware architecture for CNN inference is designed, being able to scale its performance with the resources of a target FPGA and exploit higher degrees of parallelism as the wordlengths of activations and weights decrease. The core of the architecture is a matrix-multiplication (MM) engine, parametrised with respect to the tiling of each matrix dimension and the arithmetic precisions for activations and weights. The MM engine comprises Multiply-Accumulate (MACC) units, grouped into Processing Elements (PEs) that perform dot-products (Fig. 3.2). By casting convolutions as matrix multiplications using Toeplitz matrices (also known as im2col algorithm) and resorting to batch processing for dense layers, both Convolutional (CONV) and Fully-Connected (FC) layers are mapped on the same MM unit. Particularly, every CONV layer can be described as:

$$CONV < H, W, N_{in}, N_{out}, K_H, K_W, S_H, S_W, Z > \tag{3.4}$$

where H and W denote the height and width of each input feature map, N_{in} and N_{out} the number of input and output feature maps respectively, K_H and K_W the height and width of the layer's kernels, S_H and S_W the stride size in the height and width directions of each input respectively and Z the zero-padding on the input feature maps. FC layers can be described similarly to CONV layers under the following formulation:

$$FC = CONV < 1, 1, N_{in}, N_{out}, 1, 1, 1, 1, 0 > \tag{3.5}$$

For each CONV layer, a $R_{CONV} \times P$ input activations matrix is constructed to represent its input feature maps. The input activations matrix is organised with each row containing the unrolled feature map values for a single sliding window position, concatenated for all input channels. The number of sliding window positions of a layer can be calculated as:

$$R_{CONV} = \left\lceil \frac{H + 2 \cdot Z - (K_H - 1)}{S_H} \right\rceil \cdot \left\lceil \frac{W + 2 \cdot Z - (K_W - 1)}{S_W} \right\rceil \tag{3.6}$$

while the number of elements included in the unrolled sliding window for all channels of the input feature map volume is:

$$P = K_H \cdot K_W \cdot N_{in}. \tag{3.7}$$

In the case of FC layers, batching is employed to form a similar $R_{FC} \times P$ matrix, each row of which contains the input feature vector of size P for a different input sample. Letting B denote the batch size used upon deployment, then:

$$R_{FC} = B. \tag{3.8}$$

Similarly each layer's weights are unrolled to form a $P \times C$ weights matrix, with each of the C columns corresponding to a different kernel, expressed as a P-element vector of concatenated unrolled values for N_{in} channels. Hence, the number of columns of $P \times C$ matrix is identical to the number of output feature maps of the layer:

$$C = N_{out} \tag{3.9}$$

As a result of the multiplication between the $R \times P$ and $P \times C$ matrices, a $R \times C$ matrix is produced, with each column representing the output feature map values produced by the different filters of the layer (Fig. 3.8-top). Hence, the output of each layer, acting as an input to the next layer, already adopts the required Toeplitz matrix form. Accordingly, the weights of each layer can

be converted to their matrix forms offline once, leaving the need for runtime im2col only on the input images. In the context of this chapter, this is transformation is conducted on the CPU, although more efficient alternatives can also be adopted from the literature [313].

The *MM* engine employs tiling across all three dimensions (R, P, C) with tile sizes of T_R, T_P, T_C accordingly. Input tiles $T_R \times T_P$ and $T_P \times T_C$ are loaded from the off-chip memory employing double buffering to hide the memory latency between the processing of different tiles. Moreover, to sustain a high memory bandwidth utilisation in the case of quantised weights and activations, the low-precision values are packed into larger chunks that match the memory ports width (e.g. 4-bit values packed in chunks of 16 for 64-bit ports).

Fig. 3.8 shows the proposed parametrised hardware architecture. A tile of the output activations matrix ($T_R \times T_C$) is computed by accumulating the results of $\left\lceil \frac{P}{T_P} \right\rceil$ tile multiplications. Intermediate results are kept on-chip, and only the final output tiles are transferred back to the off-chip memory. Each tile multiplication consists of a set of dot-product operations. The proposed architecture exploits two different levels of parallelism to allow for a fine-grained architectural search space. To exploit the fine-grained parallelism in dot-products, the dot-product between T_P-element vectors is fully unrolled and mapped on a PE that contains a multiplier array followed by an adder tree. At the same time, T_C PEs are instantiated concurrently, to parallelise the dot-products between all columns of the $T_P \times T_C$ tile with a single row of the $T_R \times T_P$ tile. Finally, the rows of the $T_R \times T_P$ tile are processed in a pipelined manner in order to maximise throughput.

Precision-aware mapping of PEs to FPGA resources. In contrast to the majority of the existing FPGA-based designs, the proposed architecture utilises both LUTs and DSPs to implement its MACC units. A similar strategy was partially explored in [289], focusing solely on 16-bit operands. With smaller wordlengths being less LUT-costly, employing LUT-based arithmetic units alongside DSP-based units enables the proposed architecture to reach higher performance by instantiating higher number of MACC units as the wordlength decreases. Moreover, for the LPU, this chapter introduces a strategy that packs a pair of low-precision MACCs (less than or equal to 5 bits) on a single 25×18 DSP, by positioning targeted zero guard-bits between the input operands to avoid interference on the results and, in this way, doubling the computational capacity of the FPGA's DSP blocks. This technique is enabled by the extreme quantisation employed by the LPU, since larger wordlengths would either limit its applicability only to MACC operations with a shared operand [314], or restrict it completely.

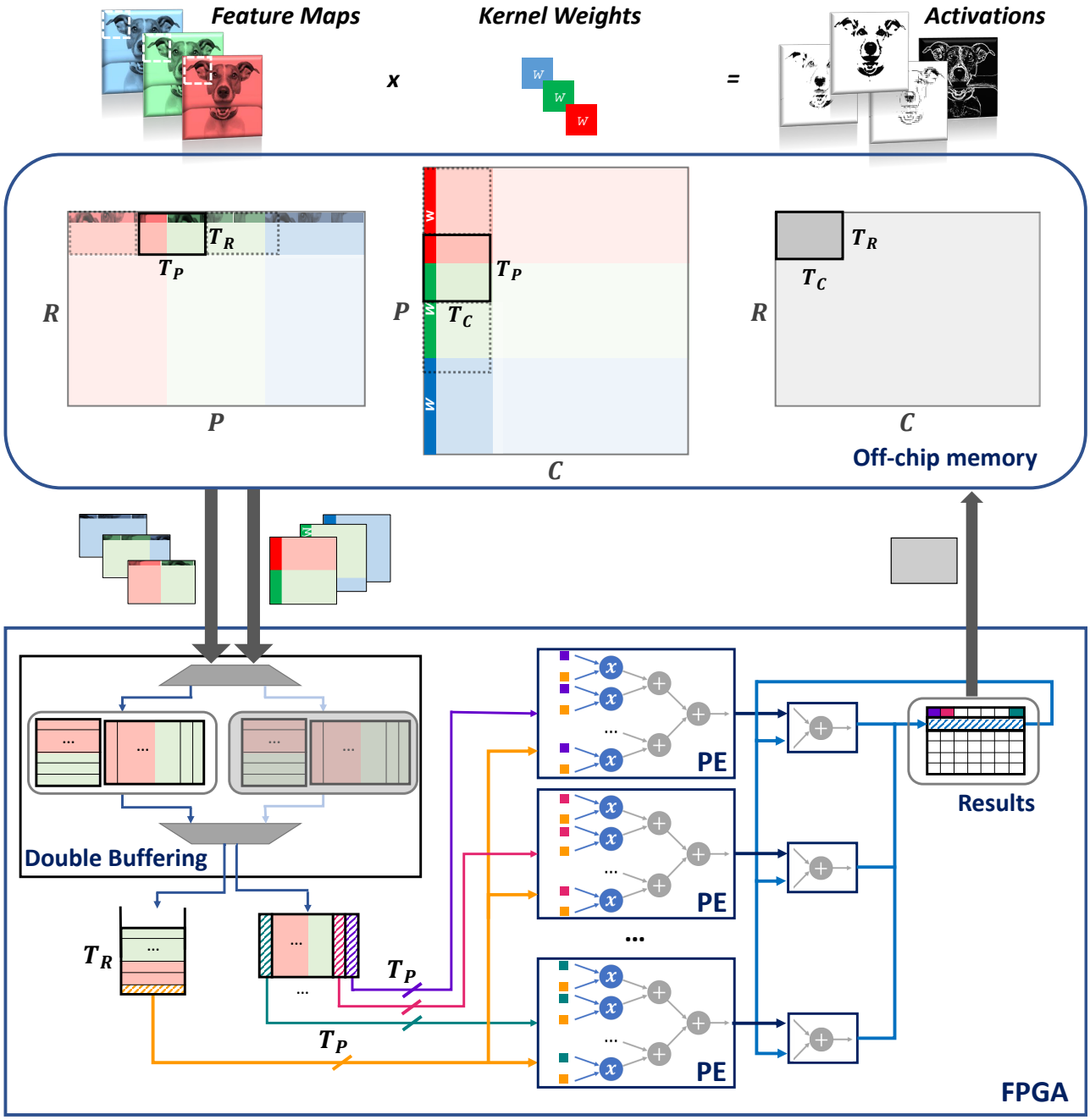


Figure 3.8: Overview of proposed hardware architecture for CascadeCNN.

3.4.5 Design Space Exploration (DSE)

Given a CNN-FPGA pair and a particular wordlength, *CascadeCNN* searches the architectural design space by means of a roofline-based performance model [315] in order to determine the highest performing configuration of the architecture. In the context of this subsection, optimisation is focused on inference throughput. The configurable parameters comprise the matrix tile sizes T_R, T_P, T_C, that determine the pipeline depth, MACCs-per-PE and number of PEs instantiated

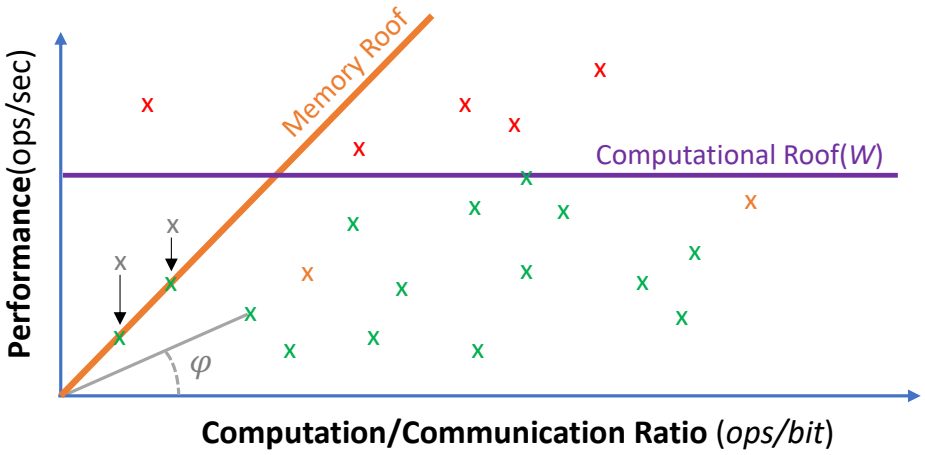


Figure 3.9: Illustration of the roofline performance model.

respectively, as described in Sec. 3.4.4, along with a tiled batch size to be discussed later in the section. In this manner, *CascadeCNN* generates two architectures, the LPU and the HPU, which are optimised for different wordlengths W_{LPU} and W_{HPU} for the provided CNN-FPGA pair.

Tile Sizes

Having parametrised the *MM* operation and its underlying architecture accordingly, the design space is explored to determine the highest performing tile size tuple $<T_R, T_P, T_C>$ for each CNN-FPGA-precision instance. A per-layer analysis is initially performed, the results of which are combined to yield a shared architecture across all layers, which maximises the overall throughput. To enable fast and exhaustive DSE, an analytical performance model of the underlying architecture is developed. Based on that, the roofline model (Fig. 3.9) is adopted to associate the predicted performance of each architectural configuration with its operational intensity (which provides an estimate of the ratio between computational workload and memory traffic), to obtain the highest performing design point, aware of the selected wordlength.

The attainable performance (roofline y-axis) of a parametrisation of the accelerator architecture, defined as a triplet of tile sizes $< T_R, T_P, T_C >$, on a specific CNN layer ℓ, defined by its matrix dimensions $< R_\ell, P_\ell, C_\ell >$, is estimated analytically as:

$$Perf_\ell = \frac{workload_\ell}{cycles_\ell} = \frac{2 \cdot R_\ell \cdot P_\ell \cdot C_\ell}{R_\ell \cdot \left\lceil \frac{P_\ell}{T_P} \right\rceil \cdot \left\lceil \frac{C_\ell}{T_C} \right\rceil} \cdot clk(W) \quad (ops/s) \tag{3.10}$$

where $workload_\ell$ denotes the total number of operations required for processing a single input through layer ℓ during inference. Workload is model-dependent and expressed in ops/input, with

each MACC accounting for two operations (one multiplication and one addition). $cycles_\ell$ denote the number of cycles required for processing this input sample through layer ℓ (also known as the Initialisation Interval of the layer) and is also dependent on the architectural configuration of the accelerator. Finally, $clk(W)$ is the achieved clock frequency of the architecture, depending on the FPGA device and selected wordlength.

As can be seen in Algorithm 1 (describing the tiled *MM* approach) the instantaneous rate of read and write memory transactions varies throughout the computation. To deal with this imbalance, the performance analysis is configured to treat each iteration of *loop2* as a single step, capturing the total memory traffic and processing load required for the computation of an output tile ($T_R \times T_C$). In order to serve the fine-grained quantisation space, the roofline model is adjusted to employ bits, instead of bytes, as its data measurement unit. Hence, the Computation-to-(off-chip memory) Communication (CTC) ratio (also known as operational intensity; roofline x-axis) of the architecture for each such step is formulated as:

$$CTC_\ell = \frac{tileOps_\ell}{tileMem_\ell} = \frac{2 \cdot T_R \cdot P_\ell \cdot T_C}{(T_R \cdot P_\ell + P_\ell \cdot T_C + T_R \cdot T_C) \cdot W} \quad (ops/bit) \tag{3.11}$$

where $tileOps_\ell$ denotes the total amount of operations required for the computation of an output tile (*ops/tile* in a single iteration of *loop2*) of layer ℓ, while $tileMem_\ell$ denotes the number of bits that needs to be transferred between the off-chip memory and the FPGA in bits/tile, during the same computation.

Algorithm 1 CNN Layer as Tiled Matrix Multiplication

```
for r = 1 to ⌈R/T_R⌉ step T_R do                                   ▷ loop1
    for c = 1 to ⌈C/T_C⌉ step T_C do                               ▷ loop2
        RegRC ← zeros(T_R×T_C)
        for p = 1 to ⌈P/T_P⌉ step T_P do                           ▷ loop3
            RegRP ← memRead(T_R×T_P)
            RegPC ← memRead(T_P×T_C)
            for rr = 1 to T_R step 1 do                            ▷ loop4
                for cc = 1 to T_C step 1 do                        ▷ loop5
                    RegRC(rr, cc)+=dot_product(RegRP(rr), RegPC(cc))
                end for
            end for
        end for
        memWrite(RegRC)
    end for
end for
```

By enumerating the possible tile size combinations and calculating the corresponding $Perf$ and CTC values, the design space is formed for the examined CNN layer and precision. Considering the target FPGA resources, only a subspace of this design space corresponds to platform-supported architectures. The limiting factors of the platform are its peak attainable performance (also known as Computational Roof), obtained by resource modelling for a wide range of precisions, and memory (on-chip storage, off-chip bandwidth) resources, as well as the implementation-specific resource utilisation per computational element, obtained through benchmarking. Specifically, MACCs are implemented on both DSPs and LUTs (Sec. 3.4.4), hence:

$$compRoof = 2 \cdot \left(\frac{availLUT}{LUTperMACC(W)} + \frac{availDSP}{DSPperMACC(W)} \right) \cdot clk(W) \quad (ops/s) \tag{3.12}$$

where $availLUT$ and $availDSP$ denote the number of available resources on the target device, while $LUTperMACC(W) \in \mathbb{N}^*$ and $DSPperMACC(W) \in \{1/4, 1/2, 1, 2, 4\}$ indicate the precision-specific resource consumption of a single MACC unit when implemented on LUTs and DSPs respectively (obtained through benchmarking). With respect to the on-chip storage, the on-chip memory requirements for each design point is obtained as:

$$onChipMem(bits) = 2 \cdot (T_R \cdot T_P + T_P \cdot T_C + T_R \cdot T_C) \cdot W \tag{3.13}$$

where the term 2 accounts for double buffering.

Design points exceeding the platform's computational roof ($\exists \ell \in [1, N_{layers}] : Perf_\ell >$ CompRoof) or on-chip memory capacity are excluded from the platform-supported design space (illustrated as red and orange points in Fig. 3.9 respectively). Additionally, for design points whose bandwidth requirement exceeds the available (illustrated as grey points), their attainable performance is vertically projected to meet the platform's memory roof at $\widehat{Perf_\ell}$ (maintaining its CTC_ℓ value), while $cycles_\ell$ in Eq. 3.10 need to be adjusted accordingly as:

$$cycles_\ell = \frac{workload_\ell}{\widehat{Perf_\ell}} \tag{3.14}$$

After completing the DSE for every layer of a given CNN using a particular precision, a single tile size triplet that maximises the overall throughput across all layers needs to be determined. Nevertheless, since significant variability is observed in the matrix dimensions of different layers, different tile sizes yield the highest performance for each layer. This can be explained by the fact that deeper layers tend to have more and smaller filters in contrast with the first

layers that consist of fewer filters, operating on larger feature maps. To calculate the average attainable performance of a design point with fixed tile sizes across all layers, a weighted average based on the percentage of processing cycles each layer is contributing for the computation of a single batch of inputs is employed:

$$ovrlPerf = \frac{\sum_{\ell=1}^{N_{layers}} k_\ell \cdot cycles_\ell \cdot Perf_\ell}{\sum_{\ell=1}^{N_{layers}} k_\ell \cdot cycles_\ell} \quad (ops/s), \tag{3.15}$$

$$k_\ell = \begin{cases} B & \text{, if } \ell \text{ is CONV} \\ 1 & \text{, if } \ell \text{ is FC} \end{cases} \tag{3.16}$$

where $cycles_\ell$ is the number of cycles for processing a single input sample by the l-th layer, as described in the denominator of Eq. 3.10 and refined for memory-bounded layers in Eq. 3.14. The platform-supported design point achieving the maximum $ovrlPerf$ is selected for the implementation of the examined CNN-FPGA-precision triplet.

Batch Size

As discussed, input samples are processed by *CascadeCNN* in batches. In the context of this section, all samples of a batch are first processed by the LPU, with the CEU estimating the confidence of each prediction. Subsequently, the FPGA is reconfigured with the optimised hardware architecture for the precision employed by the HPU, which processes a fraction of the input samples, based on the CEU's response. In this process, also illustrated in Fig. 3.10a, the reconfigurability of FPGAs is exploited to enable switching between highly optimised processing units -tailored for the selected LPU and HPU wordlengths- at runtime, while occupying nearly all the available resources of the target platform to exploit more parallelism. Nevertheless, each full reconfiguration of the device introduces substantial latency delay that can aggravate the overall system performance. Similarly to relevant works making use of device reconfiguration upon deployment [65], employing large batch sizes is used to allow multiple samples to be processed by each unit between reconfigurations. This compensates for the cost of reconfiguring the device (as reconfiguration time becomes negligible compared to the overall processing time of the batch). Thus, the largest batch size B that can be accommodated by the off-chip memory (considering both inputs and intermediate results) is selected.

However, large batch size values affect the R dimension of FC layers' input activations matrices, which can aggravate the load imbalance between layers, and thus degrade the average performance

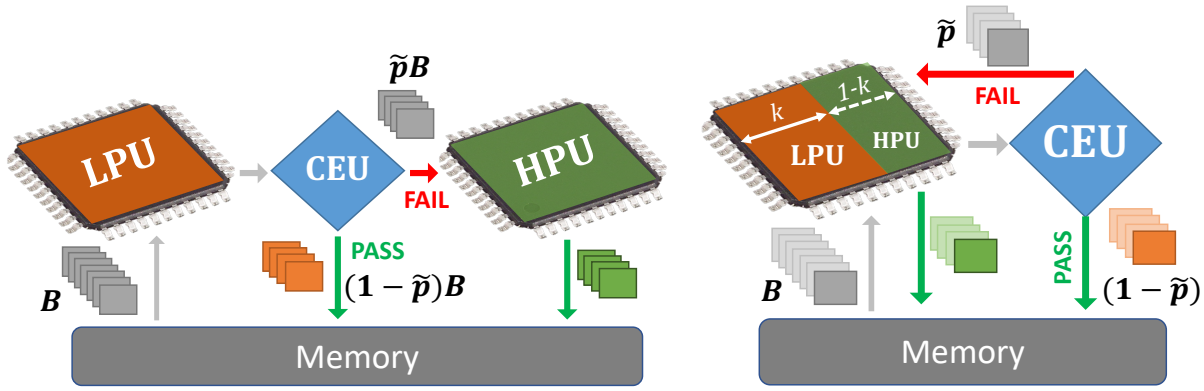


Figure 3.10: Different CascadeCNN Deployment settings for (a) Throughput-driven and (b) Latency-aware Optimisation.

of the derived architecture. To address this issue, tiling of the batch size is employed. A corresponding tile size T_{Batch} is further parametrising the design space to ensure better load balance across layers, with T_{Batch} replacing B in Eq. 3.8. To preserve the capability of exhaustively searching the design space, while maintain support for a wide variety of batching scales, only values that are powers of two or multiples of 1024 are examined for the optimisation T_{Batch}. The tiling factor of the system's batch size is selected to maximise the average attainable performance. The resulting system's LPU processes the whole batch size B in groups of T_{Batch} samples, before switching to the HPU, in order to achieve better load balance between CONV and FC layers.

Deployment Performance

CascadeCNN is fundamentally different from most CNN accelerators in the literature [285][316][317][65], typically adopting a faithful single-stage mapping of the target DNN to customised processing elements. In such "HPU-only" systems following a uniform compute paradigm across samples (Fig. 3.11a), the typical performance metrics can be used, with throughput ($Thpt$) being inversely proportional to the latency (t_H) of the implementation:

$$t_{avg} = t_H, \quad Thpt = wkld/t_H \tag{3.17}$$

In contrast, the two-stage *CascadeCNN* architecture, follows an input-dependent computational scheme, featuring batch processing and device reconfiguration between its two stages (Fig. 3.11b). Therefore, the performance metrics need to be re-shaped accordingly to effectively capture its throughput an latency:

$$\begin{aligned} t_{avg} &= t_L + \tilde{p} \cdot \Big((B-1)/2 \cdot t_L + t_{Rcfg} + \tilde{p} \cdot (B-1)/2 \cdot t_H + t_H \Big) \\ Thpt &= B \cdot wkld / (B \cdot t_L + \tilde{p} \cdot B \cdot t_H + t_{Rcfg}) \end{aligned} \tag{3.18}$$

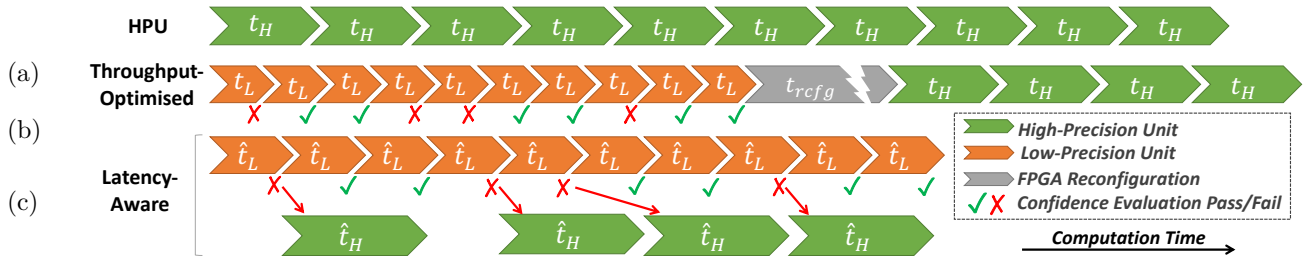


Figure 3.11: Processing timeline of a 10-sample input instance by: (a) Baseline HPU, (b) Throughput-optimised CascadeCNN and (c) Latency-aware CascadeCNN.

where t_L and t_H denote the response latency of LPU and HPU respectively, when deployed across the resources of the target FPGA and t_{Rcfg} the board-specific reconfiguration time. As depicted in Fig. 3.11b, a proportion $\tilde{p}$ (dependent on the CEU tuning and therefore on the user-specified error tolerance) of the batch B is forwarded by the CEU to the HPU for re-processing. As a consequence of the throughput gains achieved through this approach, the inference latency of these samples is negatively affected by factors including: (i) the time for processing the remainder of the batch on the LPU (consisting of $(B\text{-}1) \cdot \tilde{p}/2$ samples on avg. case), (ii) device reconfiguration, (iii) "queuing time" while other ($(B\text{-}1) \cdot \tilde{p}^2/2$ on avg.) samples of the batch requiring re-processing are served, and iv) HPU re-processing time.

3.4.6 Latency-Aware DSE

Although forming large batches is proven an effective strategy to remedy the overhead of device reconfiguration and sustain high inference throughput, latency is substantially affected. This overhead is particularly evident in the portion of input samples forwarded to the HPU for re-processing, as apart from the direct impact of batching, the reconfiguration time (being in the order of seconds) is also accumulated on inference latency. This approach is therefore unsuitable for latency-sensitive applications, involving real-time mission critical decision making, such as robot navigation and autonomous driving.

Despite the fact that model cascades are not well-suited for latency optimisation, to address this issue this section re-visits the DSE methodology of *CascadeCNN* in an alternative formulation that also considers a user-provided latency constraint, aiming to strike a better balance between the attainable throughput and latency overhead of the proposed approach. This new formulation is particularly useful for visual tasks related to autonomous systems such as mobile robots, which frequently impose competing high-throughput and low-latency requirements, calling for

throughput-latency co-optimisation. In Simultaneous Localisation and Mapping (SLAM) for example, where DL models are emergently incorporated to enhance the constructed map with semantic information [77], a high frame-rate (throughput) is necessary to sustain robust localisation (tracking of a mobile agent's position and orientation) [318], while a controlled response time is also required [319] to avoid prolonged delay between sensing and incorporation of the perceived information to the map, which may regularly be used for other downstream tasks, such as grasping.

To remedy the latency burden of the design flow introduced in the previous section, here a resource-sharing approach is employed to replace the previously compulsory device reconfiguration between the LPU and HPU deployment stages and its consequential need for input-batching, enabling its deployment to latency-sensitive scenarios. The proposed method, introduces a configurable custom hardware architecture in which the LPU and HPU are deployed concurrently on the same FPGA device (Fig. 3.10b), forming a two-stage cascade. Alongside, this chapter adopts a novel multi-stage Design Space Exploration methodology that repeatedly expands and prunes the design space to arrive at a set of platform-supported {LPU,HPU} pairs, out of which a resource-efficient and highly-optimised architectural configuration is selected, maximising inference throughput while satisfying user-specified latency and accuracy requirements. This approach is effectively reducing the complexity of an exhaustive configuration search, maintaining the ability to identify the best cascade design point. In more detail, the proposed approach consists of the following stages:

(a) Independent LPU and HPU Performance Analysis. Starting from the same single-stage hardware architecture, initially, an exhaustive performance analysis of all possible architectural configurations is conducted for the selected LPU and HPU precision (wordlength; W) independently, using the roofline model introduced in Sec. 3.4.5. In this manner, each design point's computational and memory demands are fully characterised for both units.

(b) LPU and HPU Design Space Pruning. Having expanded to all possible architectural configurations of each unit, the subspace of platform-supported design points is identified identified as before. The main difference is that in this case memory-bounded design points are directly discarded, due to their poor resource efficiency that would lead to starvation of the available off-chip memory bandwidth. Additionally, FC layers also become severely more memory-bounded in the absence of a large batch size, alleviating the cost of their weight reads from the memory.

As such, and consistently with common practise on other latency-driven accelerators [298] [317], in the latency co-optimised setting of CascadeCNN FC layers are mapped to the CPU.

(c) Exploration of Cascaded LPU-HPU Combinations. Different from the throughput-optimised DSE case and relevant literature employing single-stage hardware architectures (e.g. [320]), that would select a design point optimising the performance objective at this point of the analysis, a resource sharing approach is employed targeting to fit an {LPU,HPU} pair concurrently on the target device. Therefore, having identified the sets of platform-supported design points DP_{LPU} and DP_{HPU} for the LPU and HPU accordingly, the combinatorial pair space of {LPU, HPU} = $DP_{\text{LPU}} \times DP_{\text{HPU}}$ is explored. To reduce this search space, individual design points demonstrating excessively low performance or approaching the platform's CR are discarded.

Providing that the LUT utilisation for a single MACC operation continuously scales with wordlength, whereas DSPs can only take advantage of excessively low-precision arithmetic through discrete packing of MACCs [314], the proposed framework evaluates the LPU and HPU wordlengths, in accordance to the developed resource utilisation models, in order to perform an efficient allocation of FPGA resources between the two units.

(d) Cascade Design Space Pruning. In the proposed approach, the LPU and HPU operate concurrently on the same device, being allocated dedicated computational and on-chip memory resources but competing for the off-chip memory bandwidth (BW). Consequently, all {LPU, HPU} pairs whose cumulative computational resource requirements are exceeding the availability of the target platform are discarded. Furthermore, each unit processes a different sample and may perform calculations corresponding to different layers of the CNN at each instant. Therefore, this chapter models the joint memory needs of both units, considering the BW requirement ($bits/s$) of each design point as the ratio between CTC_ℓ and $Perf_\ell$ (equivalent to the tangent of ϕ, as illustrated in Fig. 3.9):

$$BW_\ell = \tan\left(\measuredangle(CTC_\ell, 0)(0,0)(CTC_\ell, Perf_\ell)\right) = \tan(\phi_\ell) = \frac{CTC_\ell}{Perf_\ell} \tag{3.19}$$

Two modelling approaches are investigated: (i) *worst-case*: considers the maximum BW requirement of each model:

$$BW = \max_{\ell=1}^{NL}\left(\tan(\phi_\ell^{(LPU)})\right) + \max_{\ell=1}^{NL}\left(\tan(\phi_\ell^{(HPU)})\right) \tag{3.20}$$

while (ii) *avg-case* considers the average BW between layers, weighted w.r.t. each layer's workload:

$$BW = \frac{\sum_{\ell=1}^{NL} wkld_\ell \cdot tan(\phi_\ell^{(LPU)})}{\sum_{\ell=1}^{NL} wkld_\ell} + \frac{\sum_{\ell=1}^{NL} wkld_\ell \cdot tan(\phi_\ell^{(HPU)})}{\sum_{\ell=1}^{NL} wkld_\ell} \tag{3.21}$$

In either approach, design point pairs that overpass the target platform's available memory BW are discarded from the platform-supported design space.

(e) Multi-objective Optimisation Search. All the remaining {LPU,HPU} design pairs comprise platform-supported cascade design-points. The proposed methodology formulates the selection of the best architectural configuration as a multi-objective optimisation problem, in view of the attainable throughput ($Thpt$) and avg. latency (t_{avg}) of each cascade, to accommodate the application-specific requirements.

Considering the total workload $wrkld = \sum_{\ell=1}^{N_{layers}} wkld_\ell$ of the base model, the response latency of each individual design point for unit U can be estimated as: $\hat{t}_U = wkld/Perf^{(U)}$. Therefore, given the sample re-processing ratio $\tilde{p}$ characterising the employed confidence evaluator (configured to meet the user-specified error tolerance), the overall prediction latency of the cascade classifier is formulated as:

$$t_{avg} = \hat{t}_L + \tilde{p} \cdot \left(\hat{t}_H + \sum_{i=1}^{\lceil \hat{t}_H/\hat{t}_L \rceil} \tilde{p} \cdot (\hat{t}_H - i\hat{t}_L) \right) \tag{3.22}$$

where the last term accounts for the HPU "waiting time", when consecutive missclassifications of the LPU are queuing for re-processing to the slower HPU (Fig. 3.11c). However, since in the proposed system the LPU and HPU operate concurrently, the LPU can receive new inputs at a rate depending solely to its response latency:

$$Thpt = wkld/\hat{t}_L \ \ (ops/s) \quad | \quad \hat{t}_L \geq \tilde{p}\hat{t}_H \tag{3.23}$$

In order to effectively project the LPU's input rate to the output of the overall cascade system, the condition in the second part of Eq. 3.23, ensuring that the HPU is "fast enough" to accommodate the LPU's miss-classification rate and avoid stacking of samples, should be satisfied. Therefore, cascade design points violating this condition are abandoned.

By enumerating the remainder {LPU,HPU} pairs, the highest performing design point in terms of attainable throughput, that meets the user-specified latency constraint ($t_{avg} \leq t_{constraint}$) is selected for deployment by the design space exploration.

3.5 Evaluation

3.5.1 Experimental Setup

Experiments are conducted targeting image classification, using pretrained models on the ImageNet [21] dataset. The *CascadeCNN* toolflow (Fig. 3.1) is fed with AlexNet [20] and VGG-16 [36] models, along with a small subset of the labelled ImageNet validation set (2% of its samples) for calibrating the quantisation and confidence evaluation units. A wide range of values for the user-specified parameter of error tolerance, spanning between 1 and 5 percentage points (p.p.) in terms of top-5 accuracy, are considered.

The toolflow is built around Matlab (2017b), which is used to investigate the quantised fixed-point network behaviour, determine the highest achieving quantisation scheme for each wordlength, as well as to obtain network predictions, tune the CEU parameters and implement the proposed performance modelling and design space exploration methodology. All hardware designs were synthesised and placed-and-routed using the Xilinx Vivado HLS and Vivado Design Suite (v17.2) and evaluated on the Xilinx Zynq ZC706 and UltraScale+ ZCU102 boards. Performance measurements are conducted using the ARM CPU of the embedded FPGA platform, which is also used to schedule computations on the architecture's Processing Elements and set-up the off-chip memory transactions, as well as executing the FC layers in the case of latency-aware configuration (Sec. 3.5.4).

3.5.2 Architectural Performance and Accuracy

In this section, the performance and accuracy of the proposed architecture implementing *CascadeCNN*'s processing units, is evaluated as a function of the employed wordlength. For both AlexNet and VGG-16, *CascadeCNN* yields a wordlength of 4 bits for the LPU, by selecting the smallest wordlength that did not experience catastrophic accuracy loss during the quantisation scheme exploration (Sec. 3.4.2). This 4-bit LPU introduces a 14.38 percentage point (p.p.) and 18.65 p.p. degradation in classification accuracy compared to a 16-bit precision for AlexNet and VGG-16, respectively (Fig. 3.12). It should be noted that (similarly to what has been reported in [299]) equivalent accuracy can be achieved across the range between 16-bit and 8-bit wordlength implementations, provided by the employed quantisation methodology. As the resulting architectures for higher-precision units in the range of 16 bits demonstrate degraded

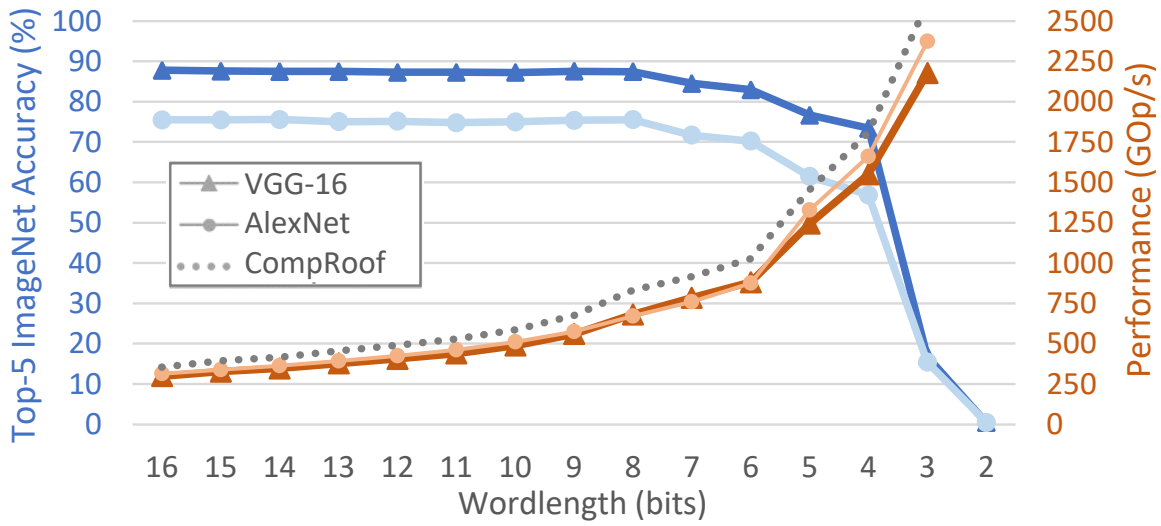


Figure 3.12: Achieved classification accuracy (blue) and measured performance (orange) as a function of wordlength on Zynq ZC706.

performance due to increased $LUTperMACC$ requirements and lower achieved clock frequencies, an 8-bit implementation is selected to act as a baseline for *CascadeCNN*'s error. The CEU parameters are tuned on the evaluation dataset to generate systems that introduce a wide range of classification errors, compared to the 8-bit baseline.

Using the roofline performance analysis of Sec. 3.4.5, the design points achieving the highest performance for AlexNet and VGG-16, for a wide range of wordlengths are extracted. Fig. 3.12 shows the measured performance on the ZC706 board and the achieved accuracy across multiple wordlengths. The fine granularity of the architectural design space allows the selected design points for all examined precisions to approach the device's computational roof. Furthermore, for low precisions (below 6 bits) the proposed architecture demonstrates increased scalability in its performance as a result of the proposed DSP packing scheme (Sec. 3.4.4).

Specifically, the 4-bit LPU architectures achieve a throughput improvement of 2.28× for VGG-16 and 2.48× for AlexNet compared to the faithful, zero-error 8-bit architecture[1] and a speed-up of 5.18× for VGG-16 and 5.29× for AlexNet compared to the 16-bit counterpart, which is the most widely used precision by existing FPGA accelerators [289][298].

[1] AlexNet, 4-bit: [DSPs:100%, LUTs:80.6%, BRAM:5.16%]
AlexNet, 8-bit: [DSPs:100%, LUTs:83.1%, BRAM:4.34%]
VGG-16, 4-bit: [DSPs:100%, LUTs:81.4%, BRAM:4.88%]
VGG-16, 8-bit: [DSPs:100%, LUTs:82.9%, BRAM:5.76%]

3.5.3 Throughput-Optimised Cascade Evaluation

To evaluate the performance gains of *CascadeCNN*, the generated two-stage system is compared with a baseline single-stage architecture, for different error tolerance values. *CascadeCNN*'s fine granularity provided by the CEU search space enables it to fully exploit the performance-accuracy trade-off supporting arbitrary classification accuracy. In contrast, the coarser precision-accuracy trade-off forms the only tunable dimension of the single-stage quantisation approach adopted by the baseline. To address this incompatibility in the granularity of design points across the model accuracy dimension, each *CascadeCNN* instance is compared with an optimised baseline HPU employing a precision that achieves the same or better accuracy as the cascade system (ranging from 5 to 8-bit wordlengths). *CascadeCNN*'s HPU adopts the same precision as the baseline, whereas its LPU maintains the 4-bit wordlength yielded by the toolflow. For the cascade architecture, the overall measured processing time for a batch includes LPU's and CEU's latency for the whole batch, HPU's latency for the re-classified samples and the FPGA's reconfiguration time.

The achieved throughput gains are depicted in Fig. 3.13 across a wide range of error thresholds on ZC706 and ZCU102 boards. When zero or extremely small error tolerance (below 0.25 p.p.) is allowed by the user, the CEU adopts a strict evaluation policy that results in an excessively high percentage of rejected samples forwarded to the HPU for re-processing. This introduces significant computational load that results in a slow-down of the overall cascade architecture, reaching up to 70% of the baseline's performance. In such cases, where the baseline architecture outperforms the two-stage design, *CascadeCNN* generates an optimised single-stage architecture, that meets the required error tolerance. For intermediate error levels (0.5-5 p.p.) in both target platforms, the proposed cascade system outperforms the baseline by up to 48% for AlexNet and 55% for VGG-16, for the same resource budget and accuracy. Finally, in the case of high error tolerance (over 5 p.p.), the speed-up becomes less significant as the gap in wordlength between the LPU and the baseline design closes.

A persistent gain on throughput is observed across the two target devices, which comes with the scalable performance obtained by the highly parametrisable architecture. Moreover, although resource-rich devices, such as ZCU102, are burdened by larger reconfiguration time, *CascadeCNN*'s DSE alleviates that cost by employing larger batch sizes, that allow more rare reconfigurations and hence higher amortisation of their cost.

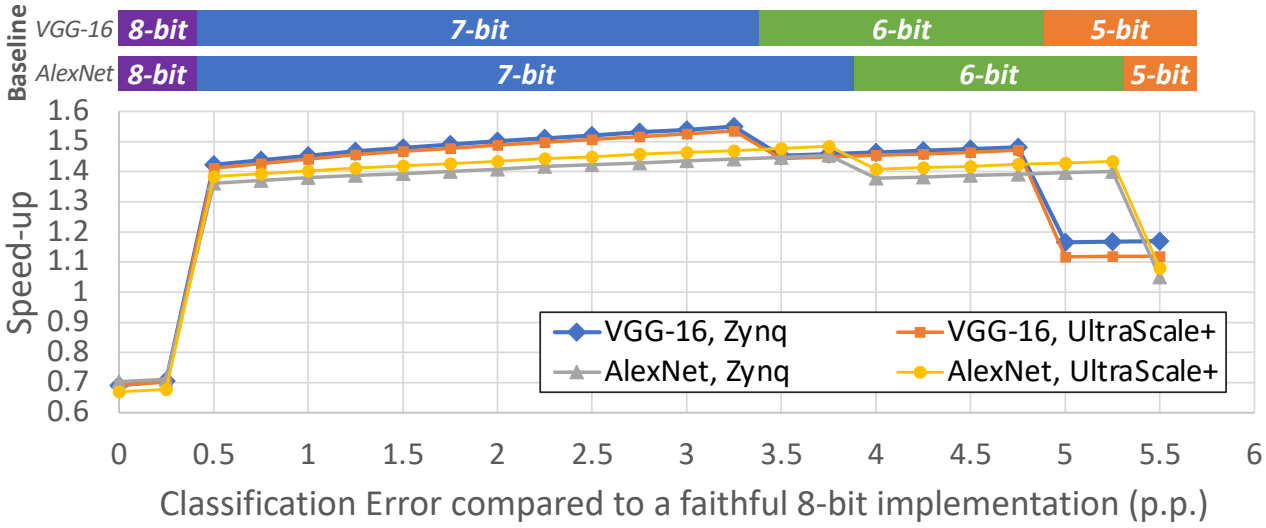


Figure 3.13: *CascadeCNN*'s speed-up as a function of classification error.

The CEU component is executed together with the softmax layer on the CPU, with their aggregate processing time being 4× faster than the FPGA-side computations. With the FPGA and the CPU acting as a processing pipeline across the batches of inputs, the latency of the CPU side is hidden and the system's overall performance is determined by the FPGA computations.

Comparison with Existing FPGA Work

This section explores the performance of the proposed framework with respect to existing FPGA work that does not consider model re-training. This is investigated by comparing with a set of state-of-the-art works, including the highest performing AlexNet and VGG-16 on Zynq 7045 [298][292], on Arria 10 [321][322], the Escher accelerator that optimises memory bandwidth utilisation [323] and an RTL-based automated CNN-to-FPGA toolflow [289]. Table 3.1 presents the measured performance results. The performance column lists the throughput in GOp/s and the performance density in GOp/s/DSP (shown in brackets). For each network, three *CascadeCNN* design points are examined, generated by the toolflow by setting the error tolerance to 1, 3 and 5 p.p..

HPU Performance. To evaluate the architecture of the HPU, each accelerator is compared with the HPU of the same wordlength. In this respect, the 16-bit AlexNet HPU outperforms the existing designs of [298] and [323] in both achieved GOp/s and GOp/s/DSP and reaches 78% of the GOp/s/DSP of the highly-optimised, hand-tuned design of [321]. With respect to VGG-16, the 16-bit HPU outperforms [298], [292] and [289] and reaches 56% of the performance

Table 3.1: Evaluation of Throughput-optimised CascadeCNN

Method	FPGA	Wordlength	Clock	Performance[1]	Speed-up[2]	Err
AlexNet						
HPU	Z-7045	16-bit	131 MHz	313.48 (0.35)	**1.00**	0*p.p.*
[298]	Z-7045	16-bit	125 MHz	161.98 (0.18)	0.51	0*p.p.*
[323]	Virtex-7	16-bit	100 MHz	135.00 (0.06)	0.17	0*p.p.*
[321]	Arria10‡	16-bit	303 MHz	1382.00 (0.45)	1.29	0*p.p.*
	Z-7045	[4,7]bit†	150 MHz	1190.35 (1.32)	3.77	$\leq$ 1*p.p.*
CascadeCNN	Z-7045	[4,7]bit†	150 MHz	1238.66 (1.37)	3.91	$\leq$ 3*p.p.*
	Z-7045	[4,6]bit†	150 MHz	1343.61 (1.49)	4.26	$\leq$ 5*p.p.*
VGG-16						
HPU	Z-7045	16-bit	131 MHz	299.07 (0.33)	**1.00**	0*p.p.*
[298]	Z-7045	16-bit	125 MHz	123.12 (0.14)	0.42	0*p.p.*
[292]	Z-7045	16-bit	150 MHz	136.97 (0.15)	0.45	0*p.p.*
[289]	Arria10‡	16-bit	200 MHz	619.13 (0.23)	0.70	0*p.p.*
[322]	Arria10‡	16-bit	385 MHz	1790.00 (0.59)	1.79	0*p.p.*
HPU	Z-7045	8-bit	150 MHz	680.91 (0.75)	2.27	0*p.p.*
[292]	Z-7045	8-bit	150 MHz	273.76 (0.30)*	0.91	0*p.p.*
	Z-7045	[4,7]bit†	150 MHz	1140.28 (1.26)	3.82	$\leq$ 1*p.p.*
CascadeCNN	Z-7045	[4,7]bit†	150 MHz	1208.69 (1.34)	4.06	$\leq$ 3*p.p.*
	Z-7045	[4,6]bit†	150 MHz	1450.13 (1.61)	4.88	$\leq$ 5*p.p.*

* projected, † denotes [LPU,HPU] precision pair, ‡ refers to GX115 device
1. Expressed in: [GOp/s (GOp/s/DSP)] using 18×18 and 25×18 DSP configurations
2. Normalised performance over the 16-bit HPU of each network (in GOp/s/DSP)

density of the mixed OpenCL-RTL design of [322]. Finally, the 8-bit VGG-16 HPU achieves a speed-up of 2.5× over [292], employing the same precision.

CascadeCNN Performance. The two-stage cascade architecture however, achieves up to 3.31× higher performance density on AlexNet compared to the hand-tuned architecture of [321], with an average of 3.1× (geo. mean) in the error tolerance range of 1 to 5 p.p.. In the same error range, *CascadeCNN* also outperforms the works of [298] and [323] by up to 8.29× and 24.83× respectively. Similarly for VGG-16, the full cascade architecture outperforms the mixed OpenCL-RTL design of [322] by up to 2.73× with an average of 2.36× (geo. mean) in the error tolerance range 1 to 5 p.p., while demonstrating remarkable gains compared to all other examined designs. Overall, in cases that tolerate an extra error of 1 to 5 p.p., *CascadeCNN* demonstrates significant speed-ups with respect to the state-of-the-art existing accelerators.

3.5.4 Latency-aware Cascade Evaluation

This section focuses on the evaluation of the throughput-latency co-optimised flavor of *CascadeCNN*, introduced in Sec. 3.4.6. In this setting, employing an error tolerance of 3.5 percentage points (p.p.), the toolflow again selects 4-bit precision for the weights and computations of the LPU and 7-bit representation for the HPU of both examined models. Alongside, the CEU is configured to satisfy the predetermined accuracy requirements, yielding a sample re-processing ratio $\tilde{p}$ of 46.3% and 36.5% for AlexNet and VGG-16 respectively. Given the above cascaded CNN models, the proposed DSE methodology is used to instantiate optimised cascade hardware architectures for each model, embodying the introduced resource-sharing approach.

The resulting models are again compared a single-stage HPU, relevant accelerators from the literature, as well as throughput-optimised *CascadeCNN* instances. The results of these comparisons are listed in Table 3.2. All the examined methods are implemented on the same FPGA board. For both benchmark models, the achieved clock frequency and the numerical precision of its underlying implementation are reported, along with the measured throughput and latency. For cascaded and excessively quantised models, the accuracy degradation with respect to the base CNN is also reported.

Overall, it can be seen that the proposed methodology is providing significant speed-ups both in terms of throughput and latency, compared to single-stage approaches from the literature, targeting applications that can tolerate a controlled accuracy sacrifice. Furthermore, the co-optimised two-stage methodology introduces a new pareto points in trade-off between throughput and latency in contrast to the throughput-optimised *CascadeCNN* and latency-optimised HPU baseline, being able to achieve similar throughput improvement against the HPU with *CascadeCNN*, with substantially smaller impact on latency, under the same error tolerance. This enables its adoption by a wider span of applications related to real-time systems, such as Augmented Reality (AR), wearable devices, mobile robots and Unmanned Aerial Vehicles, demanding the consumption of high frame-rate inputs while imposing stringent latency constraints.

In more detail, for AlexNet [20], the proposed throughput-latency co-optimised cascade reaches 87% of the throughput-optimised *CascadeCNN* implementation's throughput, demonstrating however 677× less latency (on average) under the same error tolerance of 3.5 p.p.. Compared to the latency-optimised single-stage HPU baseline built based on the proposed hardware architecture

Table 3.2: Evaluation of Latency-aware CascadeCNN (Z-7045)

Method	Wordlength	Clock	Latency	Throughput	Error
AlexNet [20]					
[298]	16-bit	125 MHz	8.22 ms	161.98 GOp/s	0*p.p.*
[316]	16-bit	100 MHz	12.30 ms	108.25 GOp/s	0*p.p.*
[317]	16-bit	250 MHz	9.95 ms	120.30 GOp/s	0*p.p.*
HPU	16-bit	131 MHz	7.1 ms	304.04 GOp/s	0*p.p.*
HPU	7-bit	150 MHz	2.9 ms	747.46 GOp/s	≤3.5p.p.
CascadeCNN-T	[4,7]bit†	150 MHz	3793.9 ms	1310.7 GOp/s	≤3.5p.p.
CascadeCNN-L	[4,7]bit†	150 MHz	5.6 ms	1139.9 GOp/s	≤3.5p.p.
VGG-16 [36]					
[298]	16-bit	125 MHz	249.50 ms	123.12 GOp/s	0*p.p.*
[285]	16-bit	150 MHz	163.42 ms	187.80 GOp/s	0*p.p.*
HPU	16-bit	131 MHz	104.4 ms	293.9 GOp/s	0*p.p.*
HPU	7-bit	150 MHz	40.8 ms	751.9 GOp/s	≤ 3.5*p.p.*
CascadeCNN-T	[4,7]bit†	150 MHz	25991.5 ms	1222.9 GOp/s	≤ 3.5*p.p.*
CascadeCNN-L	[4,7]bit†	150 MHz	76.1 ms	1213.1 GOp/s	≤ 3.5*p.p.*

† Denotes precision for [LPU,HPU] cascade pair:
CascadeCNN-T: denotes throughput-optimised design of Sec. 3.4.5.
CascadeCNN-L: denotes throughput-latency co-optimised design of Sec. 3.4.6.

under the same error tolerance, the proposed approach achieves a 1.52× speed-up in throughput, being able to maintain real-time response latency (5.6ms).

Similarly, in the case of VGG-16 [36], the proposed approach sustains up to 99% of the throughput-optimised *CascadeCNN* operation rate, while demonstrating 341× less response time on average compared to *CascadeCNN*. This translates to a performance boost of 61% in throughput compared to the HPU single-stage approach, at the expense of a much smaller latency overhead than *CascadeCNN*.

Additionally, the comparison with 16-bit works from the literature indicates that there exists significant space for adoption of input-dependent computation methodologies, such as this work, for applications that can tolerate a controlled accuracy degradation (e.g. ≤3.5p.p for the examined instance), to gain substantial performance improvement both in throughput and latency. Indicatively, the proposed approach provides gains of 3.74×-10.5× in throughput and 1.46×-2.2× in avg. latency, compared to its 16-bit single-stage counterparts for AlexNet; and 6.45×-9.85× in throughput and 2.14×-3.27× in avg. latency for VGG-16 accordingly.

The proposed resource-sharing approach sacrifices part of the LPU speed-up to alleviate the need for batching and reconfiguration time. In this manner, misclassified samples can

Table 3.3: Analytical Performance Model Accuracy

	Performance Model	Error: AlexNet	Error: VGG-16
(i)	Individual LPU/HPU	6.8%	6.8%
(ii)	Combined - *avg-case* Mem.	9.2%	9.7%
(iii)	Combined - *worst-case* Mem.	8.4%	-

be forwarded directly to the concurrent HPU unit (Fig. 3.11c), being prone to significantly reduced latency overhead (Eq. 3.22).

3.5.5 Performance Modelling Accuracy

In this section, the accuracy of the developed performance models is evaluated across multiple points of the design space, by comparing the predicted and the measured performance of the hardware architecture. This is in order to capture the confidence in generating the best designs from the DSE. The results are summarised in Table 3.3.

In the case of the precision-aware roofline model focusing on each unit independently (Sec. 3.4.6a, b), an error of 6.8% (geo. mean) is revealed (row (i)). I/O delay variations and software overhead are considered as driving factors of the model error.

Considering the whole cascade system (Sec. 3.4.6c-e), when adopting the *avg-case* memory model of Eq. 3.21 the error (row (ii)) expands up to 9.7% (geo. mean), accounted to the non-uniform distribution of LPU missclassifications across the inputs, and the unpredictable off-chip memory read/write transaction scheduling, when the available bandwidth is surpassed. When the worst-case memory model (Eq. 3.20) is used, the model error is reduced to 8.4% in the case of AlexNet (row (iii)), due to the stringent barrier established on the memory bandwidth requirements of both compute units. However, this model fails to provide any valid combination of design points for VGG-16, conceivably due to its challenging (and unevenly distributed between layers) memory requirements [117].

Resource-wise the introduced modelling was able to accurately capture the DSP and on-chip memory requirements of the proposed architecture. In terms of LUTs, withholding in-advance 30% of the device's available LUT-count to be used for routing purposes was proven sufficient across all examined designs.

3.5.6 Qualitative Analysis on Scene Recognition

Finally, this section presents a qualitative analysis of the proposed approach, concentrating on the CEU as well as failure cases and the underlying limitations. For this purpose the analysis focuses on the downstream task of scene recognition [324]. Visual place understanding has been one of the early applications of deep learning in robotics, equipping mobile agents with the ability to identify the semantics of their operating space (e.g. kitchen, office-area etc.) and plan their actions accordingly [76].

In this context, a VGG-16 model is trained on the Places365 dataset [325], and fed to the proposed toolflow along with a subset of 200 samples from Places365 Validation set, to provide an efficient cascade counterpart system. The baseline model achieved 84.9% top-5 accuracy, while the [5,8]-bit cascade yielded under a 3.5 p.p. top-5 error tolerance demonstrated a speed-up of 1.56× compared to an 8-bit baseline within the same error range, consistently with the observations from the previous sections. Selected examples from the Places365 dataset are illustrated in Fig. 3.14 along with the corresponding LPU and HPU predictions. Alongside, confidence scores are listed for both units[2], as provided by the proposed CEU as well as the widely adopted Top1 metric [174], concentrating on the maximum value of the softmax output.

Initially (row (i)) demonstrates an easy example that was classified correctly by both units. The proposed CEU metric (gBvSB), being crafted to act as a Cascade exit-policy rather than a generic uncertainty estimation, demonstrates better effectiveness on reflecting the correctness of the prediction at hand. The same applies on more challenging inputs, as in the case of (row (ii)), where the proposed model is still able to yield a correct prediction (in terms of the examined top-5 accuracy metric).

Notably, due to quantisation, the values of the final layer's logits of the LPU are affected, and so do the softmax probabilities displayed on the Figure. As such, even on easy cases, quantisation (at the examined level) does not leave the probability distribution unaffected. Furthermore, in some cases the excessive quantisation of the LPU has triggered an error to its prediction, that is usually reflected by an almost uniform distribution across all classes, presumably caused by truncation of some important activation values. This case is easily detectable by the CEU, that correctly triggers a re-computation of this sample from a higher precision unit, to restore accuracy (row (iii)).

[2]This contrasts the actual deployment of the proposed Cascade, where only LPU predictions are evaluated by the CEU.



Figure 3.14: Qualitative analysis illustrating LPU and HPU predictions on indicative samples of Places365, along with their confidence evaluation scores (Top1: highest softmax score (pass if >0.5), gBvSB: proposed metrics (pass if >0.6)).

The impact of quantisation, however, is not always so profound. Cases exists, where the loss of detailed information incurred by quantisation was enough to perturbate the class distribution in some predictions, while passing unnoticed from the CEU (row (iv)). Such false-positive confidence evaluations trigger unrecoverable errors, being mainly responsible for the controlled accuracy degradation evident on the operational sweet-spot of CascadeCNN models.

Very rarely, the LPU was able to yield a correct prediction for samples that were later missclassified by the HPU. In all found cases, this has been due to a coincidental shuffling between classes of equally low-confidence (row (v)), where the exit-policy either way propagated the sample to the HPU. Notably, due to the limited quantisation levels available under the excessive LPU quantisation scheme, it is not uncommon to have multiple classes with exactly the same probability on the LPU's output.

Finally, a notable portion of genuinely hard or ambiguous samples exists in most classification datasets to date. Such samples are often missclassified by both compute units (row (vi)). However, low confidence is usually demonstrated by the LPU's in such cases, and thus a re-computation is triggered by the CEU, incurring additional computational cost, without improving the predictive accuracy of the overall system. This fact is responsible for a large proportion of CascadeCNN's overhead, and forms the main differentiating factor between the desirable properties of uncertainty estimation metrics and exit-policy approaches, that can motivate further investigation by the community, as discussed later, in the conclusions of this thesis (Ch.8).

3.6 Discussion

This chapter focused on applying approximations on the computation level of the deep learning deployment stack. In this context, precision quantisation was employed to approximate the computation for all layers of a CNN, coupled with a highly-optimised hardware architecture that enabled the realisation of performance gains arising from the reduction in precision. The proposed methodology does not rely on any fine-tuning of the model or access to training data.

Aiming to push the limits of precision-quantised CNN inference, this chapter adopted the input-dependent inference paradigm, introducing a novel cascade of multi-precision CNN classifier. This comprised an excessively low-precision unit (LPU), followed by a confidence evaluation unit that distinguished between samples for which the LPU was able to yield a confident

prediction and those that need to be further processed by a subsequent computationally heavier high-precision unit (HPU).

A custom hardware architecture with the ability to scale its performance as a function of the wordlength was also introduced along with an analytical performance model that enabled fast optimisation of the architectural configuration of each unit to the CNN-FPGA pair at hand through design space exploration. For throughput-optimised use-cases, device reconfiguration was employed between the two stages of the cascade, with the reconfiguration overhead being alleviated by large batch sizes at the expense of higher inference latency. A throughput-latency co-optimised variation of the proposed methodology replaced device reconfiguration with a resource sharing approach for usecases that cannot tolerate the inflated latency cost. In this setting the processing units are instantiated alongside on the same device, drastically reducing inference latency at the expense of slightly lower inference efficiency and a less deterministic computation/scheduling pattern.

The whole process was wrapped as an automated toolflow, termed *CascadeCNN* that can generate a two-stage model cascade along with its hardware implementation, in view of a target CNN-FPGA pair, user-specified requirements on the speed-accuracy characteristics of the system and a small calibration set for the target application.

The proposed approach was able to demonstrate significant performance gains across CNN models and devices for applications that can tolerate a controlled accuracy degradation, by offering new pareto points on the speed-accuracy trade-off of precision-quantised DNN inference.

An extension of the proposed methodology to N-stage cascades (where N>2) is possible if justified by the underlying precision-accuracy and precision-performance trade-offs. Although in the examined use-cases the additional re-processing overhead that a 3-stage cascade would introduce was not justified by the attainable performance results, the picture is likely to change in a scenario where training data are available to enable for quantisation-aware training of the LPU, pushing its precision to the extreme low end (1 or 2 bits). This investigation is left as a potential direction for future work.



Image sourced from BDD Dataset (modified).

4

Approximate LSTMs
for Time-Constrained/Progressive Inference

Contents

4.1 Overview . . . 96
4.2 Motivation . . . 96
4.3 Related Work . . . 99
4.3.1 Efficient LSTM Deployment . . . 99
4.3.2 Progressive Deep Learning Inference . . . 103
4.4 Methodology . . . 103
4.4.1 Overview . . . 103
4.4.2 LSTM Workload Analysis . . . 104
4.4.3 LSTM Approximations . . . 106
4.4.4 Hardware Architecture . . . 110
4.4.5 Design Space Exploration . . . 114
4.4.6 Model-Hardware Co-design . . . 115
4.5 Evaluation: A Case study on Autonomous Driving . . . 116
4.5.1 Autonomous Driving Background . . . 116
4.5.2 Experimental Setup . . . 117
4.5.3 Architectural Performance . . . 119
4.5.4 End-to-end Evaluation . . . 120
4.5.5 Qualitative Analysis . . . 125
4.6 Discussion . . . 126

Keywords: *LSTMs, Pruning, Low-rank decomposition, Autonomous Driving, Progressive Inference, FPGA architecture, Design Space Exploration, Model-Hardware co-design*

4.1 Overview

Although approximating the computation of CNNs through quantisation demonstrated great results under a controlled accuracy degradation, not all deep learning models exhibit the same level of resilience to low-precision representations. Long Short-Term Memory (LSTM) networks, for example, that have arisen as a prominent AI model in many emerging applications with the need to recognise long-term dependencies in sequential data such as video streams, are shown to be significantly more sensitive to quantisation. At the same time, LSTMs exhibit complex dependencies and minimum re-use of parameters, leading to heavy memory requirements. This introduces unique challenges in their deployment on latency-critical systems, such as mobile robots and self-driving cars, that are equipped with limited computational resources on-board.

To address these challenges, this chapter focuses on approximations that can be applied on the ***model layer*** *of the deep learning deployment stack, aiming to loosen the memory demands of LSTM models through post-training pruning and low-rank approximation of the model parameters. At the same time, approximations on the* ***computation layer*** *of the stack are also applied, by means of an iterative refinement scheme that re-structures the forward-pass computations of LSTMs to support the paradigm of progressive inference. This is tightly coupled with a custom hardware accelerator that is co-designed to efficiently map the resulting approximate LSTM model. More specifically, the proposed scheme combines model pruning and computation restructuring (in the form of iterative low-rank compression), to obtain the best possible approximation of the inference result under a pre-specified latency budget for the target application. Algorithmic and hardware parameters are co-optimised to enable latency-critical systems to make informed decisions even in early stages of the computation, meeting application-specific requirements that affect safety and robustness.*

This chapter is based on two published papers (one journal and one conference) both co-authored with Michalis Rizakis, Dr. Stylianos I. Venieris and Prof. Christos-Savvas Bouganis: (ARC 2018), (CEM 2020).

4.2 Motivation

Recurrent neural networks (RNNs) form a family of machine learning models with the ability to recognise patterns in sequential and temporal data. In the past decade, long short-term

memory (LSTM) networks [156] have emerged as the dominant RNN by setting the state-of-the-art in various AI tasks, such as machine translation [326] and video understanding [327]. Among various LSTM-enabled applications, LSTMs have been successfully deployed on mobile robots operating in complex environments for visual tasks including scene labelling [328], human trajectory prediction [329] and ground classification [327]. In this setting, AI agents rely on LSTM-based mechanisms for perceiving their surroundings and (re-)acting [330] under stringent latency constraints. In such scenarios, making the most informed decision under a limited time budget is of vital importance in order to ensure the robust, safe and successful operation of the system within complex and uncertain environments [331].

A representative example is a driverless car navigating autonomously in an urban environment under the control of an LSTM that predicts the desired throttle/brake position and steering angle based on the input video sequence. With human driver reaction time in the order of milliseconds (varying with situation and individual person) [332], autonomous driving systems target a relevant low-latency envelope to take action from the moment an event occurs on the road, in order to preserve the ability of achieving comparable reliability with humans. In this respect, extracting the best possible approximation of the desired action to be commanded within the real-time latency constraints is preferred from a more accurate decision later in time.

From a technical viewpoint, performing the most informed action under a time budget reduces to the problem of obtaining the highest quality output from an LSTM given a constraint in computation time. Current methods for deploying LSTMs follow the behaviour depicted in Fig. 4.1. Conventional implementations [333][334][335] require the whole inference computation to finish in order to obtain meaningful information from the LSTM and thus prolong the sensing-to-action loop. Instead, the stringent latency deadlines of real-life systems call for *progressive inference* designs that can provide the best possible estimate of the final output for a given time budget and improve on it as more time budget becomes available (Fig. 4.1). This property would enable the agent to exploit the maximum possible amount of information that is available in the current input and effectively optimise its overall operation.

From a workload perspective, LSTMs are challenging by being parameter-heavy and memory-bounded. This property means that their entire set of weights typically don't fit at-once in the on-chip memory of most accelerators in the embedded space, and thus, have to be interchangeably loaded on-the-fly, in smaller chunks. Additionally, in contrast to convolutional layers and in

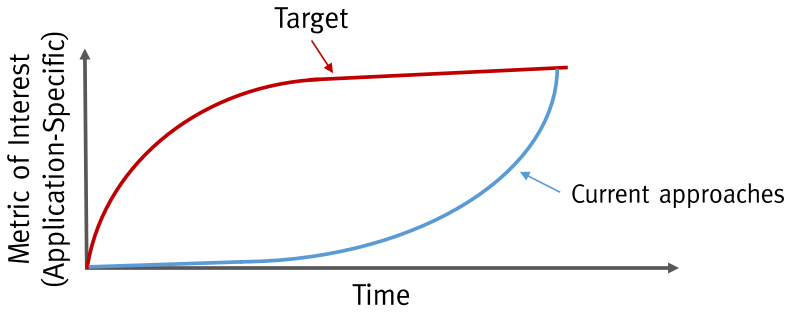


Figure 4.1: Behaviour of AI systems under the conventional and progressive inference paradigms.

line with fully-connected ones, the limited re-use of each weight during inference renders the performance of faithful LSTM implementations to be usually limited by the available memory bandwidth of the platform, rather than by its available computational power. Furthermore, the excessive memory accesses and the inefficient use of computational resources when executing LSTMs on conventional platforms leads to substantial power inefficiencies which are critical for battery-operated systems. To attack this issue, recent works deviated from general-purpose computing platforms and adopted a *model-hardware co-design* approach for the generation of custom hardware architectures [336]. Enabled by the customisation and flexibility of FPGAs, several works propose different approximation techniques, focusing on model compression [287], quantisation [337][338] and pruning [335][337], together with an associated FPGA-based hardware accelerator, tailored to the computational needs of the model and its approximate computing scheme, to match the computational demands of LSTMs.

Despite the effectiveness of these methods, their application requires a *retraining step*, which allows the refinement of the model in order to compensate for any approximation losses in the model's accuracy. For the retraining step to be feasible, availability of the training set is required, which is not a realistic assumption in privacy-aware applications [339], as in the case of large-scale datasets collected by commercial companies that remain proprietary, making privacy-preserving AI techniques increasingly relevant [232][293].

In this chapter, a novel approximate computing scheme along with a custom hardware architecture for LSTMs are introduced as an end-to-end framework to address the problem of high-performance LSTM deployment in time-constrained settings.

The main contributions of this chapter can be summarised as:

- An iterative approximation method that applies simultaneously low-rank (Singular Value Decomposition (SVD)-based) compression along with hardware-friendly (structured) pruning on a pre-trained LSTM model. While the number of refinement iterations and sparsity level remain tunable, this iterative process enables the proposed framework to (i) exploit the resilience of the target application to approximations, (ii) explore the trade-off between computational and memory load and application-level accuracy and (iii) execute the LSTM under a time constraint with increasing accuracy as a function of computation time budget.
- A novel FPGA-based architecture that exploits the inherent parallelism of the LSTM, parametrised with respect to the level of compression and pruning to scale its performance.
- A model-hardware co-optimisation framework, that simultaneously configures the parameters of the hardware architecture and approximation method, in view of a target application, to generate a system tailored to the target LSTM model, available FPGA resources and the computation time constraints and accuracy requirements of the underlying application, without the need of any re-training.

As a result, this chapter showcases a significantly improved speed-accuracy trade-off presented by the proposed progressive inference scheme, compared to a baseline implementation of the same model, while both designs are exploiting the customisation capabilities of an FPGA. This is achieved by the proposed approach effectively reducing the computational workload of a given LSTM model, aiming to meet the desired quality of result. The experimental evaluation of the proposed approach is conducted on a state-of-the-art driving model for autonomous vehicles, as self-driving cars form a representative example of a system with tight computation time budget to make mission critical decisions, while being also constrained in a limited computational resource environment.

4.3 Related Work

4.3.1 Efficient LSTM Deployment

With a focus on LSTM workloads, several works proposed optimisations for executing LSTMs on conventional programmable platforms such as CPUs [340] and GPUs [341][342][343]. By employing tailor-made caching and data-locality strategies, this line of work has demonstrated significant performance gains and has approached the performance limits of commodity programmable

hardware architectures. To push further the attainable performance of LSTMs, FPGAs can be exploited to propose custom accelerator designs. However, the majority of existing work on hardware accelerators has focused on compute-intensive convolutional neural networks (CNNs). With memory-bounded recurrent models having substantially different computational patterns and deployment challenges, the CNN-centric methods and accelerator designs either provide minimal gains or are not directly applicable to LSTMs [343][344].

Recently, the effectiveness of RNNs in sequence modelling tasks has attracted some attention leading to dedicated hardware accelerators for recurrent models. Li et al. [345] proposed an FPGA-based accelerator for the training of an RNN language model. In [346], the authors focus on the optimised deployment of the Gated Recurrent Unit (GRU) model [347] in data centres with server-grade FPGAs, ASICs, GPUs and CPUs and propose an algorithmic memoisation-based method to reduce the computational load at the expense of increased memory footprint. Zhang et al. [335] presented an FPGA-based accelerator for a Long-Term Recurrent Convolutional Network (LRCN) for video footage description that consists of a CNN followed by an LSTM. Their design focuses on balancing the resource allocation between the layers of the LRCN and pruning the fully-connected and LSTM layers to minimise the off-chip memory accesses. Finally, the authors of [334] present an empirical study of the effect of different architectural designs on the computational resources, on-chip memory capacity and off-chip memory bandwidth requirements of an LSTM model.

Specifically for LSTMs, based on the stage where optimisations are applied, system-based approaches can be categorised into: (i) training-stage methods, (ii) post-training with fine-tuning, and (iii) run-time methods.

Training-stage methods

By modifying the model design process, Wang *et al.* [337] proposed a compression technique that modifies the LSTM model before the training stage. By applying a circulant structure to the matrices within each LSTM gate, this approach allows the same weights to be shared across several neurons and substantially reduces model size and storage requirements. Extending this technique, the E-RNN [348] framework introduces a blocking version of circulant matrices and treats the block size as a tunable parameter to balance processing speed and accuracy. The block-circulant matrix operations were executed in the frequency domain to leverage the computational efficiency of FFT. At the hardware level, to bypass the limitations of conventional platforms when executing

irregular computations, E-RNN proposed a highly-parametrised custom hardware architecture mapped on the flexible FPGA fabrics, leading to a 7.7× speed-up over ESE [287].

In contrast with the post-training approach introduced in this chapter, both of these methods intervene substantially with the LSTM model design and training process. Nevertheless, since the SVD-based decomposition is applicable to circulant matrices, the proposed inference scheme is orthogonal to these works and can be applied in a complementary manner to yield further performance improvements.

Post-training methods with fine-tuning

By putting emphasis on minimising the effect of memory-boundness of LSTM workloads, ESE [287] proposes to sparsify LSTMs via a pruning scheme and map it on an FPGA-based accelerator tailored for sparse workloads. Given a pre-trained model, its weights are pruned in an iterative manner using a load-balance-aware strategy that aims to sustain the utilisation of the accelerator high. Furthermore, to avoid excessive accuracy drop, at each iteration the unpruned weights are fine-tuned using the training set. To overcome the inefficiencies of CPUs and GPUs when executing the sparse, pruned model, ESE exploits the customisability of FPGAs to propose an accelerator optimised for sparse computations. As a result, the load-balance-aware pruning leads to 6.2× faster execution over dense LSTMs on ESE's accelerator. To further improve the load balancing, Park *et al.* [349] proposed an alternative encoding format for storing sparse matrices and managed to achieve higher sustained utilisation of the PEs on the same accelerator.

Overall, despite the fact that the pruning method used by both ESE and Part *et al.* is applied *post-training* on pre-trained LSTMs, access to the training set is required in order to iteratively prune and fine-tune the model's weights and thus not significantly degrade the accuracy. In contrast, the method proposed in this chapter is also applied post-training on pre-trained models, but it does not require access to the dataset and hence is suitable for privacy-aware cases.

Run-time methods

This class of methods exploits techniques to dynamically skip unnecessary computations during the execution of an LSTM. In this context, DeltaRNN [303] employs a strategy to dynamically avoid computations based on the estimated impact on the output of the network. The skipping criterion is based on the degree of change of each input activation. To effectively implement

this technique without significantly dropping the accuracy, the target LSTM has to be trained using the Delta Network scheme [350]. From a hardware perspective, to overcome the inefficiency of GPUs due to the conditional execution strategy, the DeltaRNN-trained LSTM is mapped on a custom accelerator design which exploits the reconfigurability of FPGAs to efficiently perform the dynamic computations. Nevertheless, despite the run-time computation-skipping, DeltaRNN requires the target model to be trained using the Delta Network algorithm and hence is limited to settings where the training set is available, while requiring substantial modification of the training scheme and tuning of the hyperparameters. In contrast to this, our method avoids the time overhead and engineering effort of training and parameter tuning and can be directly applied to pre-trained LSTMs.

A similar approach is proposed in [343] tailored to mobile-grade GPU architectures. This work employs dynamic row skipping by predicting near-zero results at run time and omitting all preceding computations. Compared to DeltaRNN, this approach does not require modification of the training scheme and hence can be applied directly on pre-trained models post-training. The implementation is optimised to exploit the characteristics of mobile GPUs.

Among the existing designs, reduced arithmetic precision schemes have also been used to obtain gains in terms of performance and power efficiency. ESE [287] and E-RNN [348] employ 12-bit fixed-point precision for both weights and activations. However, due to the much lower resilience of LSTMs to quantisation and in order to avoid the severe degradation of accuracy due to the limited numerical precision, a fine-tuning step is required by means of additional training iterations. Alternatively, DeltaRNN [303] avoids fine-tuning and employs a 16-bit fixed-point representation. Nonetheless, DeltaRNN's quantisation is not network-agnostic, but hand-tuned to minimise the accuracy losses of the target network. In the context of this chapter, single-precision floating-point format is used to avoid the need for fine-tuning and limit the sources of quality-of-result degradation to the proposed approximate computing techniques. Nevertheless, the proposed methodology is orthogonal and independent of employed numerical representation and thus can be combined with fixed-point arithmetic representation to further boost both performance and power efficiency at the attainable level.

4.3.2 Progressive Deep Learning Inference

Closer to the progressive inference philosophy of the proposed approach lie CNNs that employ early-exit classifiers. CNNs with early exits [175][174][304] provide a run-time accuracy-latency trade-off and are able to produce an increasingly refined output as a function of time, which casts them suitable for time-constrained inference scenarios. However, as the early-exit classifiers have to be trained, access to the training set is necessary and complex hyperparameter tuning is required [174][304]. Furthermore, although early exiting has been applied to CNN-based classifiers with promising results, this mechanism is not directly applicable to the substantially different topology of LSTMs. Alternatively, the proposed method allows for progressive inference using LSTM models without the need to access the training set and the development time overhead for tuning the associated hyperparameters.

4.4 Methodology

4.4.1 Overview

This chapter approaches the challenge of efficient LSTM deployment from a progressive inference angle, through an automated toolflow that provides a holistic and co-designed software-hardware solution, tailored to a user-specified LSTM-FPGA pair, and application-level requirements (in accuracy, latency etc.), without the need to access the training data of the original model. A high-level overview of the proposed toolflow is illustrated in Fig. 4.2.

More formally, *given*:

- The weight matrices of a trained LSTM model.
- The available computational and memory resources of the target FPGA platform (LUTs, DSPs, Memory bandwidth, on-chip and off-chip memory capacity)
- An application-level error tolerance in a user-defined metric, or an inference latency constraint.
- A limited set of labelled data forming a calibration/evaluation set for the target application.

The proposed framework jointly searches the model approximation and architectural design spaces, to *provide*:

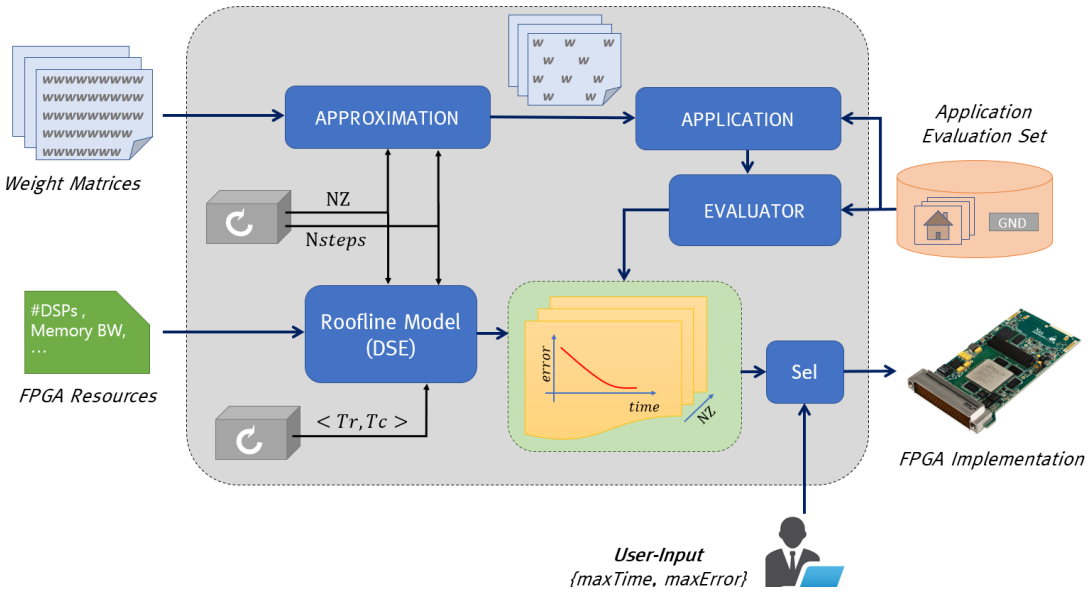


Figure 4.2: Design flow of the proposed Approximate LSTM framework.

- An appropriate algorithmic approximation scheme for the provided LSTM, applying low-rank compression and pruning to tunably reduce the workload of the model.
- A custom hardware accelerator design, tailored to the target LSTM-FPGA pair and adopted approximation scheme, allowing the efficient execution and progressive refinement of the inference process.

In this manner, the trade-off between computation time and application-level error can explored for different approximation schemes, while effectively co-designing the model and hardware. The design point to be implemented on the device is selected based on user-specified requirements for the maximum computation time or application-level error tolerance.

4.4.2 LSTM Workload Analysis

LSTMs are specialised RNNs with enhancements that enable learning long-term dependencies. The key component of an LSTM is a set of units named *gates* which control its behaviour at run time. Fig. 4.3 depicts the structure of an LSTM. The core element of LSTMs is the cell state, shown along the horizontal line at the top of the diagram. At each time step t, the LSTM removes or adds information to the cell state $\mathbf{c}^{(t)}$ via its gate modules $\mathbf{g}^{(t)}$.

Computationally, a gate receives the new input sample $\mathbf{x}^{(t)}$ and the previous output $\mathbf{h}^{(t-1)}$ as inputs and performs matrix-vector multiplications with the weight matrices $\mathbf{W}_{\mathrm{x}}^{\mathbf{g}}$ and $\mathbf{W}_{\mathrm{h}}^{\mathbf{g}}$

respectively. Next, the resulting vectors are accumulated and passed through a nonlinear function, such as a sigmoid $\sigma(\cdot)$, to form $\mathbf{g}^{(t)}$:

$$\mathbf{g}^{(t)} = \sigma(\mathbf{W}_{\text{x}}^{\mathbf{g}} \cdot \mathbf{x}^{(t)} + \mathbf{W}_{\text{h}}^{\mathbf{g}} \cdot \mathbf{h}^{(t-1)}) \tag{4.1}$$

The elements of the weight matrices are learned during the training stage of the target application and remain fixed throughout the inference stage that takes place upon deployment. The nonlinear function operates in an element-by-element fashion and outputs a vector with values between 0 and 1, capturing how much of each element should be kept. A value of 0 represents total forgetting of information, 1 represents total propagation and intermediate values dictate what fraction of the information should be kept. In this manner, by multiplying the output of the nonlinear function with another vector $\mathbf{y}^{(t-1)}$ in an element-by-element fashion, the resulting vector $\mathbf{y}^{(t)}$ comprises a filtered version of its previous state: $\mathbf{y}^{(t)} = \mathbf{y}^{(t-1)} \odot \mathbf{g}^{(t)}$ where $\odot$ denotes the element-wise multiplication between two vectors defined as $(\mathbf{a} \odot \mathbf{b})_i = \mathbf{a}_i \cdot \mathbf{b}_i$.

An LSTM consists of four gates[1]. Starting from the left of the diagram in Fig. 4.3, the *forget* gate $\mathbf{f}^{(t)}$ determines the amount of information that will be forgotten from the previous cell state $\mathbf{c}^{(t-1)}$. Next, the *input* gate $\mathbf{i}^{(t)}$ and the *cell* gate determine how much of the new information at the input will be stored in the new cell state $\mathbf{c}^{(t)}$. The cell gate employs $\tanh(\cdot)$ for its nonlinear function and creates a vector of new candidate values for the new cell state, while the input gate controls which values of the current cell state will be updated. At this point, the new cell state $\mathbf{c}^{(t)}$ has been formed as:

$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \odot \tanh(\mathbf{W}_{\text{x}}^{\mathbf{c}} \cdot \mathbf{x}^{(t)} + \mathbf{W}_{\text{h}}^{\mathbf{c}} \cdot \mathbf{h}^{(t-1)}) \tag{4.2}$$

The final step involves the calculation of the new output vector $\mathbf{h}^{(t)}$, which is a filtered version of the cell state. This is generated by passing the cell state through a $\tanh(\cdot)$ nonlinearity[2] and multiplying the result with the output of the *output* gate $\mathbf{o}^{(t)}$ in order to update only parts of the cell state:

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \odot \tanh(\mathbf{c}^{(t)}) \tag{4.3}$$

[1] by loosely defining cell state as a gate.
[2] Several variants of this configuration appear in the literature.

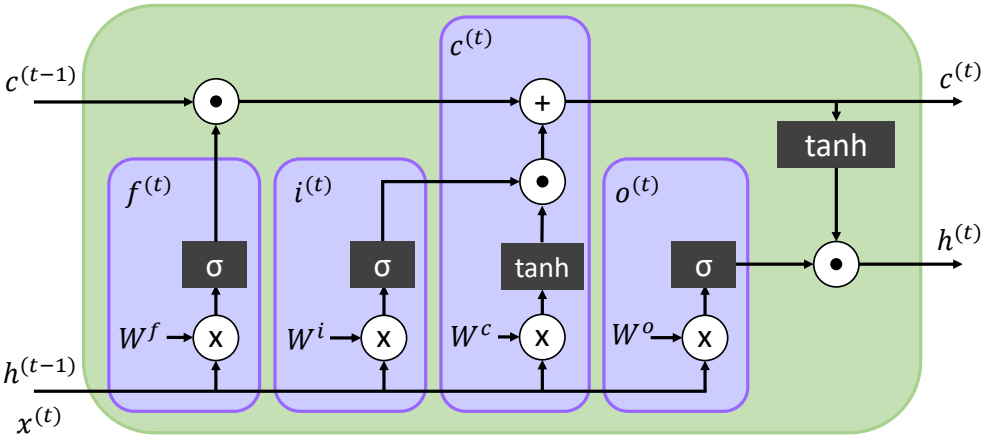


Figure 4.3: Computational Structure of an LSTM model.

4.4.3 LSTM Approximations

As discussed, at the core of an LSTM's workload lie the linear algebra operation of matrix-vector multiplication (Eq. 4.1), which takes place in each of the four gates. Neural networks have been extensively studied to have redundancy in terms of their trained parameters [351]. This property permits the restructuring of the computations of LSTM gates in a manner that allows to extract the maximum information at any time instant. In this respect, this chapter proposes an approximate computing scheme that enables the tuning of the quality of result (QoR) in exchange for an increase in performance. The proposed approach exploits the statistical redundancy of LSTMs by acting at two levels: (i) approximating weight matrices with a low-rank Singular-Value Decomposition (SVD) and (ii) pruning the network by sparsifying the weight matrices based on an importance criterion of their elements. These techniques enable the restructuring of the computations of an LSTM in order to design a computing system that performs the most information-carrying computations first, in order to obtain the peak QoR given a time budget.

Information-maximising low-rank approximation

Eq. 4.1 suggests that each LSTM gate consists of two weight matrices in tune with the current input and previous output respectively. The proposed computation scheme, first concatenates the two weight matrices and input vectors to form a single augmented matrix and vector respectively for each gate as:

$$\mathbf{W^g} = [\mathbf{W^g_x}, \mathbf{W^g_h}] \in \mathbb{R}^{R \times C} \tag{4.4}$$

$$\hat{\mathbf{x}}^{(t)} = \left[\mathbf{x}^{(t)\top}, \mathbf{h}^{(t-1)\top}\right]^{\top} \in \mathbb{R}^{C\times 1}. \tag{4.5}$$

Hence the computation of each gate is refined as:

$$\mathbf{g}^{(t)} = \sigma(\mathbf{W}^{\mathbf{g}} \cdot \hat{\mathbf{x}}^{(t)}) \tag{4.6}$$

This transformation, although equivalent to the one of Eq. 4.1, significantly simplifies the computation and provides larger parallelisation potential.

As a next step, the augmented weight matrix $\mathbf{W}^{\mathbf{g}}$ for each gate is substituted with a low-rank approximation that reduces the computation and memory footprint cost while minimising the information loss. These properties are satisfied by the rank-1 approximation of each weight matrix based on the Singular Value Decomposition (SVD). This approach allows to *approximate the weight matrix* as the outer product of two vectors (the singular vectors) followed by an element-wise multiplication with a constant number (the singular value):

$$\widetilde{\mathbf{W}}^{\mathbf{g}} = \sigma_1^{\mathbf{g}} \cdot \mathbf{u}_1^{\mathbf{g}} \cdot \mathbf{v}_1^{\mathbf{g}\top} \tag{4.7}$$

This calculation is conducted offline, once, considering the weight matrices of a pre-trained LSTM model. Therefore, to obtain the rank-1 approximation a full SVD decomposition for each of the four augmented matrices is independently performed as:

$$\boldsymbol{W}^{\mathbf{g}} = \boldsymbol{U}^{\mathbf{g}} \cdot \boldsymbol{\Sigma}^{\mathbf{g}} \cdot \boldsymbol{V}^{\mathbf{g}\top}, \ \forall \mathbf{g} \in \{\mathbf{f}^{(t)}, \mathbf{i}^{(t)}, \mathbf{c}^{(t)}, \mathbf{o}^{(t)}\} \tag{4.8}$$

where $\boldsymbol{U}^{\mathbf{g}} \in \mathbb{R}^{R\times R}$, $\boldsymbol{\Sigma}^{\mathbf{g}} \in \mathbb{R}^{R\times C}$ and $\boldsymbol{V}^{\mathbf{g}} \in \mathbb{R}^{C\times C}$ and keep the singular vectors ($\mathbf{u}_1^{\mathbf{g}}$, $\mathbf{v}_1^{\mathbf{g}}$) that correspond to the largest singular value ($\sigma_1^{\mathbf{g}}$).

With respect to computational cost at inference time, the original matrix vector multiplication $\widetilde{\mathbf{W}}_i \cdot \tilde{\mathbf{x}}^{(t)}$ is replaced by a dot product followed by an element-wise multiplication between a vector and a constant number, *i.e.* $\sigma_1^i \cdot \mathbf{u}_1^i \cdot (\mathbf{v}_1^{i\top} \cdot \tilde{\mathbf{x}}^{(t)})$, leading to a significant reduction on both the number of operations and the memory footprint of the weight matrix, while retaining the highest amount of information that a rank-1 approximation can have (Eckart–Young theorem [352]).

Pruning by means of network sparsification

The second examined approximation on the LSTM comprises the structured pruning of the weight matrices at each gate. Pruning can interpreted as a type of sparsity in which individual weights are masked as zeros, and therefore the respective computations can be skipped. When studied in

isolation, the proposed structured pruning scheme limits sparsity to the structure of rows of the weight matrices. This selection of granularity allows to always obtain an approximate value for each element of the resulted output vector, instead of having zeroed values at the output vector that carry no information. Overall, the pruning scheme preserves a total of N_{NZ} (non-zero) elements with the highest absolute value *on each row* of each weight matrix. This imposes a uniformity in the sparsity level across rows, allowing for an efficient and hardware friendly implementation. The value of N_{NZ} is tuned to provide the highest possible application-level accuracy, considering the user-specified latency budget, which can be expressed as:

$$\tilde{\mathbf{w}}_j^{\text{pruned}} = \text{prune}(\tilde{\mathbf{w}}_j, N_{\text{NZ}}), \quad \text{for all rows } j \text{ of matrix } \widetilde{\mathbf{W}} \tag{4.9}$$

where $\tilde{\mathbf{w}}_j$ is the j-th row vector of matrix $\widetilde{\mathbf{W}}$ and N_{NZ} denotes the number of Non-Zero elements on each row, determining the desired sparsity level of the resulting vector $\tilde{\mathbf{w}}_j^{\text{pruned}}$. The level of sparsity can be tuned to provide the highest possible application-level accuracy, considering the user-specified latency budget.

More formally, to represent a sparse LSTM, four binary mask matrices are introduced as $\boldsymbol{F}^{\mathbf{g}} \in \{0,1\}^{R \times C}$, one for each gate, with each entry representing whether a connection is pruned or not. Overall, the following notation for a (weight, mask) matrix pair is employed:

$$\{\boldsymbol{W}^{\mathbf{g}}, \boldsymbol{F}^{\mathbf{g}} \mid \mathbf{g} \in \{\mathbf{f}^{(t)}, \mathbf{i}^{(t)}, \mathbf{c}^{(t)}, \mathbf{o}^{(t)}\}\} \tag{4.10}$$

Individual weight values are set to zero by means of a magnitude-based criterion which determines the importance of a weight using its absolute value. Then LSTM pruning can be casted as an optimisation problem of the following form:

$$\min_{\boldsymbol{F}^{\mathbf{g}}} ||\boldsymbol{W}^{\mathbf{g}} - \boldsymbol{F}^{\mathbf{g}} \odot \boldsymbol{W}^{\mathbf{g}}||_2^2, \quad \text{s.t.} \quad ||\boldsymbol{f}_j^{\mathbf{g}}||_0 = N_{\text{NZ}}, \ \forall \mathbf{g} \in \{\mathbf{f}^{(t)}, \mathbf{i}^{(t)}, \mathbf{c}^{(t)}, \mathbf{o}^{(t)}\}, \forall j \in [1, R] \tag{4.11}$$

where $\boldsymbol{f}_j^{\mathbf{g}}$ is the j_{th} row of $\boldsymbol{F}^{\mathbf{g}}$ and N_{NZ} is the number of non-zero elements on each row of $\boldsymbol{F}^{\mathbf{g}}$. $||\cdot||_0$ is the l_0 pseudo-norm denoting the number of non-zero entries in a vector. The solution to the optimisation problem in Eq. 4.11 is given by keeping the N_{NZ} elements on each row of $\boldsymbol{W}^{\mathbf{g}}$ with the highest absolute value and setting their indices to 1 in $\boldsymbol{F}^{\mathbf{g}}$.

In contrast to common practise in existing approaches, the proposed pruning method does not employ retraining and hence removes the requirement of accessing the training set, which is important for privacy-critical applications. Even though the proposed sparsification method does

not explicitly capture the impact of pruning on the application-level accuracy, but considers a numerical error proxy instead, the proposed co-design scheme detailed later in this section searches over different levels of sparsity exploring the effect of pruning on the end-task metric of interest.

Hybrid compression and pruning

Aiming to leverage the advantages of both aforementioned techniques, their combination takes the form of the following weight matrix approximation:

$$\widetilde{\boldsymbol{W}}^{\mathbf{g}} = \boldsymbol{F}^{\mathbf{g}} \odot (\sigma_1^{\mathbf{g}} \cdot \boldsymbol{u}_1^{\mathbf{g}} \cdot \boldsymbol{v}_1^{\mathbf{g}^\top}) \tag{4.12}$$

In this setting, for each gate $\mathbf{g}$ the ranking of the absolute values in each row of the rank-1 approximation $\sigma_1^{\mathbf{g}} \cdot \boldsymbol{u}_1^{\mathbf{g}} \cdot \boldsymbol{v}_1^{\mathbf{g}^\top}$ depends only on $\boldsymbol{v}_1^{\mathbf{g}}$, with each element of $\sigma_1^{\mathbf{g}} \cdot \boldsymbol{u}_1^{\mathbf{g}}$ operating as a shared scaling factor for all elements of a row. Therefore, all the rows of $\boldsymbol{F}^{\mathbf{g}}$ become identical and hence can be represented by a single mask vector $\boldsymbol{f}^{\mathbf{g}} \in \{0,1\}^C$. This leads to a weight matrix with zeros along $(C - N_{\text{NZ}})$ of its C columns, which is described by the following expression:

$$\widetilde{\boldsymbol{W}}^{\mathbf{g}} = \sigma_1^{\mathbf{g}} \cdot \boldsymbol{u}_1^{\mathbf{g}} \cdot (\boldsymbol{f}^i \odot \boldsymbol{v}_1^{\mathbf{g}})^T \tag{4.13}$$

Iterative Refinement

In order to obtain a refinement mechanism that allows to increase the quality of result as a function of time, the computation of each LSTM gate is restructured to employ an iterative approximation method. This involves the progressive refinement of the computed output over a number of iterations, with each refinement step involving the addition of a low-rank approximation of a correction factor (residual) together with its pruning. With this scheme, the final approximate output vector is formed after applying N_{steps} refinement steps:

$$\tilde{\boldsymbol{y}}^{\mathbf{g}} = \sum_{n=1}^{N_{\text{steps}}} \underbrace{\{\sigma_1^{\mathbf{g}(n)} \cdot \boldsymbol{u}_1^{\mathbf{g}(n)} \cdot ((\overbrace{\boldsymbol{f}^{\mathbf{g}(n)} \odot \boldsymbol{v}_1^{\mathbf{g}(n)}}^{\text{pruning}})^T \cdot \tilde{\boldsymbol{x}}^{(t)})\}}_{\text{refinement step}} \tag{4.14}$$

As a result, the workload of each gate is reduced to $N_{\text{steps}} \cdot (2 \cdot R + 2 \cdot N_{\text{NZ}} + 1)$ operations.

The complete approximation process is described by Algorithm 2. On lines 4-6 the first approximation of the weight matrix is constructed by obtaining the rank-1 approximation of the original matrix and applying pruning in order to have N_{NZ} non-zero elements on each row, as in Eq. 4.13. Next, the weight matrix is refined for N_{steps} iterations, by computing the residual (error)

matrix (line 9) and employing its pruned rank-1 approximation as a correction factor updating the weight matrix (line 14). In this way, the error introduced by both the SVD and the pruning are considered, in contrast to the case of progressively applying higher-rank approximations of the original weight matrix, minimising in this way the information loss.

Algorithm 2 Iterative LSTM Model Approximation

Inputs:

1: Weight matrices $\boldsymbol{W}^{\mathbf{g}} \in \mathbb{R}^{R \times C},\ \forall \mathbf{g} \in \{\mathbf{f}^{(t)}, \mathbf{i}^{(t)}, \mathbf{c}^{(t)}, \mathbf{o}^{(t)}\}$
2: Number of non-zero elements, N_{NZ}
3: Number of refinement iterations, N_{steps}

Steps:

1: - - For all gates - -
2: **for** g in $\{\mathbf{f}^{(t)}, \mathbf{i}^{(t)}, \mathbf{c}^{(t)}, \mathbf{o}^{(t)}\}$ **do**
3: - - Initialise weight matrix approximation - -
4: $\left[\boldsymbol{u}_1^{\mathbf{g}(0)}, \sigma_1^{\mathbf{g}(0)}, \boldsymbol{v}_1^{\mathbf{g}(0)}\right] = \text{SVD}(\boldsymbol{W}^{\mathbf{g}})_1$
5: $\boldsymbol{f}^{\mathbf{g}(0)} \leftarrow$ solution to Eq. 4.11 for vector $\boldsymbol{v}_1^{\mathbf{g}(0)}$
6: $\widetilde{\boldsymbol{W}}^{\mathbf{g}(0)} = \sigma_1^{\mathbf{g}(0)} \boldsymbol{u}_1^{\mathbf{g}(0)} \left(\boldsymbol{f}^{\mathbf{g}(0)} \odot \boldsymbol{v}_1^{\mathbf{g}(0)}\right)^T$
7: - - Apply refinements - -
8: **for** $n = 1$ to N_{steps} **do**
9: $\boldsymbol{R}^{\mathbf{g}(n)} = \boldsymbol{W}^{\mathbf{g}} - \widetilde{\boldsymbol{W}}^{\mathbf{g}(n-1)}$
10: - - Compute refinement - -
11: $\left[\boldsymbol{u}_1^{\mathbf{g}(n)}, \sigma_1^{\mathbf{g}(n)}, \boldsymbol{v}_1^{\mathbf{g}(n)}\right] = \text{SVD}(\boldsymbol{R}^{\mathbf{g}(n)})_1$
12: $\boldsymbol{f}^{\mathbf{g}(n)} \leftarrow$ solution to optimisation problem of Eq. 4.11 for vector $\boldsymbol{v}_1^{\mathbf{g}(n)}$
13: - - Update weight matrix approximation - -
14: $\widetilde{\boldsymbol{W}}^{\mathbf{g}(n)} = \widetilde{\boldsymbol{W}}^{\mathbf{g}(n-1)} + \sigma_1^{\mathbf{g}(n)} \boldsymbol{u}_1^{\mathbf{g}(n)} \left(\boldsymbol{f}^{\mathbf{g}(n)} \odot \boldsymbol{v}_1^{\mathbf{g}(n)}\right)^T$
15: **end for**
16: **end for**

Notes: $\text{SVD}(\boldsymbol{X})_1$ returns the rank-1 SVD-based approximation of $\boldsymbol{X}$.

Therefore, in the hybrid method, different combinations of levels of sparsity and refinement iterations correspond to different candidate design points in the computation-accuracy space. In this respect, the number of non-zero elements N_{NZ} in each binary mask vector and the number of refinement steps N_{steps} form tunable parameters that are exposed to the proposed co-design methodology to optimise the LSTM computation-accuracy trade-off.

4.4.4 Hardware Architecture

In this section, a novel hardware architecture is proposed, aiming to overcome the limitations of programmable processors by being tailored to the proposed progressive-inference based implementation for LSTMs, exploiting its properties. The main characteristics of the accelerator design include the adoption of *dataflow processing* to alleviate the overheads of conventional

computing platforms, the *offline precomputation* of SVD and binary masks, the exploitation of both the *inter-gate* and *intra-gate parallelism* of LSTMs to boost performance and the compile-time *tunable scaling* of the architecture to match the available resources and the response-time demands of the target application.

Dataflow processing

In contrast with the control-flow paradigm of general-purpose processing units where individual instructions are scheduled for execution, the proposed accelerator adopts a data-driven dataflow architecture. In this scheme, the availability of input samples triggers the LSTM processing to be performed on them without the need for explicit control and synchronisation between computation units. From a hardware perspective, this approach allows to remove any generic instruction-handling hardware logic and re-purpose the resources of the FPGA chip specifically for LSTMs. In this way, the architecture avoids the time, resource and power overhead of off-the-shelf platforms and boosts the attainable performance by dedicating more hardware resources for computation.

SVD and Binary Masks Precomputation.

In Algorithm 2, the number of refinement iterations (N_{steps}), the level of sparsity (N_{NZ}) and the trained model, i.e. the weight matrices of each gate ($\boldsymbol{W}^{\mathbf{g}}$), are data-independent and known at compile time. As such, the required SVD decompositions along with the corresponding binary masks are precomputed for all N_{steps} iterations at compile time. As a result, the singular values $\sigma_1^{\mathbf{g}(n)}$, the vectors $\boldsymbol{u}_1^{\mathbf{g}(n)}$ and only the non-zero elements of the sparse $\boldsymbol{f}^{\mathbf{g}(n)} \odot \boldsymbol{v}_1^{\mathbf{g}(n)}$ are stored in the off-chip memory, so that they can be looked-up at run time.

Inter- and intra-gate parallelism

Fig. 4.4 shows the block diagram of the architecture. At its core, the architecture is organised as a pipeline of five coarse stages, including four parallel *hardware gate units*, a set of nonlinear operators and a number of multiplier and adder arrays. Starting on the left-hand side, the four *parallel hardware gate units* are the heart of the architecture. The proposed design exploits the coarse-grained, inter-gate parallelism by mapping each LSTM gate to a dedicated hardware gate unit, with all units operating concurrently.

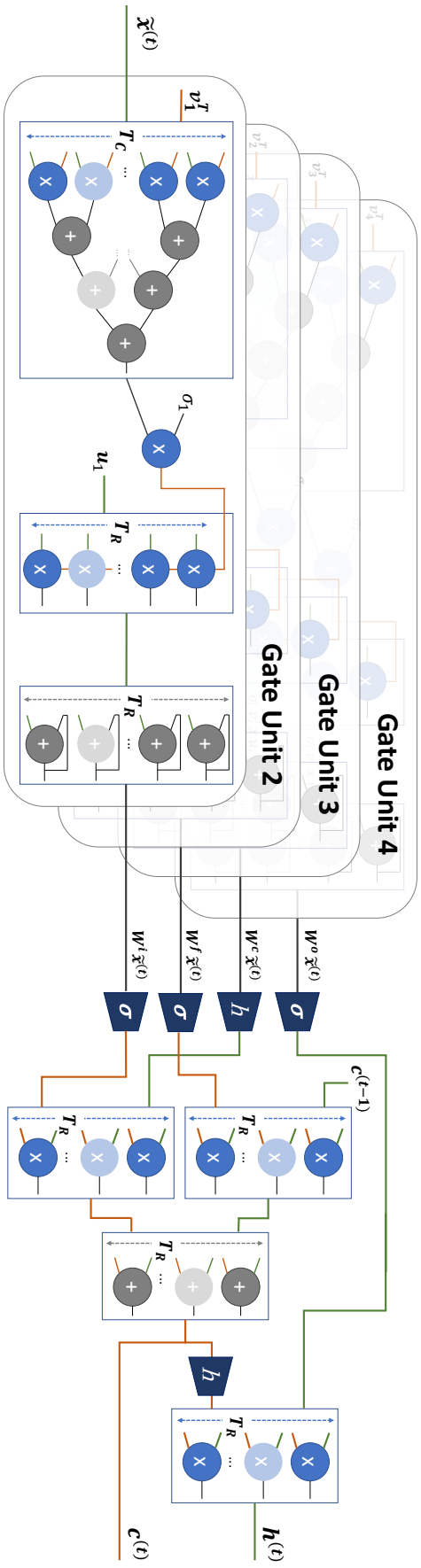


Figure 4.4: Custom hardware architecture for the proposed Approximate LSTM computation scheme.

At each LSTM time-step t, a hardware gate unit computes its output by performing N_{steps} refinement iterations, as in Eq. 4.14. As a first step, the current input vector $\tilde{\boldsymbol{x}}^{(t)}$ is sent from the off-chip memory into an on-chip buffer as it will be reused across all refinement iterations, by all four gates. In the n-th iteration, the singular vectors $\mathbf{u}_1^{\mathbf{g}(n)}$ and $\mathbf{v}_1^{\mathbf{g}(n)}$ for the gate $\mathbf{g}$ are streamed in from the off-chip memory in a tiled manner with tile sizes T_R and T_C respectively, along with the singular values $\sigma_1^{\mathbf{g}(n)}$. This leads to a total of $\left\lceil \frac{R}{T_R} \right\rceil$ and $\left\lceil \frac{C}{T_C} \right\rceil$ tiles respectively, being sequentially streamed in the accelerator.

Internally, each hardware gate unit contains three processing modules: a *dot-product* unit, a *multiplier array* and *adder* array (Fig. 4.4). By mapping the operations of a gate to parallel circuits, the architecture capitalises on the fine-grained, intra-gate, parallelism of these operations to obtain performance gains. Specifically, the dot-product unit is unrolled by a factor of T_C in order to process one tile of $\boldsymbol{v}_1^{\mathbf{g}(n)}$ per cycle. After accumulating the partial results of all the $\frac{C}{T_C}$ tiles, the result is produced and scaled by $\sigma_1^{i(n)}$. The result is passed as a constant operand to the multiplier array, with $\boldsymbol{u}_1^{\mathbf{g}(n)}$ as the other operand. The multiplier array has a size of T_R in order to match the tiling of $\boldsymbol{u}_1^{\mathbf{g}(n)}$. As a final stage, an array of T_R accumulators performs the summation across the N_{steps} iterations as expressed in Eq. 4.14, to produce the final gate output.

After the hardware gate units have applied all the necessary refinements, the outputs of the four gates are passed through their corresponding nonlinear operators. Subsequently, the produced outputs are multiplied element-by-element to produce the cell state $\boldsymbol{c}^{(t)}$ (Eq. 4.2) and the current output vector (through an additional $\tanh(\cdot)$ non-linearity) $\boldsymbol{h}^{(t)}$ (Eq. 4.3). All three multiplier arrays and the one adder array for this post-processing have a size of T_R to match the tile size of the incoming vectors and exploit the available parallelism.

Configurable scaling

At compile time, the configuration of the architecture is controlled by means of two parameters: $T_R \in [1, R]$ and $T_C \in [1, N_{\text{NZ}}]$. T_R controls the size of all the arrays, while T_C determines the number of multiply-add operators in each hardware gate unit. Different values of T_R and T_C correspond to different scaling of the architecture and provide a tunable performance-resource cost trade-off which is used to customise the design based on the available resources and the response-time requirements.

4.4.5 Design Space Exploration

Based on the discussion on the previous sections, the LSTM inference approximation method is parametrised by N_{NZ} and N_{steps}, while the hardware architecture by T_R and T_C. In this section, an analytical performance model for the proposed hardware architecture is developed. The aim is to investigate the attainable performance of different architectural configurations, and subsequently optimise its parametrisation, given an approximation algorithm parametrisation. For this purpose, the roofline model [315] is adopted, due to its ability to *analytically* model the attainable performance, allowing for all tile size combinations $T_R \times T_C$ to be explored exhaustively, as well as its ability to also identify the causes of performance bottlenecks (i.e. being computation or memory related).

Specifically, given the pruning level N_{NZ} and the number of refinements N_{steps}, the *attainable performance* (in GOp/s) of the accelerator can be modelled as a function of its architectural configuration (T_R, T_C):

$$Perf = \frac{workload}{cycles} clk = \frac{4 \cdot N_{\text{steps}} \cdot (2 \cdot N_{\text{NZ}} + 2 \cdot R + 1) + 37 \cdot R}{max(N_{steps} \cdot max(\frac{R}{T_R}, \frac{N_{\text{NZ}}}{T_C}), 37 \cdot \frac{R}{T_R})} \cdot clk \quad (ops/s) \tag{4.15}$$

where each gate performs $2 \cdot N_{\text{NZ}} + 2 \cdot R + 1$ operations per iteration and $37 \cdot R$ accounts for the rest of the operations required to calculate the cell-state update and LSTM output. The *cycles* (per samples), also known as Initiation Interval, are determined based on the slowest between the gate stage and the rest of the computations. Notably, a last portion of the cycles for the latter are spent on the calculation of the non-linear functions, that generally challenge custom-hardware accelerators when implemented in full precision. However, although efficient and hardware-friendly approximations for these functions are broadly studied in the literature [353], experiments showed that in the case of LSTMs (where typically $R << C$) these computations do not form a computation bottleneck. Similarly, a gate's initiation interval depends on the slowest between the dot-product unit and the multiplier array (Fig. 4.4).

As this chapter focuses on LSTMs whose entire set of weights do not fit in the on-chip memory of an FPGA, the *operational intensity* is also modelled, as multiplication and addition operations per byte of weights accessed from the external memory (GOp/byte):

$$CTC = \frac{workload}{mem\ access} = \frac{4 \cdot N_{\text{steps}} \cdot (2 \cdot N_{\text{NZ}} + 2 \cdot R + 1) + 37 \cdot R}{4 \cdot (4 \cdot N_{\text{steps}} \cdot (N_{\text{NZ}} + R + 1) + R + N_{\text{NZ}})} \quad (ops/byte) \tag{4.16}$$

where the off-chip memory transfers include the singular vectors and the singular value for each iteration of each gate, as well as the loading of the input and the write-back of the output vectors.

Notably, the cell state $\boldsymbol{c}^{(t)}$ and augmented input vector $\tilde{\boldsymbol{x}}^{(t)}$ are kept on-chip in order to be reused across the input samples and the N_{steps} iterations, respectively. All data are represented with a single-precision floating-point format and require four bytes, while double buffering is employed to hide some of the latency of memory transactions by overlapping them with computation. As such, the on-chip memory requirements of the proposed architecture are modelled as:

$$onChipMem = 2 \cdot (R + N_{\text{NZ}} + 4 \cdot (2 \cdot T_R + T_C + 1)) \cdot 4 \quad (bytes) \tag{4.17}$$

The small number of design points allows enumerating all possible tile size combinations $T_R \times T_C$ across different levels of sparsity N_{NZ} to obtain the performance and CTC values for the complete design space. Similarly to the methodology followed in Chapter 3, the target platform's peak performance, memory bandwidth and on-chip memory capacity are considered to identify the subspace of platform-supported design points. Consequently, the highest performing architectural configuration for each level of sparsity is identified, within that subspace.

4.4.6 Model-Hardware Co-design

Different combinations of the approximation and architectural parameters correspond to alternative system designs. For a fixed-time constraint, each candidate design is characterised by its: (i) quality of result (QoR), (ii) performance in terms of processing speed and (iii) resource consumption. To explore this space, we need to study the effect of the architectural parameters on the performance of the hardware implementation as well as the impact of the approximations on the QoR of the target application. In terms of the hardware design, the performance model introduced on the previous section is employed for this purpose. Accordingly, the approximation method parameters are studied based on an application-level metric (discussed in Section 4.5.1), that measures the impact of each applied approximation on the accuracy of the target application (assuming the existence of a small evaluation data set).

Instead of fixing the approximation parameters first and then explore the architectural configuration to map on hardware (model-aware hardware design), or vice versa (hardware-aware model design), this section co-optimises all approximation (N_{NZ}, N_{steps}) and hardware (T_R, T_C) parameters following a model-hardware co-design paradigm. This process happens in view of

application-level requirements in inference speed and accuracy, after establishing corresponding metrics able to capture the effects of each parameter on performance and accuracy.

More precisely, initially the error induced by the proposed LSTM approximations on an application is experimentally measured as a function of the targeted iterations. Given an (N_{NZ}, N_{steps}) pair, the approximate LSTM is generated from the original LSTM. Next, the target application in run end-to-end over the provided Validation Set using both the original and the approximate LSTM. By treating the final output of the original model as the ground truth, an application-specific metric is employed to assess the QoR of the approximate LSTM. The quality metric measures the similarity between the original and the approximate result and must have a suitable form based on the target domain, such as the relative error between the approximate and reference result or the Kullback-Leibler (KL) divergence that captures the distance between the respective probability distributions.

Overall, by varying the values of (N_{NZ}, N_{steps}) and evaluating the accuracy of the resulting scheme along with conducting the proposed DSE methodology that selects the corresponding architectural configuration (T_R, T_C) and obtaining its performance, the relationship between the level of approximation and the QoR (in terms of both accuracy and latency) is captured. This allows exploring the underlying trade-off, in view of the target application and FPGA device at hand.

4.5 Evaluation: A Case study on Autonomous Driving

4.5.1 Autonomous Driving Background

One of the emerging AI-driven applications with the highest potential for societal impact is autonomous driving. Although initial efforts began in the late 1980s [354], the field of autonomous driving has experienced significant progress in the past decade, owing to efforts from both the industrial and academic communities. The main enablers of the emerging technologies being developed are: (i) the advancement of *deep learning algorithms* allowing the extraction of powerful representations, (ii) the availability of *real-world training data* provided by open-source datasets [22][355] and (iii) the development of *embedded processing platforms* with enhanced computational capabilities that allow the deployment of computationally expensive software on-board the vehicle [356][336], satisfying the imposed low-latency and safety constraints.

Vision-based driving assistance and autonomy [357][26][358] is gaining attention due to the low-cost, widely available cameras that can be used independently or accompany other sensors for environmental perception. With such sensors providing a stream of measurements, recurrent models such as LSTMs form a promising learning paradigm that can extract and exploit temporal information from the incoming data to develop a smooth and consistent driving policy, in place of the independent per-input predictions provided by classical deep learning models that exploit solely spatial information [359].

Self-driving car systems consist of a large set of computationally demanding tasks, including sensor pre-processing, localisation, mapping, path planning and obstacle avoidance, control and emergency handling [360]. Hard low-latency constraints [361] between perception and action impose the need for high-performance implementations that guarantee the extraction of highly accurate approximations on each individual component, to meet the real-time performance requirements of the overall system with insignificant effect on accuracy. As an example, a coarse but in-time estimation of the vehicle's obstacle avoidance system to take a "sharp" left turn and avoid a collision, is preferred to a delayed but rather accurate regression of an exact steering angle response to a visual input.

4.5.2 Experimental Setup

Target Application

The end-to-end driving model presented in [362], trained on the Berkeley DeepDrive Video dataset (BDDV), a large-scale crowdsourced driving video dataset forming an early version of the BDD100K Dataset [355], is examined as a case study for evaluating the proposed framework. Input frames for each video are first processed by a Fully-Convolutional Network (FCN) to encode the spatial features which are then fed to a trained LSTM model that predicts the probability distribution across a discrete set of feasible future actions for the vehicle (go forward, stop, turn left, turn right) taking advantage of the temporal motion information from previous representations. The LSTM input is enhanced with the linear and angular velocities of the vehicle predicted by the system from the previous frame. This FCN-LSTM architecture is a novel version of Long-term Recurrent Convolutional Networks (LRCNs), typically consisting of a convolutional neural network feeding its output to an LSTM, combining the current state-of-the-art in visual and sequence learning to extract spatio-temporal information for input streams. By replacing the CNN with a FCN, the

learned visual representation preserves the locality of information, while the end-to-end network is trained jointly with a side task of semantic segmentation undertaken by the FCN.

This chapter's experiments focus on the LSTM part of the examined driving model, each gate of which forms an $R \times C$ augmented weight matrix, with $R = 64$ and $C = 4160$. The method is evaluated on a previously unseen part of the validation set of BDDV dataset that was used to train the model, by cropping a segment of 100 consequent frames from over 1800 real videos of diverse driving scenarios. As in the original work, ground truth labels are obtained using GPS/IMU data from the vehicle and discretised to four classes. To generate action probability distributions, that will act as ground truth for the evaluation of the proposed approximation method, the proposed co-design process is followed for the execution of the driving model end-to-end over the validation set. In order to measure the effect of the low-rank approximation and pruning algorithms on the target application and clearly demonstrated the progression of the quality-of-results obtained at every approximation step, the Kullback-Leibler (KL) divergence[3] -a commonly used metric of dissimilarity between distributions- is calculated between the original and approximated models' predictions, at different timesteps.

Configuration

The Xilinx's ZC706 board mounting a Zynq 7045 chip is targeted in the experiments. This platform is an industry standard for FPGA-based embedded systems and is based on the Zynq-7000 System-on-Chip which integrates a dual-core Arm CPU alongside an FPGA fabric on the same chip. During inference, the CPU coordinates the operation of the system by: (i) scheduling the computations between different tiles from all 4 LSTM gates and mapping them to the available processing elements of the custom hardware accelerator, and (ii) setting up the communication interface between the accelerator and the external memory. To this end, the four AXI-based High-Performance (HP) ports that are available on the target device are used. Each port is configured with a 64-bit width and instantiate a dedicated DMA engine (initialised by the CPU prior to execution), clocked at 150 MHz, to independently perform the memory transfers. For the data format, single-precision floating-point representation (*float32*) is used to comply with the typical precision requirements of LSTMs as used by the deep learning community.

[3]The KL divergence between a target $\hat{\mathbf{p}}$ and approximated $\mathbf{p}$ distribution of length N is given by $KL(\mathbf{p}, \hat{\mathbf{p}}) = \sum_{i=1}^{N} \hat{\mathbf{p}}_i \cdot \log(\frac{\hat{\mathbf{p}}_i}{\mathbf{p}_i})$. Identical distributions have a KL value of 0, otherwise KL can take any positive value up to $+\infty$.

All hardware designs are synthesised with Vivado HLS and Vivado Design Suite (v2017.1) achieving a clock frequency of 100 MHz. The driving model is run on tensorflow to obtain accuracy metrics, whereas the proposed framework and all approximations are implemented on Matlab (v2018a).

To ensure a fair comparison with baselines deployed on different compute platforms (such as CPUs and GPUs) on Table 4.1, performance results are normalised with respect to the power consumption during inference. For this purpose, power is measured through an external power-monitoring device, where measurements are averaged across inference time after idle power of each development board is subtracted.

Baseline Architecture

The core LSTM workload of the proposed approximate computing scheme, as well as the baseline LSTM implementation, are deployed on the FPGA. The baseline follows a highly-optimised faithful mapping of the original LSTM to a matrix-vector multiplication engine based accelerator from the literature, tailored to LSTM inference [333].

In more detail, this baseline architecture consists of four gate units with a total of 1.06M *float32* parameters (surpassing by more than 1.7× the entirety of the on-chip memory available on the target FPGA). Each gate is implemented in a concurrent hardware unit that performs the matrix-vector multiplication operations of LSTM gates (Fig. 4.3) in a blocked manner. The computational workload for the kernel of each gate is $2 \cdot R \cdot C$ operations. Parametrisation with respect to the tiling along the rows (T_R) and columns (T_C) of the weight matrices is applied and roofline modelling and design space exploration are used to obtain the highest performing configuration (T_R, T_C), similarly to the proposed system's architecture (Fig. 4.5).

To obtain the application-level QoR of the baseline design under time-constrained scenarios, the KL divergence between the intermediate LSTM output at each tile step of T_R and the predictions of the reference model is examined, emulating progressive inference though a faithful implementation.

4.5.3 Architectural Performance

First, the attainable performance of the proposed hardware accelerator is evaluated and compared to the baseline architecture. Simultaneously, the architectural implications of different levels of sparsity for the proposed approximation scheme are also ablated. As demonstrated in the roofline analysis [315] across different architectural and approximation variants illustrated in Fig. 4.5,

all configurations are memory-bound (i.e. while scaling up they meet the memory roof of the target device earlier than the computational roof). In this case, their attainable performance is vertically projected on the memory roof, as the instantiated compute units starve from data and spend idle time anticipating the completion of memory transactions. Typically, to avoid the ineffectual area overhead of this situation, smaller designs achieving equal performance while exploiting less parallelism between computation (to the degree that can be accommodated by the memory), as selected by the DSE, leading to significantly lower utilisation of the available compute resources and performance barriers.

The baseline appears to be affected the most by this issue, while it can be seen that by introducing sparsity through the proposed pruning methodology, the Computation-to-Communication ratio increases proportionally, leading to higher device utilisation and thus up to 3.6× higher attainable performance. Nonetheless, higher sparsity introduces larger approximation errors, that translates to a larger number of refinement iterations required to reach the same accuracy, incurring a corresponding overhead in inference latency. To balance this trade-off, the proposed toolflow jointly considers metrics of hardware performance and approximation accuracy in its exploration, essentially co-designing the software and hardware configuration of the resulting system, for the LSTM-FPGA pair at hand, in-view of the provided application-level requirements.

Notably, however, all designs remain memory-bound, which is an inherent characteristic of LSTMs, due to the minimum re-use of their weights at each inference [44]. At the same time data dependencies (e.g. on the cell state between time-steps) don't allow to exploit any parallelism across the temporal dimension [363].

4.5.4 End-to-end Evaluation

Latency-Accuracy Trade-off

In this section, the latency-accuracy trade-off provided by different instantiations of the proposed approach, adopting varying levels of sparsity N_{NZ}, is evaluated as a function of the number of refinement steps N_{steps} and inference latency t. The results are illustrated in Fig. 4.6a and b respectively. In the latter case, latency measurements are obtained on the FPGA device after running a complete DSE for each instance to determine the highest performant architectural configuration. The faithful LSTM baseline described above, is also illustrated (with a solid black line) on both cases.

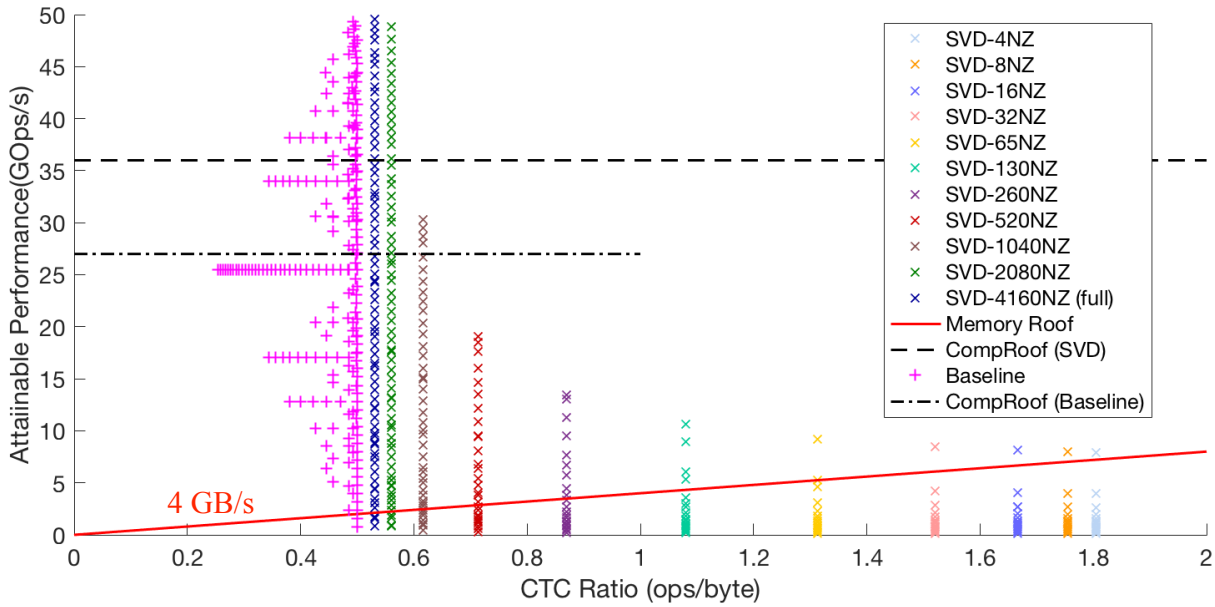
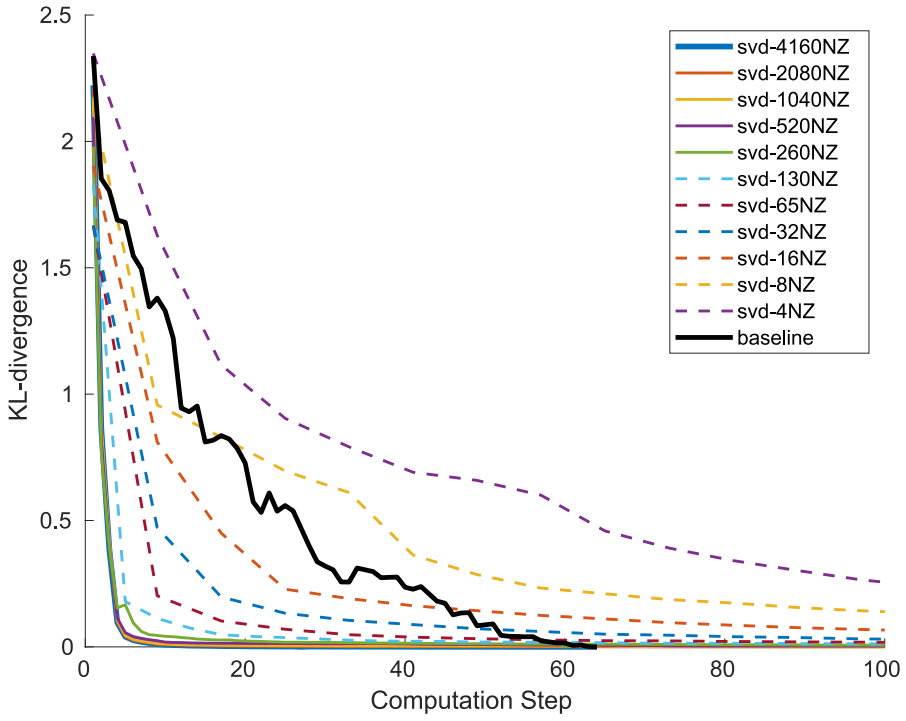


Figure 4.5: Roofline model analysis for the baseline architecture and various configurations of the proposed method (with different sparsity levels), on ZC706.

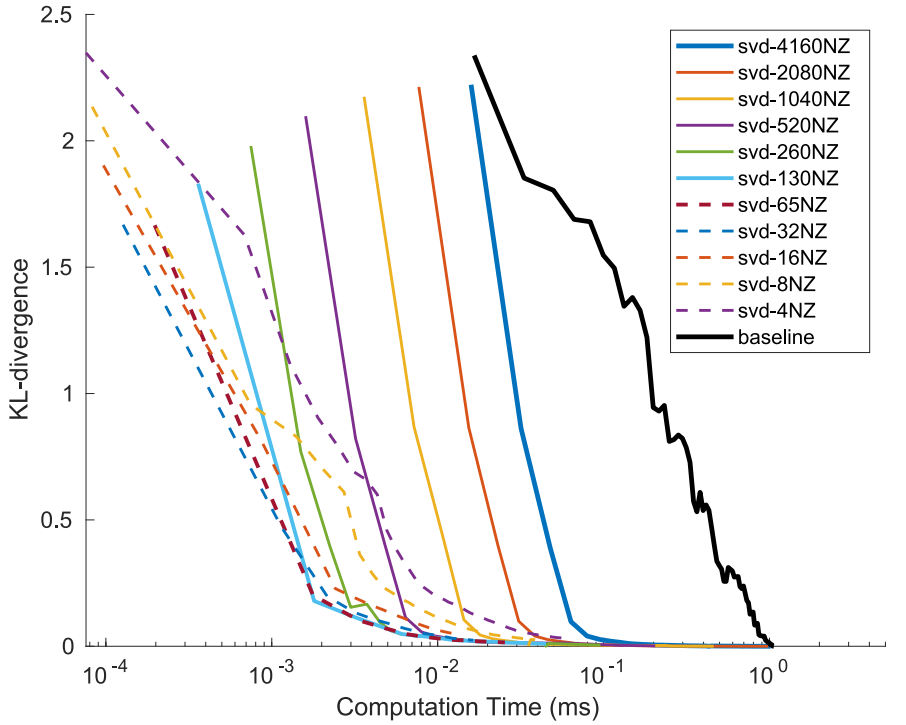
As illustrated in Fig. 4.6a, the quality of result of a configuration is inversely proportional to its level of sparsity. Dense configurations, such as those with 50% non-zero elements or more, tend to converge to negligible divergence values (below 10^{-6}) in less than 15 computation steps, in contrast with sparser configurations that require more than 75 computations steps to converge to the same divergence level (∼2% non-zero elements) or converge to higher divergence values (as in the case of $N_{\text{NZ}} = 16$, featuring ∼0.4% non-zero elements).

Since computation time per computation step is also inversely proportional to the level of sparsity of a given configuration, Fig. 4.6b suggests that some sparse configurations demonstrate superior accuracy than other denser settings, when examined under the same latency constraint. This behaviour, however, is not monotonic due to the fact that extremely sparse configurations require a significantly larger number of computation steps to reach the same level of accuracy. Therefore, the selection of the appropriate level of sparsity is dependent on the latency constraint imposed by the application-level needs. It can be argued that the design point with $N_{\text{NZ}} = 32$ achieves a sweet-spot in the overall latency-accuracy trade-off, showcasing the effectiveness of the proposed hybrid approximation scheme, compared to its SVD-only (dense) counterpart. Overall, it is observed that the proposed methodology achieves a speed-up of 198× on average (76× geo. mean) across different quality-of-result levels compared to the baseline approach.

In particular, when only negligible KL-divergence is allowed (KL $\leq$ 0.001) between the



(a)



(b)

Figure 4.6: KL-divergence between approximate prediction and reference model output (lower-is-better) as a function of: (a) Computation Step, (b) Computation Time.

approximate and reference prediction, the proposed system achieves 2.93× faster inference by exploiting the LSTM model's inherent redundancy. Furthermore, the proposed method demonstrates up to 317× lower inference time to achieve an intermediate QoR prediction (KL $\leq$ 0.1) exploiting the computation time-accuracy trade-off.

Conclusively, by exploiting the computation time-accuracy trade-off, the proposed progressive inference methodology can provide high-quality approximations of the final result at early stages of the computation, which are iteratively refined as more time budget becomes available. This scheme is particularly useful for systems with hard computation-time constraints (*e.g.* in mission-critical real-time applications), enabling them to maximise the attainable quality of result within the given latency envelope. Furthermore, the introduced highly-parametrised custom hardware architecture for the proposed methodology demonstrates remarkable power efficiency by exploiting the enhanced customisation capabilities and flexibility of FPGAs. In this manner, highly-optimised hardware mappings of different configuration instances of the proposed approximation scheme are generated, while being tailored to the needs of the target application.

Quantitative Comparison to Baselines

In this section, the impact of the proposed approach is thoroughly evaluated in comparison to the faithful FPGA baseline, as well as highly-optimised CPU and GPU implementations of LSTM inference. For this purpose, the pytorch LSTM implementation is adopted and deploy it on the NVIDIA Jetson AGX Xavier board. This platform employs an 8-core ARM 64-bit CPU along with a 512-core Volta GPU. The most equitable metric for cross-platform comparison is power efficiency, as it effectively normalises the results with respect to the available computational resources of each target platform. The idle power is subtracted from all measurements, to allow for a comparison between the actual power consumed by the LSTM execution (capturing both computation and memory transactions).

Table 4.1 summarises the results of this comparison. The first 3 rows present a comparison between the adopted baselines. The employed FPGA baseline achieves a 2.28× speed-up compared to the CPU implementation, while suffering a 0.75× slow-down with respect to the GPU, in terms of absolute latency. However, when power consumption is also considered, these results are translated into a 4.31× and 1.96× improvement on power efficiency compared to the CPU and GPU baseline respectively. These demonstrated gains in power efficiency achieved by the use of a

Table 4.1: Quantitative comparison of the proposed approach with CPU, GPU and FPGA baselines.

Platform	Benchmark	Latency	Power	Power Efficiency
CPU	Baseline	2.4266 ms	5.13 W	0.342 GOp/s/W
GPU	Baseline	0.7974 ms	7.11 W	0.752 GOp/s/W
FPGA	Baseline	1.0620 ms	2.72 W	1.476 GOp/s/W
FPGA	†* (KL≤ 0.001)	0.36190 ms	2.76 W	4.267 GOp/s/W
FPGA	†** (KL≤ 0.01)	0.03924 ms	3.20 W	33.943 GOp/s/W
FPGA	†** (KL≤ 0.1)	0.00335 ms	3.20 W	397.713 GOp/s/W
FPGA	†*** (KL≤ 1.0)	0.00072 ms	3.41 W	1735.992 GOp/s/W

† This work, *SVD-4160NZ (no pruning), **SVD-130NZ, ***SVD-32NZ

custom FPGA-based solution render FPGAs as a strong candidate platform for LSTM deployment in many power-constrained applications, especially in the embedded space of autonomous systems.

Multiple instances of the proposed approximate computing scheme are also listed in the last 4 rows of Table 4.1, for different KL-divergence requirements. Indicative experiments showed that approximate model predictions with $KL < 1$ were already able to predict the same top-1 class with the original model in 96% of the examined validation set (with notably lower confidence in many cases), while $KL < 0.001$ reassembles an almost perfect match between the two distributions. For reference, qualitative examples of how KL divergence translates to classification accuracy on the target task are also illustrated in Fig. 4.7 (discussed later).

It can be seen that by utilising solely the proposed computation-restructuring methodology, a speed-up of 6.7×, 2.2× and 2.93× is achieved in the latency required to yield (almost) identical outputs (KL≤ 0.001) with the reference design, compared to the CPU, GPU and FPGA baselines accordingly, also translated into an improvement of 12.46×, 5.67× and 2.89× in power efficiency. These significant gains arise by the proposed methodology exploiting the inherent redundancy of LSTM models in order to maximise the achievable accuracy at every stage of the computation. By performing the most information-carrying computations first, the workload (and computation time) required to reach similar accuracy with the baseline is effectively reduced.

By relaxing the error tolerance into slightly higher KL-divergence values (≤ 0.1), the proposed hybrid compression-and-pruning methodology provides informed approximations of the inference outputs, while demonstrating remarkable performance gains of up to 724×, 238× and 317× in latency (1161×, 529× and 269× improved power efficiency) compared to the same CPU, GPU

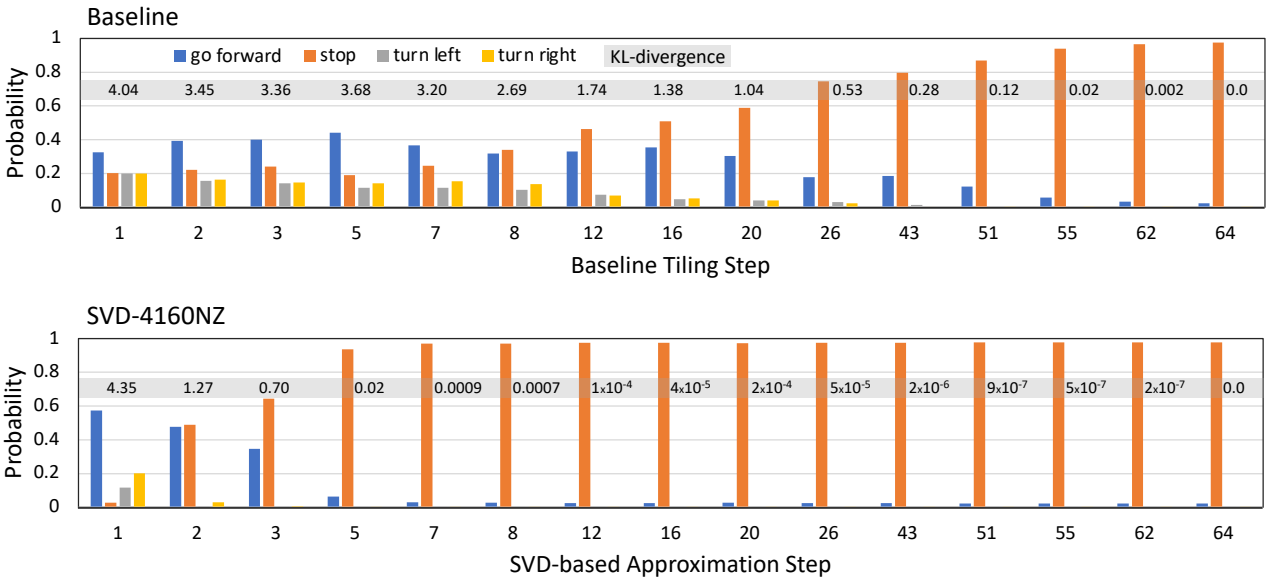


Figure 4.7: Intermediate prediction instances obtained by the progressive inference baseline and the proposed SVD-based approach ($N_{\text{NZ}} = 4160$) on the same data sample, as a function of computation steps. (Computation time per step is similar between these two cases.)

and FPGA baselines respectively. These gains are amplified remarkably by further relaxing the error tolerance into higher KL-divergence (≤ 1.0), which however still yield "meaningful" results.

Since all the configurations of the proposed (and baseline) approach are memory bounded (Fig. 4.5), the attainable parallelism and performance (GOp/s) of the underlying hardware architecture increase proportionally to the selected sparsity level. As it can be noticed in Table 4.1, the absolute power consumption of sparser configurations increases. This is due to the fact that since sparser configurations lead to higher CTC ratio (Fig. 4.5), more parallel processing can be exploited in this case, by instantiating more on-chip computational resources on the FPGA. Consequently, although processing becomes faster, the absolute power of the accelerator also increases as a result of the on-chip power consumption.

4.5.5 Qualitative Analysis

Finally, in this section, a qualitative analysis of the proposed progressive computation scheme, along with a comparison to the faithful baseline implementation, is presented to showcase its benefits and impact directly on the target application.

Initially, Fig. 4.7 presents a set of predicted probability distribution instances corresponding to several progressive refinement steps for a representative input frame along with their corresponding KL-divergence values. For the proposed scheme, the dense (SVD-4160) instance is selected, as it

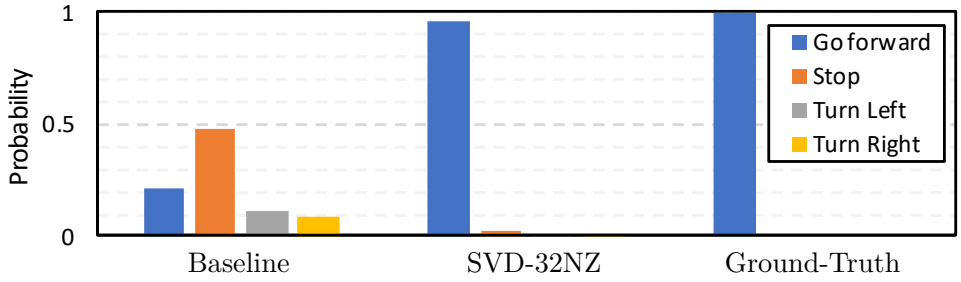


Figure 4.8: Intermediate prediction instances obtained by the baseline and the proposed approach with N_{NZ}=32 on the same data sample, under the same latency constraint (t=10^{-1}ms).

features a comparable computation time per step (16.6^{-2}ms/step) to the baseline implementation (15.9^{-2}ms/step). It can be seen that the proposed approach convergences to "meaningful results" (application-wise) and significantly lower KL-divergence values, in much smaller number of computation steps, by exploiting the inherent redundancy of the LSTM model. This highlights the improved quality of result of the proposed iterative refinement scheme as a function of time budget.

Fig. 4.8 describes the same story from a different view-point. Two representative intermediate probability distributions are illustrated, extracted by an instance of the proposed approach and the baseline, both satisfying the same latency constraint of 10^{-1}ms. For the proposed approach, the sparse SVD-32NZ instance is identified as the operational sweet-spot, and selected for deployment. To obtain these outputs, both methods were fed with the same input and while calculating their predictions their computation was cut short as the available time budget was hit. The illustrated intermediate output distributions indicate that the proposed approach makes a more informed prediction, significantly closer to the ground-truth compared to the baseline. This property is particularly useful in scenarios where tight real-time requirements impose hard latency constraints on the available computation time budget for inference.

4.6 Discussion

This chapter focused on approximations that were jointly applied across the computation and model layer of the deep learning deployment stack. In this context model pruning was employed to sparsify a pre-trained LSTM model, effectively reducing its computational and memory footprint. Alongside, a novel SVD-based parameter approximation scheme was introduced to re-structure the underlying computation in an information-maximising manner. A novel hardware architecture,

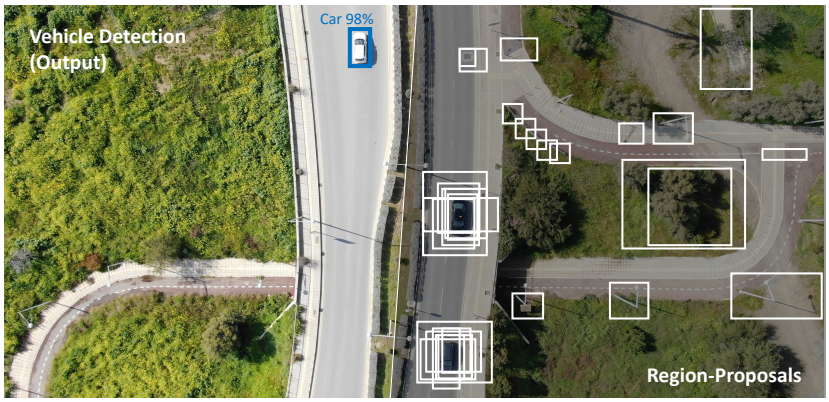
tightly coupled to the combined approximation methodology is introduced, to enable the realisation of performance gains through the proposed model pruning and iterative refinement schemes.

The proposed pruning methodology adopted a well-defined sparsity structure, to allow its efficient mapping on hardware and enable the attainment of corresponding performance gains by reducing the memory-boundness of the workload. Simultaneously, the proposed SVD-based approximation scheme reduces the workload required to achieve the desired quality of result for the target application. Taking advantage of this information maximising approximation, a progressive inference scheme is also introduced, enabling the iterative refinement of the model prediction to facilitate "anytime inference" for time-constraint usecases. All three techniques are applied post-training on the weights of a given LSTM and, hence, don't rely on accessing the training data of the original model.

A configurable hardware architecture, being customisable scale its performance with the sparsity level of the underlying model in order to maximise its attainable performance is also introduced. Alongside, an analytical performance model enables instantiating the most efficient accelerator tailored to the target use-case through exhaustive design space exploration.

As such, the proposed methodology is wrapped as an automated toolflow that instantiates the proposed system in view of an LSTM-FPGA pair and a validation set for the target application where user-specified speed-accuracy requirements can be evaluated. This formulation enables the co-optimisation of the LSTM approximation and the architecture in order to satisfy the application-level computation time constraints

In the context of this chapter, the proposed approach was applied on the task of end-to-end autonomous driving, where LSTMs form a main component of systems deployed in latency-critical settings. The resulting system was able to provide approximate, but informed, LSTM predictions in time-constrained environments. As a result a significantly improved speed-accuracy trade-off on the application level was offered, compared to conventional implementation approaches for LSTMs.



5

Informed Region Selection for Efficient UAV-based Object Detection

Contents

5.1 Overview . . . 130
5.2 Motivation . . . 131
5.3 Related Work . . . 134
5.3.1 Convolutional Object Detectors . . . 134
5.3.2 Efficient UAV-based Object Detection . . . 136
5.3.3 Vehicle Detection Datasets . . . 138
5.4 The CyCAR Dataset . . . 138
5.4.1 Context . . . 139
5.4.2 Data and Annotations . . . 139
5.5 Methodology . . . 141
5.5.1 Overview . . . 141
5.5.2 Informed UAV-based Region Selection . . . 142
5.5.3 Region Selection Criteria for Vehicle Detection . . . 144
5.5.4 GPU Deployment Aspects . . . 151
5.6 Evaluation . . . 152
5.6.1 Experimental Setup . . . 152
5.6.2 Speed/Accuracy Trade-offs in Object Detectors . . . 153
5.6.3 End-to-end Evaluation . . . 155
5.6.4 Ablation on Region Selection Criteria . . . 158
5.6.5 Qualitative Analysis . . . 159
5.7 Discussion . . . 160

Keywords: *CNNs, Object Detection, Dynamic Region Selection, Embedded GPUs, UAVs, Mobile Sensing, Vehicle Detection, Domain-specific AI models, CyCAR Dataset*

5.1 Overview

Most approximations applied on the computation layer typically require specialised hardware support or rely on custom hardware accelerators to translate their workload savings to realistic inference speed-ups. Although demonstrating great potential, this approach can lead to extended development cycles and prohibitively increase the cost for certain applications. With the emergence of data-driven methods, an alternative direction towards efficient deployment resorts to exploiting application-specific information, in order to develop lower-capacity domain-specific models that are highly optimised for the target task; in essence trading generalisability for efficiency.

Along these lines, this chapter focuses on the ***model layer*** *of the stack, introducing a post-training approximation methodology that exploits prior domain knowledge to identify and eliminate redundant computation at inference time. More specifically, this chapter studies deep learning-based object detection in the context of remote sensing platforms, such Unmanned Aerial Vehicles (UAVs), where additional sensor cues are available. Currently, the on-board deployment of state-of-the-art two-stage object detectors (comprising a region-proposal network, followed by a box classifier acting on each proposal), in settings that impose low-latency constraints during inference (i.e. in order to make mission-critical decisions in real-time), is challenged by their excessive computational demands. To enable the efficient deployment of region-based detectors in aerial imagery, an informed application-specific and input-dependent approximation methodology is introduced. The proposed approach integrates prior domain knowledge, along with information from the on-board sensors of the UAV (such as flying altitude), in between the two stages to identify and eliminate false-positive proposals at an early stage of the computation, reducing dynamically the number of candidate detections and the underlying workload for each input, while sustaining the detection accuracy. To facilitate the development of such approaches, a novel vehicle-annotated and altitude-stamped dataset of real UAV imagery is introduced, being captured at numerous flying heights under a wide span of traffic scenarios, reassembling challenging real-world situations.*

This chapter is based on a conference paper co-authored with Dr. Christos Kyrkou and Prof. Christos-Savvas Bouganis: (IROS 2019).

5.2 Motivation

Apart from image classification, deep learning has become a prominent technology across several downstream vision tasks, with Convolutional Neural Networks achieving state-of-the-art accuracy in object detection [4] and semantic segmentation [202]. This advancement acts as an enabler for a wide range of emerging applications related to autonomous systems, such as self-driving cars [364] and Unmanned Aerial Vehicles (UAVs) [72].

In particular, Micro Aerial Vehicles (MAVs) are gaining increasing attention, as a result of the extensive availability and ease-of-use of commercially available low-cost quadcopters. Being usually equipped with forward- and downward-looking cameras, drones can act as a mobile remote sensing platform for real-world applications including infrastructure inspection, emergency response, search and rescue operations, surveillance and traffic monitoring, by exploiting computer vision algorithms to analyse the incoming visual data captured from their on-board visual sensors.

In this way, UAVs can contribute towards "smart cities" by facilitating infrastructure-free situational awareness, with deep learning algorithms taking over the underlying visual understanding tasks. Focusing on traffic monitoring [365] as an example, UAVs can provide remote traffic estimation though a post-processing of vision-based vehicle detection. This information can be used for traffic regulation purposes or to aid the route planning of emergency response vehicles etc. Moreover, the wide field-of-view along with the enhanced mobility of UAVs allow their rapid and facile deployment for remote sensing in areas-of-interest, eliminating the need for expensive infrastructure in urban or rural environments, such as fixed surveillance cameras or embedded car sensors [366].

Simultaneously, CNN-based object detectors have experienced tremendous progress through the last years, demonstrating impressive accuracy. This advancement however, came at the cost of increased computational workload, challenging their real-world deployment. In some perception tasks, where one or more UAVs are regularly deployed to collect data from an area-of-interest, a cloud-based setup (Fig. 5.1a) performing the analysis of captured data on remote base servers is applicable [367].

At the other end of the spectrum, UAV-based deep learning models can be in-the-loop of mission critical decision making including navigation and collision avoidance [96]. In such real-time use-cases, strict low-latency requirements are imposed during inference, making the cost of a wireless



Figure 5.1: Illustration of cloud-based and on-device processing settings.

link between the drone and the base station prohibitive. Moreover, when deployed in remote areas such communication may not be possible to be established [368]. Besides, communication poses an added security risk for intercepting and exposing sensitive data. Near-sensor/On-device processing (Fig. 5.1b) provides an alternative approach for such scenarios. However, in this deployment setting, the challenges are aggravated by the limited computational resources and low-power constraints necessitated by the low-payload capabilities of UAVs.

Specifically for the task of vehicle detection, both cloud-centric and near-sensor processing settings can be considered applicable [369], according to the underlying application. In the case of vehicle tracking and following for example, navigational decisions for the UAV itself rely on the predicted detection, imposing real-time latency constraints, implying a reliance to on-board processing. In contrast, traffic-monitoring scenarios remain an online process, as the obtained information can be used for critical decision-making about traffic regulation or emergency response. Furthermore, in this setting, the maximum sample-rate that can be accommodated by the model also determines the maximum operational speed of the UAV (above which important information will start to be missed due to the underlying frame-drop). As such, although cloud processing remains an option for this application type, softer near-real-time (NRT) requirements are usually still in-place, and latency remains a critical factor for optimisation.

Single-shot detectors (SSD) [5][191] are well-suited to meet these requirements, being positioned at the low-latency end of the performance-accuracy pareto frontier [370]. As such, Single-shot detection models currently dominate the deployment landscape of embedded systems [371], including mobile robots [372]. SSD-based approaches, however, suffer from significant accuracy degradation in objects with small spatial resolution, due to the small input resolution (and progressive feature volume down-sampling) that acts as a prerequisite for maintaining low-latency

[373]. In UAV-based visual sensing, where high flying altitudes[1] are preferred as they provide larger ground coverage on aerial imagery, this issue becomes crucial since combined with the wide filed-of-view of UAV cameras it results in objects with very limited spatial resolution. These accuracy limitations are successfully remedied by region-based detectors [374] [4], which however lie in the other end of the pareto frontier being considerably more computationally expensive [370].

In the approach introduced in this chapter, the region-proposal based approach is adopted and combined with a novel data-driven region selection methodology. The proposed model incorporates application-specific domain knowledge and considers additional information provided by the UAV's on-board sensors (altitude, GPS, video etc.), to optimise its efficiency by dynamically discarding a large portion of unnecessary computation at runtime. This approximation is applied on the model layer of the DNN deployment stack, and is agnostic to the target computation platform. As a result, the proposed methodology enables the efficient deployment of state-of-the-art region-based detectors on latency-constrained UAV applications under either a cloud-based or an embedded processing scenario.

The main contributions of this chapter can be summarised by the following:

- A novel region selection methodology for visual detection that evaluates candidate detection regions at runtime, leveraging application-specific information, to provide a latency improvement by dynamically eliminating false positives and their corresponding computational workload at an early stage of the computation, while preserving the detection accuracy.

- The materialisation of the proposed methodology to UAV-based Vehicle Detection, where region proposals are selected based on (i) altitude-aware and (ii) visual criteria; wrapped as a module that can be incorporated on state-of-the-art region-based detection models.

- The creation and release of "CyCAR", a real-world dataset of high-resolution *altitude-stamped* UAV images annotated for vehicle detection in urban environments, capturing a diverse set of traffic scenarios from multiple flying heights.

[1] In the context of this thesis, the terms flying height, altitude and level-above-ground are used interchangeably, referring to the height above ground level (AGL).

5.3 Related Work

5.3.1 Convolutional Object Detectors

CNN-based object detectors, aiming to localise instances from a pre-specified set of object classes in an image, have recently demonstrated significant advancement achieving state-of-the-art accuracy. There are two main categories of convolutional object detectors: *single-shot* and *region-based*.

Single-Shot Object Detectors

Single-shot detectors consist of a feed-forward CNN that directly predicts the class probabilities and locations of multiple objects though a single pass of the input image. Initially the input image is fed though several layers of a Feature Extraction (FE) network, that commonly comprises a widely-used classification pre-trained backbone. Subsequently, the activations of the FE are directly fed to a series of Fully-Connected layers. These are branched deeper in the network to predict class probabilities and box coordinates for different regions of the input image, all in a single pass, leading to low computational complexity. YOLO [192] and SSD [191] form representative examples of this detector class. Although this simple approach has proven effective and efficient on several applications, its accuracy is degraded when considering objects with small spatial resolution [373]. This limitation is attributed to the progressive down-sampling of the feature volume occurring through the layers of the FE backbone, along with the typically smaller input resolution requirement of single-shot detectors which reflects directly to the dimensionality of the FC-layer part and thus their training and inference workload. This restriction, however, forms a major downside for the applicability of SSD models in aerial imagery, where high flying altitudes and large camera field-of-view limits the spatial resolution of objects-of-interest on the ground.

(The Evolution of) Region-based Object Detectors

Region-based detectors, on the other hand, consist of two separate stages. During the first stage, a large number of class-agnostic candidate regions are extracted and forwarded to the second stage that evaluates each region proposal independently to predict class-specific detection probabilities along with refined bounding boxes.

R-CNN [374] is an early paradigm of region-based object detection systems. A set of category-independent candidate detections is initially extracted from the input image through a Selective

Search algorithm [375]. Selective Search detects image regions that are likely to contain an object by performing greedy super-pixel merging on hand-crafted low-level features forming blobs. The corresponding window of each candidate detection (also referred to as region proposal) is cropped from the input image and an affine image wrapping transformation is applied in order to generate a fixed-side square frame, regardless of the proposal's original size and shape. In the original R-CNN, a set of approximately 2,000 such crops are fed to a CNN trained on the task of Image Classification, with the output probabilities of the CNN's prediction for each frame being assigned to a bounding box that forms a fine-tuned version of the corresponding region proposal on the original image, calculated by a trained linear regression model. Although this approach comprises a simple way to bridge the gap between the tasks of object detection and image classification, it involves a three orders of magnitude larger amount of computation for detection due to the multiple forward passes required to run inference each selected crop.

Fast R-CNN [376] partly alleviates the computational burden of running multiple forward passes per frame, by processing the entire input image through the feature extractor of the CNN once, before extracting region proposals from the output feature maps of the FE, based on the candidate detections of the Selective Search algorithm acting on the original image, using a novel Region-of-Interest (RoI) pooling layer. The extracted candidate detections are converted to a fixed-length feature vector and pushed independently through a FC classifier and a 4-value per-class regressor predicting class probabilities and box coordinates respectively, for each candidate detection. Although Fast R-CNN employs a computation sharing approach that improves the computational efficiency and performance of this predecessor R-CNN, it still relies on a computationally demanding external proposal generator, which remains out of the training loop of Fast R-CNN.

In contrast, its next iteration, Faster R-CNN [4], generates class-agnostic candidate detections using a Region-Proposal Network (RPN) at its first stage, which is trained jointly (or in an alternating fashion) with the final box regression and classification tasks in an end-to-end manner. The activations of an intermediate convolutional layer of the Feature Extractor are pushed to the RPN, which predicts a set of category-independent candidate detection boxes (typically 300). These are selected based on their predicted "objectness" probability, out of a large set of refined box priors (called anchors) spread across a wide range of spatial locations, scales and aspect ratios, organised in a regular grid. In the succeeding second stage, each of the selected region proposals is independently pushed through a box predictor consecutively, after its feature volume

is being cropped from the activations of the same intermediate layer. Notably, by exploiting a shared Feature Extractor and being trained end-to-end, Faster-RCNN exploits cross-region computation sharing to improve the computational efficiency of the approach by eliminating duplicate computation on overlapping windows.

Faster R-CNN [4] constitutes the most representative and widely used example of region-based object detectors to date, achieving state-of-the-art accuracy [370] in various well-established datasets [377][190]. Importantly for the target application of this chapter, region-based approaches retain their strong predictive capability when dealing with small scale objects [370]. However, after feature extraction, a large number of candidate regions still require to run through the second-stage classifier independently. As a result, Faster R-CNN's workload approaches 1 TFLOP/frame (depending on the adopted backbone) remaining prohibitively expensive for many latency-sensitive applications in the embedded landscape.

This chapter focuses on region-based detectors, due to their significant advantage in accuracy when considering objects with small spatial resolution. The main aim of this chapter is to adapt region-based detectors, when deployed on a UAV-based scenario, by incorporating application-specific information and prior domain knowledge, in order to improve their efficiency and thus enable their applicability on near real-time applications, such as vehicle detection.

5.3.2 Efficient UAV-based Object Detection

Recent literature has explored the performance-accuracy trade-off on single-shot detectors, towards efficient real-time vehicle detection on embedded platforms. In [114] the model architecture is optimised by exposing the number and size of filters and the input image resolution to a grid search approach. Accordingly, [378] employs a novel automated filter-pruning approach along with careful design choices, to find efficient network designs.

Application-specific customisations have also been proposed for single-shot detectors, utilising prior domain knowledge to adapt the model in a controlled manner that improves efficiency. For example, in [379] a single-shot detector is employed for gate detection in drone racing. Considering the simple geometry of square gates used in drone racing competitions, plenty of the unnecessary high-level feature layers are removed. Moreover, taking into account the maximum distance between the drone and its closest gate, regression operations for detecting small objects are omitted and the default box size range is modified accordingly. Along the

same lines, in [380] a Markov Decision Process is used to adjust all the tunable parameters of an object detection algorithm in order to exploit the accuracy-execution time trade-off for sign recognition on resource-constrained mobile robot platforms.

Although SSD models have almost met the accuracy of region-based approaches for large objects, while providing lower runtimes per image, they demonstrate significantly poorer detection accuracy on small objects [370]. SSD models can only target a fixed-resolution input image, conversely to region-based, while their fast inference speed is conditioned on handling low-resolution inputs [373] (typically between 200x200 and 600x600). Hence, high-resolution input images have to be down-scaled to match the network's receptive field suffering, as a side-effect, a reduction in detection accuracy. Recent research has proposed pushing cropped overlapping tiles of the input image through the CNN independently, without degrading the initial resolution [381][382][383]. However, this approach introduces a considerable computational overhead, which can only be partly alleviated by the use of attention and memory mechanisms for selective tile processing [384].

This shortcoming of single-shot detectors on small-scaled objects is particularly relevant in aerial imagery where objects appear in a vast variety of different scales, accounting to the fact that the spatial resolution of an object is inversely proportional to the flying altitude of the UAV. To address such scale ambiguities, in the work of [385] two built-in sub-networks are incorporated to account for the variability of scales in car detection, acting as ensemble of models trained on disjoint scale ranges, leading however to notable computation overhead.

Additionally, high altitude flights are often preferred in remote sensing applications, as they provide greater area coverage on ground [369]. Recent research has shown that region-based detectors demonstrate remarkable accuracy under such challenging conditions [386], at the expense of increased computational payload. In the work of [370], the speed-accuracy trade-off in convolutional detectors is broadly discussed. Specifically for Faster R-CNN based detectors, it is shown that the adopted number of the RPN's candidate detections has a significant impact in the overall latency that is dominated by the second-stage detector. Thus, reducing the number of region proposals results in a considerable computation saving, to the proportional detriment of detection accuracy[2].

[2] A relevant analysis with similar findings, focusing specifically on the target task of UAV-based vehicle detection, is also discussed in the evaluation section of this chapter.

In the context of this chapter, an informed region selection methodology is introduced and incorporated on Faster R-CNN. The proposed methodology exploits prior domain knowledge to establish application-specific region selection criteria considering information available from other sensing modalities that are available on-board the UAV. This allows to approximate the original model's predictions, while dynamically reducing the number of region proposals for each frame in an input-dependent manner, that controls the impact of this approximation in detection accuracy.

5.3.3 Vehicle Detection Datasets

Existing datasets for top-view detection mostly consist of satellite [387][388], or static surveillance camera [389][390] imagery. UAV-based detection datasets are also usually restricted to specific scenes such as carparks (CARPK [391]) or university campuses (Stanford [392]) or adopt a static hovering UAV position (highd [393]), lacking diversity that would enable their applicability in real-world urban environments. Additionally, many UAV-based datasets incorporate a very limited range of flying altitudes, e.g. below 25m in (UAV123 [394]) and 30m in (AU-AIR [395]). UAVDT [396] and VisDrone2018 [397] comprise the most diverse large-scale UAV benchmarks for detection and tracking to date. However, Visdrone only contains object annotations, while UAVDT labels its frames according to the UAV height, using however only 3 coarse classes: low($<30m$), mid, high($>70m$).

In order to facilitate the development of data-driven approaches such as the one introduced in this chapter, a new benchmark dataset termed "CyCAR" is created and shared with the community. CyCAR is primarily annotated for vehicle detection, however all its frames are altitude-stamped at a much finer granularity (10 meters), while capturing a wide variety of flying heights (20-500m) and traffic scenarios.

5.4 The CyCAR Dataset

This section introduces "CyCAR", a dataset of *altitude-stamped* high-resolution UAV imagery, targeted on the task of ground vehicle detection[3].

[3]The CyCAR dataset has been made publicly available, under an Open Data Commons (ODC) Attribution License, and can be found at https://www.imperial.ac.uk/intelligent-digital-systems/cycar/.
By using a vertical camera pose to the ground, personal information such as license plates has not been captured in any frame, to the best of the author's knowledge.

5.4.1 Context

The data collection process involved real UAV flights above the city of Nicosia in Cyprus, in varying altitudes, with the on-board camera's pose being vertical to the ground. The dataset is *altitude-stamped* based on the flying height estimated through a fusion of the UAV's barometric pressure and GPS sensor readings. A subset of the collected frames is annotated by human experts with tight bounding boxes enclosing all captured cars in the camera's Field-of-View (FoV). The annotated frames span from heavily congested to clear traffic situations. CyCAR comprises a wide variety of scenarios including: (i) *a "static" UAV above an area-of-interest*, (ii) *a moving UAV on a pre-specified path* and (iii) *a moving UAV following a target vehicle on the ground*. These contexts facilitate a comprehensive set of traffic monitoring scenarios, such as persistent monitoring of an area for traffic regulation purposes, periodical data collection for extraction of traffic statistics, and live traffic density estimation in the surrounding area of a moving target (e.g. for assisting emergency vehicle navigation), respectively.

A key feature of the introduced dataset is the broad range of UAV flight altitudes that have been included in the data collection process. With a minimum level of 20m above ground, highly-detailed close-distance images of vehicles and their surrounding environment are captured; whereas the maximum flight-level of 500m above ground provides enormous and challenging coverage of ground area embracing a large number of vehicles and cluttered background in a single frame. A large portion of the captured frames is focused on the range of [50m,130m] above ground, that forms typical flying levels of commercially available UAVs in urban environments [398]. Within that range, the different flying-mission scenarios described above were repeated at a fine granularity of 10m during the data-collection process.

Fig. 5.2 illustrates some representative examples of the collected data for CyCAR.

5.4.2 Data and Annotations

In its current version CyCAR already comprises more than 30 minutes of high-resolution (4K, 2.7K and FullHD) real-flight video (translating to approximately 48,000 frames), captured by a UAV with a camera vertically on the ground. A subset of 500 images containing more than 5,000 vehicle instances being annotated with tight Horizontal Bounding Boxes (HBB), following the annotation format of the Pascal VOC Dataset [377]. Altitude stamps, indicating the flying



Figure 5.2: CyCAR Data Samples captured across a variety of UAV flying altitudes.

height of the UAV at the moment each frame was captured, are also included in the annotation (discretised into 10m segments; a granularity that reflects the data-collection process and was found to be representative of real-world flights, where the UAVs typically maintain a fixed altitude for the largest part of their mission).

CyCAR is the first UAV-based dataset for vehicle detection featuring fine-grained flying altitude annotation (10m). This, appoints CyCAR as a complementary data source for developing and benchmarking data-driven methodologies (e.g. for traffic monitoring applications), such as the altitude-aware vehicle detector introduced in the following section.

5.5 Methodology

5.5.1 Overview

This chapter approaches the challenge of efficient UAV-based vehicle detection *with region-based object detectors*, by introducing an input-dependent **model** layer approximation, exploiting additional information and prior domain knowledge to mitigate unnecessary computation at runtime, while preserving the accuracy of the original model.

The resulting problem addressed in this section can be formulated as follows. *Given*:

- A UAV-based region-based detection model, comprising: (i) a first-stage feature extractor along with a region proposal module, generating a large set of class-agnostic candidate detection windows; and (ii) a second-stage box classifier independently processing each candidate region to provide final predictions on the target task.
- Real-time availability of additional sensor data (i.e. flying altitude measurements) of the UAV that the detector is deployed on.
- A detection dataset of aerial imagery with vehicle annotations and altitude stamps for each frame, such as the one introduced in Sec. 5.4.

the proposed methodology *aims* to:

- Extract prior domain knowledge from the target dataset, in the form of meaningful statistics about the characteristics of true-positive regions proposals, with respect to the detections of the end task, as a function of the additional sensor information.
- Exploit these statistics at runtime, in the form of an application-specific customisation, aiming to detect false-positive detections from the first stage and dynamically alleviate their associated computational burden on the second stage, to allow for proportional performance gains without sacrificing the accuracy of the detection system.

In the context of this chapter, the proposed methodology focuses on the task of Vehicle Detection from UAV imagery, employing different variants of the Faster-RCNN [4] detection model.

5.5.2 Informed UAV-based Region Selection

The processing flow of Faster-RCNN [4] can be summarised by Algorithm 3. After extracting an intermediate feature representation of the input image through a CNN backbone (line 2), region-based object detectors select a fixed-cardinality set of region proposals out of a much larger group of class-agnostic candidate regions (anchors), based on their RPN-predicted probability of *objectness* (line 3). The feature volume corresponding to each selected region is then extracted from the intermediate representation provided by the feature extractor (line 5) and processed independently by a box classifier, that predicts class probabilities and a refined bounding box for each proposal (lines 6-10). To eliminate duplicate detections of the same object that would hurt the overall detection accuracy of the model, greedy Non-Maximum Suppression (NMS) is applied to the final output of the detector, in a per-class manner (line 11). This way, all detected regions that overlap considerably with a higher-scoring detection window of the same class are rejected, based on a tunable Intersection-Over-Union (IOU) threshold.

Algorithm 3 Region-based detector; enhanced with the proposed Region Selection approach.

1: **for** each image $\mathbf{X}^{(k)}$, taken from flying altitude $h^{(k)}$ **do**
2: $\mathbf{F}^{(k)} = FeatureExtractor(\mathbf{X}^{(k)})$ ▷ 1st-stage
3: $P^{(k)} = RegionProposalNet(\mathbf{F}^{(k)})$
4: $\tilde{P}^{(k)} = RegionSelection(P^{(k)}, h^{(k)})$
5: $\mathbf{R}^{(k)} = RoIPooling(\mathbf{F}^{(k)}, \tilde{P}^{(k)})$
6: $O^{(k)} = \{\emptyset\}$ ▷ 2nd-stage
7: **for** each region p $\mathbf{r}$ in $\mathbf{R}^{(k)}$ **do**
8: $[class^{(r)}, box^{(r)}] = BoxClassifier(\mathbf{r})$
9: $O^{(k)}.append([class^{(r)}, box^{(r)}])$
10: **end for**
11: return $NMS(O^{(k)})$
12: **end for**

Although achieving state-of-the-art detection performance, the inflated computation cost of this approach limits its applicability in latency-sensitive applications within the embedded space, with many approaches resorting to SSD-based model variants and bearing the corresponding accuracy degradation [371][372][114]. Aiming to reduce the workload of region-based detectors, their architecture can be tuned to employ a statically reduced the number of region-proposals; incurring,

however, a corresponding accuracy compromise [370]. Simultaneously, in recent literature, the post-processing step of NMS has been enhanced to utilise additional application-specific information, in order to eliminate false-positive detections that harm the model's accuracy, such as bounding boxes with unexpected size, aspect ratio [386][114] and context [399]. However, since the latter methods are only applied on the final output of the predictor, they have no impact on its execution time.

Inspired by these two streams of work, the key idea of the approach proposed in this chapter is to push the identification of false positives towards earlier stages of the detection pipeline. Consequently, this would guide an approximation methodology to dynamically scale the number of proposals at runtime in an informed way, leading to a (variable) degree of approximation at each input frame and corresponding computation savings, while aiming to preserve the detection accuracy at the level of the original model. Specifically, application-specific information arising from the UAV's on-board sensors and prior domain knowledge are injected right after the region-proposal network (line 4 in Alg. 3) in the form of a novel *Region Selection* module. This newly introduced unit is able to perform an *informed and dynamic* selection (online pruning) of the RPN's region proposals, based on application-specific criteria. As a result, only candidate regions that meet the pre-specified criteria are propagated to the computationally-expensive second-stage predictor, reducing effectively the overall workload of the detector through early identification of false positives, while avoiding a compromise in accuracy.

The resulting detection pipeline is illustrated in Fig. 5.3a. The proposed data-driven approach pushes the optimality frontier of the accuracy-latency trade-off, exploiting prior domain knowledge and additional sensor information injected to the system. Specific to the vehicle detection task, the utilised information comprises: (i) *the flying altitude* used to reason about the expected spatial *size*, *shape* (Fig. 5.3c) and *density* of vehicle proposals (Fig. 5.3d) on the input image, given the camera configuration of the UAV and (ii) *computer-vision filters* applied on the input frames to segment regions-of-interest providing priors for the expected spatial location of vehicles (Fig. 5.3e). Both sources contribute on identifying a variable-sized subset of "meaningful" region proposals forwarded to the classifier, in order to be evaluated on the second-stage box predictor.

Nonetheless, although in this chapter the proposed methodology is embodied to the Faster R-CNN pipeline achieving state-of-the-art detection accuracy, any region-based detector can effectively adopt the proposed Region Selection module; with variable performance gains, depending mostly on the underlying workload breakdown between the two detection stages. Furthermore, although

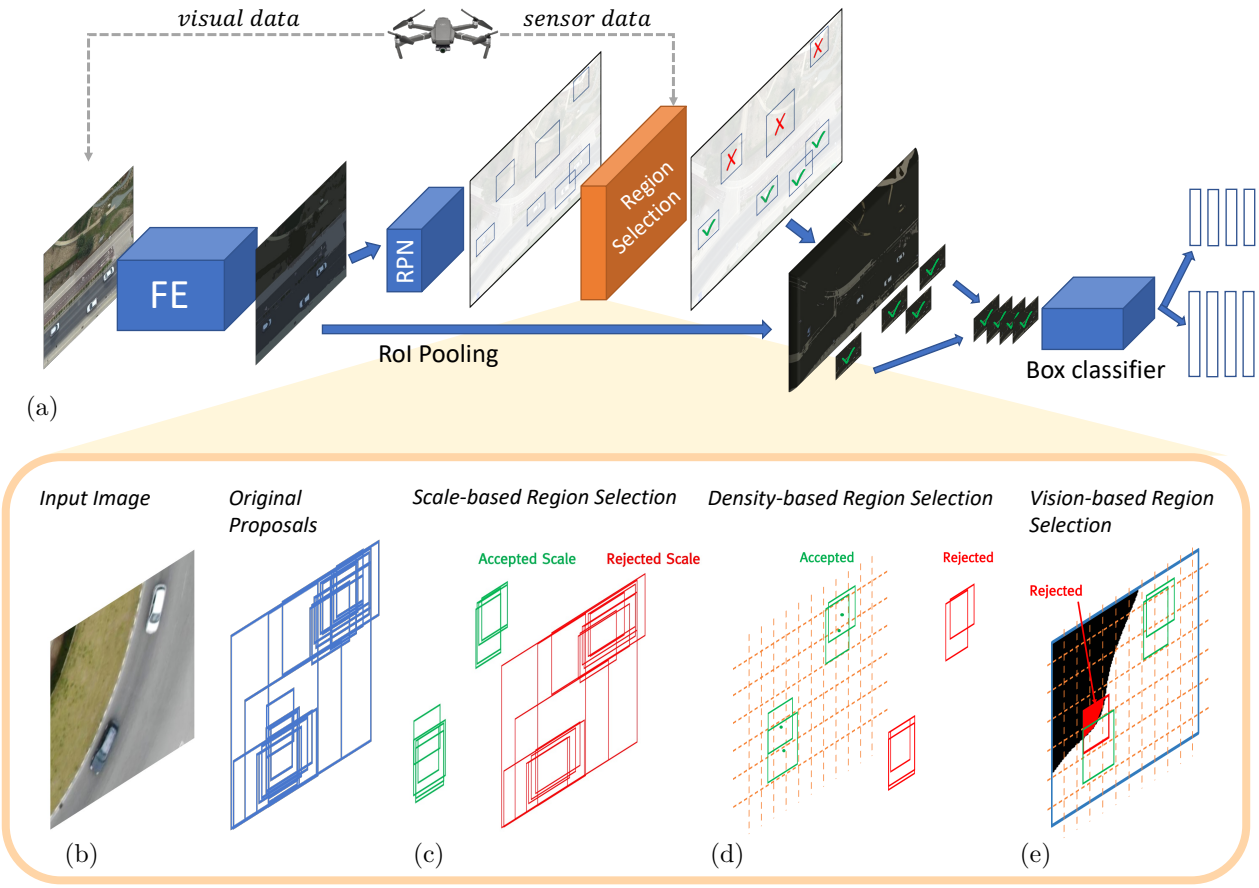


Figure 5.3: The introduced informed region selection methodology, within the Faster R-CNN pipeline. Region proposals are filtered based on sensor data and prior domain knowledge at runtime, resulting to early elimination of false-positive candidate detections and corresponding computation savings.

the remainder of this chapter focuses on the task of UAV-based vehicle detection [400], with appropriate adaptation of the region selection criteria, the proposed approach can be applied on a wide range of UAV-based detection tasks. Thereafter, the region-selection criteria targeted to vehicle detection and implemented within the proposed Region Selection module are described.

5.5.3 Region Selection Criteria for Vehicle Detection

(a) Scale-based Region Selection (Size and Shape)

The first region-selection criterion introduced in the context of UAV-based vehicle detection focuses on the size and shape of the region proposals. Since UAVs are equipped with a camera that has fixed parameters (resolution res=$\{H, W\}$ and Field-of-View fov) and can fly on an outspread of altitudes, cars on the ground are captured on a wide spectrum of different scales,

depending on the UAV's flying level-above-ground. Let the vehicle annotations from the altitude-stamped CyCAR dataset be $\mathbb{D}$:$(\mathbf{X}^{(k)}, h^{(k)}, \mathbb{V}^{(k)})$, where $\mathbf{X}^{(k)} \in [0, 255]^{H \times W \times 3}$ denotes the k-th image, while its associated altitude-stamp is denoted by $h^{(k)} \in \{20, 30, ..., 500\}m$ and its set of vehicle annotations by $\mathbb{V}^{(k)} = \{x_{min}^{k_{(i)}}, y_{min}^{k_{(i)}}, x_{max}^{k_{(i)}}, y_{max}^{k_{(i)}}\}_{i=1}^{N}{}^{(k)}$ where $N^{(k)}$ signifies the number of vehicles in image k. Utilising CyCAR training data, a polynomial model $f_{\mathbb{D}}$ is built in order to extract and abridge the encapsulated domain knowledge, by fitting the anticipated range of scales (size and shape) of vehicle bounding boxes as a function of the UAV flying altitude h, given a set of camera characteristics, in the form of:

$$[scale_{min}^{(h)}, scale_{max}^{(h)}] = f_{\mathbb{D}}(h, res, fov) \tag{5.1}$$

A variety of measures can be considered to quantify the *scale* of a bounding box, including:

$$scale = \begin{cases} area \\ diagonal\ length \\ height \\ width \\ smallSide = min(height, width) \\ largeSide = max(height, width) \\ aspectRatio = largeSide/smallSide \\ \text{any combination of the above} \end{cases} \tag{5.2}$$

also illustrated in Fig. 5.4a. In order to cancel out any irregularities such as data annotation noise, the 95-th percentile of the dataset is considered when determining the minimum and maximum bounds ($scale_{min}^{(h)}$ and $scale_{max}^{(h)}$ respectively) for each height h. Additionally, a tunable error tolerance ϵ is introduced when evaluating candidate regions upon deployment, to robustify the method against false candidate region rejections (e.g. due to sensor noise) that could harm detection accuracy.

At inference time, domain knowledge in the form of the expected scale range for vehicles given the current UAV's flying altitude (obtained by its on-board sensor readings) is injected to the RPN's post-processing. Candidate regions with scale outside of the expected range are rejected *at runtime*, as detailed in Algorithm 4.

As a result, the number of boxes that are pushed through the second-stage classifier is *dynamically* reduced, considering only to those with "meaningful" size/shape for the given altitude (Fig. 5.3c); the cardinality of which may differ significantly between frames. This step essentially eliminates a proportion of false-positive proposals, typically concentrated on

Algorithm 4 Scale-based region selection criterion

function REGIONSELECTION_SCALE($P^{(k)}, h^{(k)}$)
    - - Obtain the expected object-scale range for the current altitude from model. - -
    $scale_{min} = f_{\mathbb{D}}^{min}(h^{(k)}, res, fov)$
    $scale_{max} = f_{\mathbb{D}}^{max}(h^{(k)}, res, fov)$
    $\tilde{P}^{(k)} = \{\emptyset\}$
    for each region proposal $\mathbf{p} := [x_{min}, x_{max}, y_{min}, y_{max}]$ in $P^{(k)}$ **do**
        - - Calculate the scale of the predicted proposal. - -
        $scale^{(\mathbf{p})} = CalculateBoxScale(\mathbf{p})$
        - - Check if proposal scale is within the expected bounds. - -
        if $(scale_{min} - \epsilon) \leq scale^{(\mathbf{p})} \leq (scale_{max} + \epsilon)$ **then**
            - - Only propagate proposals with expected scale to next stage. - -
            $\tilde{P}^{(k)}.append(\mathbf{p})$
        end if
    end for
    return $\tilde{P}^{(k)}$
end function

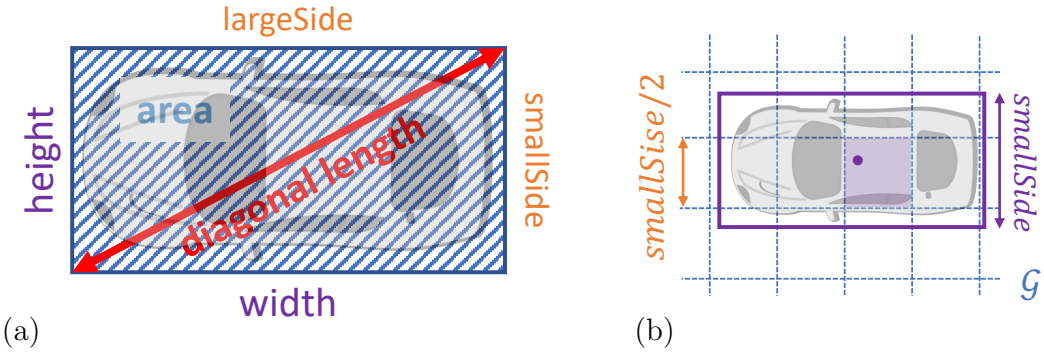


Figure 5.4: Region selection criteria: (a) scale and (b) density.

feature-rich areas of the input image, that attract the attention of the RPN, independently of their scale and existence of vehicles. Simultaneously, the adaptive nature of the proposed approximation approach safeguards the detection accuracy.

Scale refinement for Horizontal Bounding Boxes. The majority of learnable object detectors employ Horizontal Bounding Boxes (HBB), where all predicted boxes are aligned with the image axes, due to their simplicity in annotation and training and wide adoption by deep learning frameworks. However, vehicles in aerial imagery, captured in the wild, have arbitrary orientation. As a result their tightest-enclosure HBBs dissipate more area and feature reduced aspect-ratio compared to axis-aligned boxes captured from the same flying altitude (Fig. 5.5).

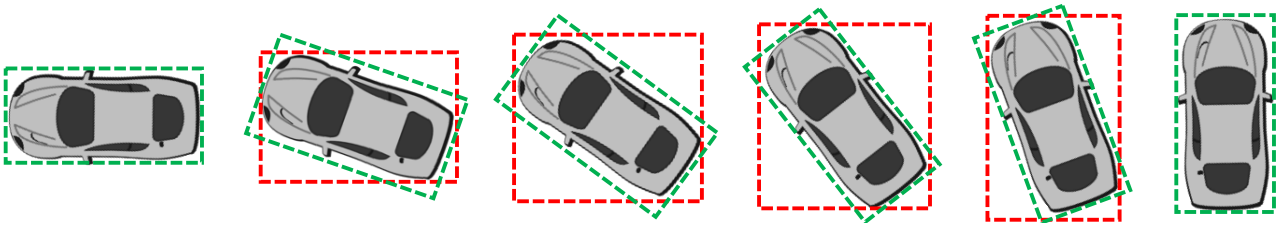


Figure 5.5: Horizontal (red) vs "Essential" (green) bounding box; for a vehicle with different orientations.

This results in extended variance in the discussed metrics for vehicle scale, leading in turn to relaxed thresholds and thus reduced effectiveness of the proposed region selection approach.

To deal with this ambiguity, the mean aspect-ratio $\bar{r}$ of axis-aligned vehicles is extracted from CyCAR dataset $\mathbb{D}$ using the model of Eq. 5.1. Subsequently, an exponential model is fit to estimate the "essential" scale (Fig. 5.5) of each of the RPN's region proposals. This model is predominantly refining region proposals corresponding to axis-unaligned vehicles, which are identified by their deviation from the expected aspect-ratio, expressed as:

$$\widehat{scale}_i = scale_i - (\bar{r} - r_i)^a \cdot b \cdot scale_i \ , \qquad \forall i \in \{1, 2, ..., |P|\} \tag{5.3}$$

where $\widehat{scale}_i$ denotes the inferred *essential scale* for the i^{th} region proposal (out of $|P|$ cardinality set of the RPN's region proposals) with initial scale $scale_i \in \{area, diag, ...\}$ and aspect ratio r_i; where a, b correspond to tunable parameters of the exponential model. This refinement occurs while calculating the scale metrics for each box (line 8, in Alg. 4), to provide a *scale*-estimate of the equivalent HBB given the axis-aligned bounding box originally predicted by the RPN, to be used for region selection by the proposed methodology. This step, although optional, enables to tighten the expected range of scales for a given flying altitude (Eq. 5.1), and consequently further reduce the number of pruned region proposals with corresponding performance gains.

(b) Density-based Region Selection

During the RPN's inference a large set of neighbouring anchors are activated by the existence of each vehicle, leading to multiple partially overlapping proposals. As a result a substantial proportion of the region proposals for an input image are concentrated around specific spatial locations, with many of them demonstrating comparable *scale* (size and shape) and small displacement (Fig. 5.6a). Due to the correctness of their scale, these proposals are not captured by the region selection criterion of Sec. 5.5.3a and could lead to unnecessary computational workload for the second-stage predictor.

As a direct consequence of the spatial dimensionality of objects, their maximum expected density in the original image can also be predicted for a given the UAV flying altitude. This fact is exploited in this section to address the issue described above, through a altitude-aware density-based region selection criterion, that sparsifies the proposals in areas with unexpectedly high density given the UAV flying height.

More specifically, as detailed in Algorithm 5, considering the UAV's altitude h the expected scale of a vehicle's bounding box (line 3), in terms of the lower-bound for the adopted $smallSide^{(h)}$ metric (defined in Sec. 5.5.3a), is re-used from the previous section. A grid $G^{(k)}$ of $smallSide^{(h)}/2$-sized square cells across each input image is constructed (line 4), and every region proposal is assigned to a cell of that grid, based on the coordinates of its centre (line 5-10). This assignment process is also depicted in Fig. 5.4b. Subsequently, the proposed method allows only a single proposal from each cell to be forwarded for classification, considering the RPN's predicted objectness probability for each candidate region to resolve conflicts (lines 11-19).

Algorithm 5 Density-based region selection criterion

1: **function** RegionSelection_density($P^{(k)}, h^{(k)}$)
2: - - Generate a grid of altitude-aware sized cells, aligned to the input frame. - -
3: $gridSize = f_{\mathbb{D}}^{min}(h^{(k)}, res, fov)[min]/2$
4: $G[0 : gridSize : \lceil H/gridSize \rceil,\ 0 : gridSize : \lceil W/gridSize \rceil] = \{\emptyset\}$
5: **for** each region proposal $\mathbf{p} := [x_{min}, x_{max}, y_{min}, y_{max}]$ in $P^{(k)}$ **do**
6: - - Assign each region proposal to a cell of this grid. - -
7: $[x^{(\mathbf{p})}, y^{(p)}] = CalculateBoxCenter(\mathbf{p})$
8: $[g_x, g_y] = AssignToCell(G, [x^{(\mathbf{p})}, y^{(\mathbf{p})}])$
9: $G[g_x, g_y].append(\mathbf{p})$
10: **end for**
11: $\tilde{P}^{(k)} = \{\emptyset\}$
12: **for** each cell g in G **do**
13: - - Only preserve a single proposal per cell (with the highest objectness probability). - -
14: **if** $|g| > 1$ **then**
15: $g = \arg\max_{r \in g} probability(r)$
16: **end if**
17: $\tilde{P}^{(k)}$.append($P^{(k)}[g]$)
18: **end for**
19: return $\tilde{P}^{(k)}$
20: **end function**

(a)



(b)

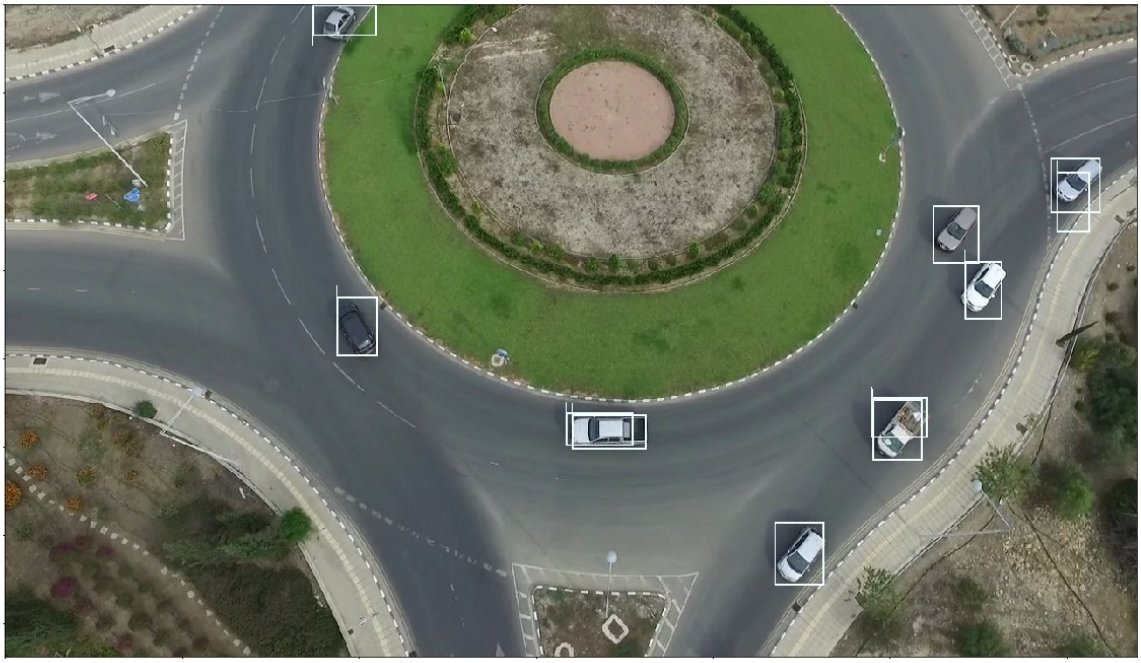


Figure 5.6: (a): Region proposals of the original Faster-RCNN. (b): Output of the proposed informed region selection methodology; on a sample frame of *CyCAR* dataset.

This is equivalent to parsing the input image with a sliding window of size $smallSide^{(h)}$ and stride $smallSide^{(h)}/2$ (Nyquist Sampling Rate), to extract a single proposal per step (Fig. 5.3d).

Notably, the above criteria complement each other, aiming to identify and prune false-positive detections via different angles, in an altitude-aware manner. As a result of the joint application of these two criteria, in the case of the previously discussed example, a significantly reduced, but robust set of meaningful proposals is preserved and propagated to the second stage (Fig. 5.6b).

(c) Vision-based Region Selection (Spatial Location)

The RPN is class-agnostic and its attention can be attracted by feature-rich areas of the input image that are unrelated to vehicles. This can become a more prominent source of inefficiency (or optimisation opportunity) in the case of high-altitude flights, were often only a small portion of the input image corresponds to areas of interest for the target detection application (i.e. road segments; Fig. 5.7b). A domain-specific optimisation targeted to the task of traffic monitoring can be used to alleviate further unnecessary computation, by discarding any proposals on areas of the image that do not correspond to road segments.

For this purpose, a light-weight computer vision filter is employed to identify regions-of-interest on the input image, with minimum computational overhead. Specifically, inspired by [399], simple HSV thresholding is used on each pixel to identify road segments. The output of this filter is post-processed by a smoothing window-median filter to eliminate false negative pixels introduced by occlusion, such as vehicles on the road, yielding a road segmentation mask $M^{(k)}$. More advanced road-segmentation techniques from the literature can also be applied at this stage [401], offering potentially higher segmentation quality. However, it was found that in the context of the efficiency optimisation proposed here where the computational overhead of the selection criteria is critically considered, this approach is sufficiently accurate (qualitatively) as with appropriate tuning it rarely suffers from false-negatives that could penalise detection accuracy (Fig. 5.7), while being immensely computationally efficient (featuring 3 orders of magnitude lower workload compared to learnable counterparts [402]) due to its simplicity.

These regions are then projected to the image cells of the altitude-specific grid as $G^{(k)} \odot M^{(k)}$ (replacing G at line 12 in Alg. 5), based on a minimum coverage threshold. This adds a new dimension to the region selection methodology by only considering proposals assigned on the subset of cells that have been identified as regions-of-interest (Fig. 5.3d). This approach discards a greater

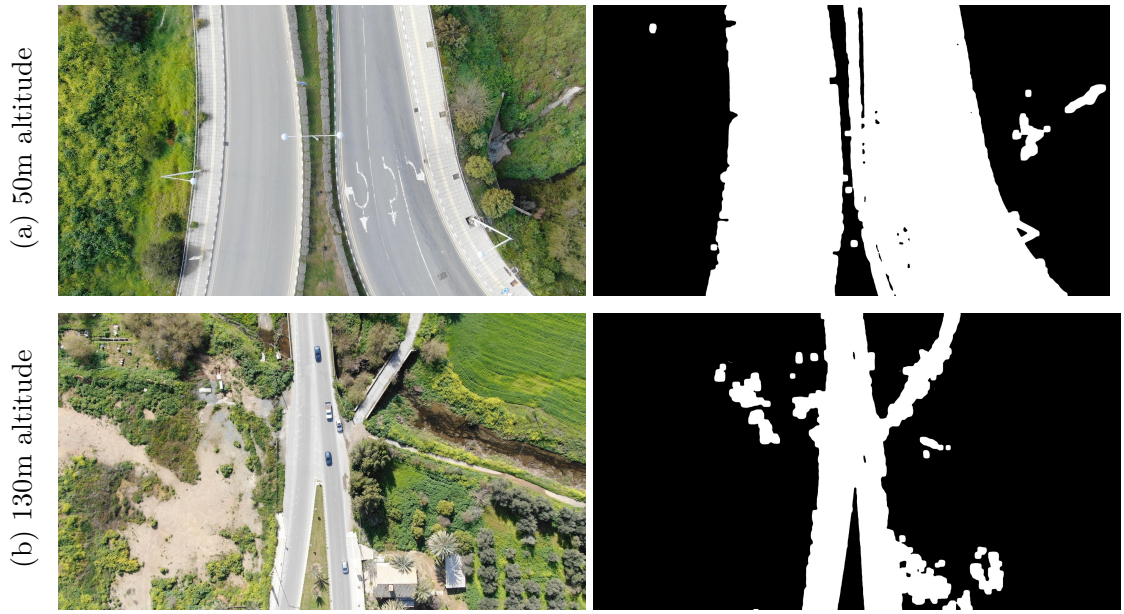


Figure 5.7: Output of HSV-based road segmentation[4] on CyCAR images from different altitudes.

expanse of object proposals, by incorporating prior knowledge regarding their spatial location's context on the input image, through light-weight pre-processing of the UAV's visual sensor inputs.

It is noteworthy that, although in the context of this chapter this criterion was built on top of (and performed in conjunction with) the approach of Sec. 5.5.3b, it can also be instantiated and used independently.

5.5.4 GPU Deployment Aspects

By combining the aforementioned criteria following the most implementation-friendly sequence (scale → vision → density), an informed application-specific region selection methodology is established, based on which a candidate region is forwarded to the box classifier if it simultaneously:

- features the expected *scale*, given the flying altitude,
- is the highest-scoring region proposal on its *density* cell,
- belongs to a cell that corresponds to a *region-of-interest (i.e. road segment)*.

All the above criteria are carefully designed to be simple and parallelisable in order to exploit the Single-Instruction-Multiple-Data (SIMD) processing paradigm of Graphic Processing Units (GPU). Their underlying implementation is also ported in the tensor processing pipeline to avoid any data transfers back to the main memory. This is achieved by wrapping the proposed criteria as

an additional module (layer) that is part of the model's computational graph, but only activated at inference time (i.e. adopting a Straight-Through Estimator for gradient during the backward pass). As a result, the proposed Region Selection module has negligible impact on inference latency, measured to be less than 1.5% of the original Faster-RCNN implementation.

5.6 Evaluation

5.6.1 Experimental Setup

In this section, the experimental evaluation of the proposed methodology is discussed. Experiments have been conducted emulating two different settings: *(i) On-board embedded processing* where all the computations are performed on-board the resource-constrained UAV platform, on an embedded GPU device (for real-time or privacy-aware applications, and/or deployment in network-restricted areas); *(ii) Cloud-based high-end processing* in which the MAV transmits the captured video stream along with other sensor data to a remote on-the-ground server, located at a base station, for processing on a high-end GPU (when near-real time processing is required and good network conditions exist; or a low-cost UAVs without AI-capable compute units is adopted). The server is equipped with an Intel Xeon E5-2630 CPU, 64GB RAM and a 2560 CUDA-Core Nvidia GTX1080 GPU (Pascal Architecture). For the embedded setting an Nvidia Jetson TX2 board featuring a 256 CUDA-Core embedded GPU (Pascal Architecture), 8GB of RAM and an Quad ARM A57 CPU is used. The models have been developed and trained using the Object Detection API [370] of Tensorflow v1.12 on the same server; the deployment on the TX2 platform resorted to Tensorflow v1.09. The UAV used in the experiments is a DJI MAVIC 2 Enterprise, equipped with a high-definition camera with Field-of-View of 82.6^o.

To facilitate fair comparisons, the concept of meta-architectures, introduced in [370], is employed across the experiments. Meta-architectures provide a level of abstraction, outlining the algorithmic body of each detector family (SSD, RPN etc), while decoupling it from its original implementation; i.e. allowing any backbone CNN model to be used for feature extraction. This allows ImageNet [21] pretrained classification backbones to form the basis for the Feature-Extractor part of commonplace detectors, the detection heads of which are appended and trained jointly with the backbone on large scale detection datasets, such as MS-COCO [190].

[4]For road segmentation, the following formula was found most effective across the examined dataset, after manual tuning: $Hue < 90$, $Saturation < 5$, $60 < Value < 110$.

5.6.2 Speed/Accuracy Trade-offs in Object Detectors

Single-Shot vs Region-based Detectors in UAV Imagery

Initially, a comparison between off-the-shelf single-shot (SSD [191]) and region-based (Faster R-CNN [4]) detectors on aerial imagery is performed. The aim of this experiment is to establish strong baselines for the proposed methodology on the task at hand and further motivate the need for more efficient region-based detectors in UAV/Aerial imagery. A series of MS-COCO pretrained models that have been identified as "key-points" on the performance-accuracy optimality frontier are selected, featuring different choices of backbone CNN architectures, namely MobileNet-V2 [182], ResNet-50 [3] and Inception-ResNet-V2 [180]. All selected models are initially trained on the DOTA multi-class dataset of aerial imagery [383] until convergence, and evaluated independently. Subsequently, fine-tuning and evaluation on the newly introduced CyCAR Dataset (Sec. 5.4) is performed. Various data augmentation techniques have been used on both datasets, including: random horizontal flipping, image re-scaling and adjustment of contrast and brightness. The PascalVOC-established metric of mean-Average Precision (mAP) [377] is reported across the Validation Set of each dataset, summarising the shape of the precision-recall curve, defined as the mean precision at a set of eleven equally-spaced recall levels in the range of [0,1].

Table 5.1 summarises the results of this comparison. As expected, the single-shot architecture based on MobileNet-V2, achieves the lowest latency both on the high-end and the embedded device. This comes at the expense of a significant accuracy compromise of approximately 24 and 30 percentage points (p.p.) in the validation sets of DOTA and CyCAR dataset respectively, compared to the highly-accurate Faster R-CNN-based detector (built on top of Inception-ResNet-V2). This can be attributed to the restricted input resolution and progressive down-sampling of the feature volume taking place in SDD detectors. These limitations of SSD are particularly penalising in terms of detection accuracy in the domain of aerial imagery, where a large portion of objects frequently appear with limited spatial resolution, and information is consequently dissipated before reaching the last layers of the model. This hypothesis is experimentally affirmed by the fact that choosing a semantically stronger (and 2.45× more computationally expensive) FE backbone for SSD (ResNet-50) demonstrates very limited impact on detection accuracy. The more accurate region-based detector, however, although achieving state-of-the-art detection results, requires over

Table 5.1: Latency/Accuracy Trade-off of Pareto Optimal Detectors

Detector Model		Performance (Latency)		Accuracy (mAP)	
Meta-Arch	**Feature Extractor**	*GTX1080*	*TX2*	*DOTA*	*CyCAR*
SSD	MobileNet-V2	36.31ms	150.81ms	36.23%	46.63%
SSD	ResNet-50	89.02ms	380.26ms	38.43%	49.99%
Faster R-CNN	ResNet-50	143.28ms	1160.60ms	50.56%	64.82%
Faster R-CNN	Inception-ResNet-V2	562.65ms	- ms	60.22%	76.31%

15x more computation time on the high-end GPU (achieving up to 1.77 fps) compared to the SDD model, while the embedded platform's memory resources could not accommodate its execution.

An intermediate design point in the pareto frontier is provided by a Faster R-CNN detector employing the ResNet-50 as its feature extractor. Compared to it's Incpetion-ResNet-V2 based counterpart, this model offers a balanced trade-off between performance and accuracy, being 3.9× faster and thus able to reach up to 7fps during inference. Although suffering a 10.5p.p. accuracy drop compared to its Incpetion-ResNet-V2 counterpart, the ResNet-50 based Faster R-CNN demonstrates a significantly improved accuracy (up to 15p.p) to that of ResNet-50 based SSD model. In spite of sharing the same backbone FE, this improvement comes at the expense of Faster R-CNN being 1.6 times slower latency-wise, attributed to the computational overhead imposed by the second-stage box predictor, along with its increased input resolution; while this latency gap is further aggravated (up to 3.9x) when compared to the most light-weight MobileNet-V2 SSD variant, with disproportional impact on accuracy, as discussed above. Nonetheless, the attainable 0.87 fps processing rate on the embedded platform, induces a substantial frame-drop that, when deployed on downstream tasks such as traffic monitoring, can considerably affect the application-level QoR through the introduction of false-negative detections at the system level (i.e. vehicles that were captured only in a handful of frames that did not get to be processed by the model).

Bridging the gap

In order to improve the accuracy of SSD, recent work resorted to performing multiple separate runs covering different windows of the input image [382]. However, in order to reach a comparable accuracy to Faster R-CNN in the target UAV-based detection task, using a 4x4 grid of overlapping windows was, indicatively, required. This resulted to a prohibitive increase in workload (ending up being 10× slower that Faster R-CNN), despite using the same backbone.

The methodology proposed in this chapter, seeks to bridge the gap between SSD and region-based detectors from an opposite perspective. Instead of aiming to improve the accuracy of SSD, it targets optimise the performance efficiency of Faster R-CNN. Tuning the number of region proposals that propagate to the second-stage box classifier is employed as the main mechanism for reducing the workload of Faster R-CNN.

Figure 5.8 demonstrates the speed-accuracy trade-off that arises by imposing a fixed (off-line tuned) constraint on the number of region proposals. Although pronounced performance gains can be achieved by reducing the region proposal cardinality, it is evident that when a naive probability-based criterion is used during the selection/pruning of candidate detections, a significant accuracy drop is provoked. More specifically, when an excessively low number of proposals is adopted, the accuracy is negatively affected as the model is natively not able to deal with inputs with large number of objects. More importantly, even with a larger number of proposals, the probabilistic region-selection methodology of the original detectors is still inducing an accuracy loss, by prioritising multiple proposals for the most pronounced objects in each frame, while leaving more ambiguous cases completely undetected.

Additionally, it is noteworthy that the performance does not scale directly with the number of proposals, across the examined range. This is attributed to two reasons: (i) the underutilisation of the target device (due to lost batching/parallelisation potential) on the extremely low end of the proposal spectrum; and (ii) the computational overhead of the shared feature extractor module, accounting for 47% and 34% of the workload of the original detectors (statically employing 300 proposals), based on the Inception-ResNet-V2 and ResNet-50 architecture respectively.

Instead of employing a constant (reduced) number of proposals, the proposed methodology utilises UAV sensor data and prior domain knowledge to dynamically determine which proposals should be kept/discarded at runtime, in an informed way. As a result, the degree of approximation is automatically tuned for each input frame, maximising the attainable performance gains while maintaining the accuracy of the original model.

5.6.3 End-to-end Evaluation

In this section, the proposed region selection methodology is evaluated in an end-to-end manner, capturing its impact on inference speed and detection accuracy. For completeness, the proposed methodology is applied to both the most-accurate Inception-ResNet-V2 based Faster R-CNN

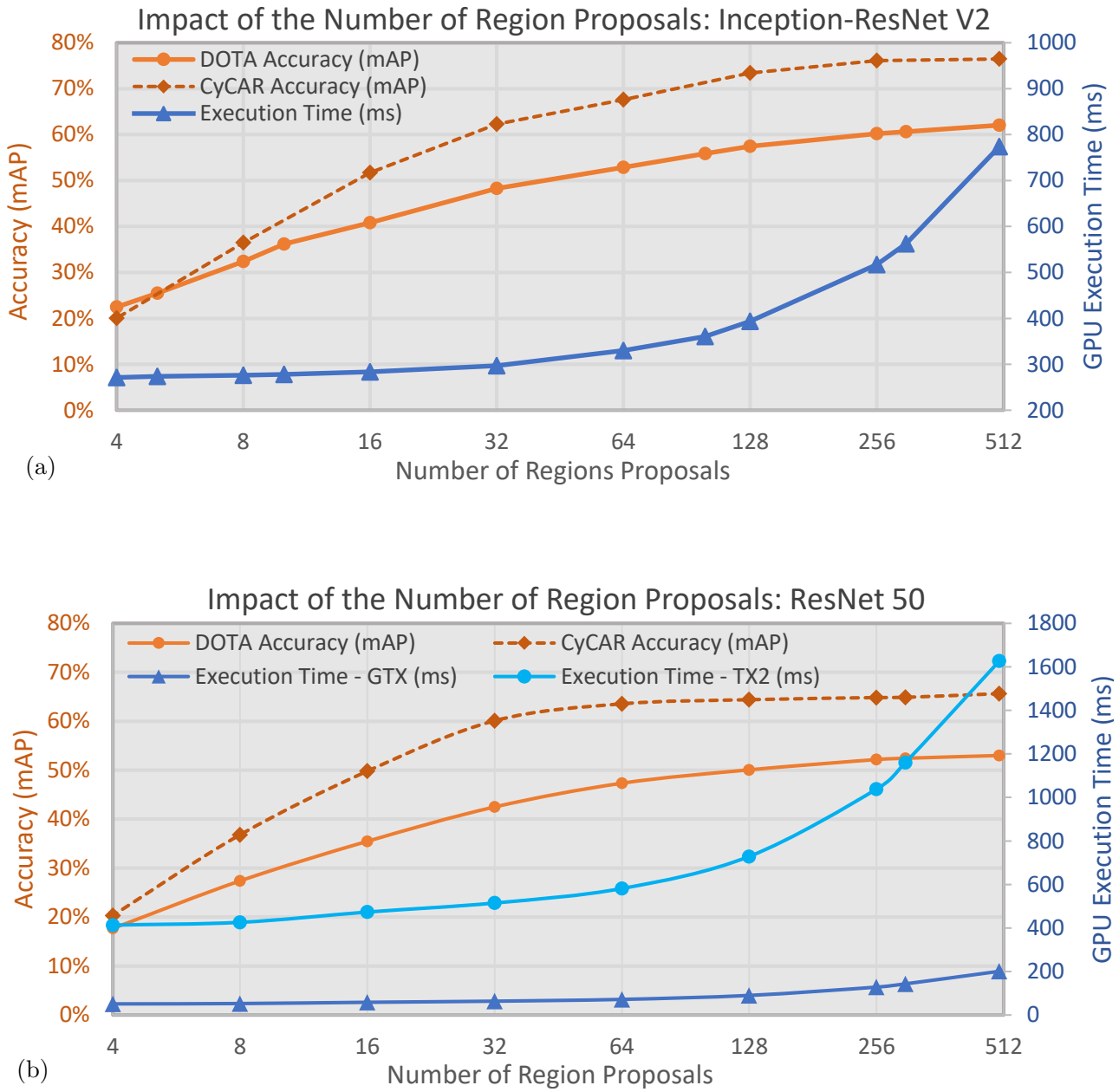


Figure 5.8: Latency-accuracy trade-off in (a) Inception-ResNet-V2 based and (b) ResNet-50 based Faster R-CNN; as a function of the number of region proposals being evaluated on the second-stage.

instance (suitable for the cloud-processing scenario), as well as the ResNet50-based variant which was employed on all experiments on-board the UAV.

The results of Table 5.2 showcase the effectiveness of the proposed approach in the task of vehicle detection. Detection accuracy ($=\frac{TP}{TP+FP+FN}$), along with its constitutive components (absolute number of True-Positive (TP), False-Positive (FP) and False-Negative (FN) detections) are reported to concede a more interpretable understanding of the impact of the proposed approximation. Accuracy is measured on a previously unseen test-set of the CyCAR dataset with a wide variety of UAV flying altitudes, consisting of 30 images capturing a total of 207 cars. In terms of prediction speed, end-to-end inference latency is measured on both devices for each sample of the test-set, employing a batch-size of one; reporting the average latency per frame.

As a result of exploiting additional flying altitude and road segmentation information, the effectiveness of the proposed region selection approach is demonstrated by a remarkable reduction (27x geo. mean) in the number of region proposals propagating to the second stage, across backbones, while successfully preserving the accuracy of the respective original models (within 1 p.p.).

The proposed approach is also compared with two *oracle* baselines that employ a reduced but *fixed* number of proposals, retrospectively hand-tuned to match the achieved accuracy and latency of the proposed methodology respectively, *in view of the specific test-set*. The results of Table 5.2 suggest that the proposed approach significantly overperforms both baselines in their respective metric-of-interest, demonstrating the impact of dynamically scaling of the number of proposals in an input-dependent manner on inference efficiency. It is noteworthy that although oracle baselines select the *ideal* number of fixed-cardinality proposals tailored to the task and data at hand (i.e. minimum adequate number of proposals), their effectiveness cannot be guaranteed on unseen data (i.e. with higher population of vehicles), harming their deployability in the real-world.

More specifically, in the case of the most computationally expensive Inception-ResNet-V2-based model, the original implementation extracting 300 candidate regions per frame, achieves high accuracy (90.05%) by detecting 190 out of 207 cars and suffering four false-positive detections, at a consistent latency of 562.65ms per image (1.77fps). The proposed methodology achieves a similar accuracy of 89.90% by detecting 187/207 cars, also reducing the number of false positives, while requiring more than 2x less computation time (3.6fps) on average. Experiments also indicate that the oracle baseline employing a fixed reduced number of proposals (being hand-tuned aware of the expected output), requires 4.75x more proposals to achieve similar

Table 5.2: Comparison of the Proposed Approach with Baselines

Approach	NumProposals	Latency (GTX/TX2)	Accuracy	TP	FP	FN
Inception-ResNet-V2						
Original Model	300	562.65ms / -	90.05%	190	4	17
This Work	(4 - 40)	278.41ms / -	89.90%	187	1	20
⊙*Matching Accuracy*	190	462.14ms / -	89.05%	187	3	20
⊙*Matching Latency*	10	278.01ms / -	63.64%	133	2	74
ResNet50						
Original Model	300	143.28ms / 1160.6ms	79.72%	169	5	38
This Work	(3 - 54)	54.11ms / 439.7ms	78.95%	165	2	42
⊙*Matching Accuracy*	210	110.40ms / 904.2ms	78.20%	165	4	42
⊙*Matching Latency*	10	54.28ms / 440.0ms	69.19%	146	4	61

⊙ denotes an *oracle baseline*, optimised directly on the metric-of-interest, in view of the test data.

accuracy to the proposed approach, leading to a 66% increase in computation time. Accordingly, a baseline oracle adopting a constant number of proposals hand-tuned to achieve the same average latency as the proposed model could only reach an accuracy of 63.64% (translating to an increase of 3.7x in missed detections (FN)).

Applying the proposed methodology to the ResNet-50-based detector provides a speed-up of 2.64x on the high-end GPU achieving a frame-rate of 18.5fps compared to the rate of 7fps of the original model, while the accuracy is preserved within 1p.p. This result is also surpassing the ResNet-50 based SSD baseline of Table 5.1 by 1.64x, dominating the pareto frontier via a region-base model. Near real-time processing (reaching 2.3fps) is also enabled by the proposed approach on the embedded platform, being 2.67x faster than the original model, that achieved 0.86fps.

5.6.4 Ablation on Region Selection Criteria

In this section, an ablation study between the three proposed region selection criteria is presented. The conducted experiments indicate that the contribution of each criterion varies severely between frames. Hence this section reports the average percentage of pruned region proposals eliminated by each criterion, when applied in a pipelined fashion (scale → vision → density) as in the proposed approach across the test set. The results, summarised in Table 5.3, follow a categorisation of input frames to *low* and *high altitude*, as this feature seemed to predominantly affect the variance.

For frames captured by the UAV in a high altitude (above 70m), it can be seen that a significantly higher proportion of region proposal rejections is conducted by the vision-based

criterion (road segmentation), in contrast to the low altitude frames (on or below 70m) where the contribution of this criterion is limited. This can be attributed to the native bias of the data collection process aiming to keep the UAV above the urban road network, which led to road segments dominating (pixel-wise) most low-altitude frames (also evident in Fig. 5.2).

Scale and density have a balanced contribution in high altitude frames, jointly accounting for a significant 59% of the proposal rejections. In low altitude flights, the contribution of density is notably uplifted, as a consequence of the larger resolution of actual vehicles leading to larger proposals (and grid cells; Sec. 5.5.3) and in-turn greater overlap between candidate boxes.

Conclusively, all three criteria play a significant and complementary to each other role in the region selection process of the proposed methodology.

Table 5.3: Percentage of region proposal rejections, with each region selection criterion.

	Scale (Sec. 5.5.3a)	**Density** (Sec. 5.5.3b)	**Vision** (Sec. 5.5.3c)
Low Altitude (≤70)	37.2%	48.6%	14.2%
High Altitude (>70m)	30.9%	28.1%	41.0%

5.6.5 Qualitative Analysis

Lastly, this section discusses a few insights obtained through a qualitative analysis of the experimental results. The quantitative analysis of Sec. 5.6.3 indicates a consistent reduction in false-positive (FP) detections by the proposed approach, compared to the original model. This is mostly accounted to duplicate detections of the original model that were not captured by the NMS post-processing owing to a large difference in their relative spatial resolution (e.g. Fig. 5.9a), and were successfully eliminated by the scale and density criteria of the proposed methodology.

On the other end, it has been noticed that aggressive thresholding on the expected scale of vehicles for a particular flying altitude may increase the number of false-negatives (FN), when outliers are present in the input image. For example, in the case of Fig. 5.9b, a light truck with larger scale than the rest of the vehicles in the frame, was not detected by the proposed approach. Adopting a less tight error tolerance ϵ in the expected vehicle scale for each altitude successfully resolves this issue, with marginal effect on the computational workload.

Finally, it is noteworthy that the oracle baseline employing a fixed reduced number of region proposals requires a much larger-cardinality of candidate regions in order to match the accuracy

of the proposed approach. This is accounted to two reasons: (i) Some frames, especially on high-altitude flights or above congested areas, capture a large number of vehicles easily surpassing small pre-specified threshold values in proposals' number; (ii) Even in less congested frames, the original method retains a fixed-cardinality set of proposals that achieve the highest "objectness" probability values. Frequently, when a small number of proposals is selected, multiple candidate detections of the same vehicle (in different scales and aspect ratios) may surpass the objectness probability of vehicles in more challenging regions of the input image (such as shadowed areas). This may result to accuracy degradation, even when a surplus of proposals exist with respect to the number of vehicles actually evident in the frame (e.g. Fig. 5.9c). The proposed approach, effectively handles the above cases (as illustrated in Fig. 5.6b), by jointly considering the spatial resolution, density and location of candidate regions in the image, to dynamically adjust the number of proposals for every frame at runtime.

5.7 Discussion

This chapter focused on application-specific approximations applied on the model level of the deep learning deployment stack, in view of prior domain knowledge, trading a generic but slower model to a faster but more specialised one. Due to the structure of the proposed approximations, their computational savings were able to directly translate to latency speed ups when deployed on off-the-shelf general purpose GPUs.

Having shown that the processing flow of region-based detector is best suited for detection tasks on UAV/Aerial imagery, where objects-of-interest appear in a limited spatial resolution, a novel methodology was proposed pushing the limits of efficient deployment of two-stage detectors in latency-sensitive applications.

The proposed methodology comprises a Region Selection module, injected half-way through the detection pipeline, aiming to evaluate candidate region proposals and identify false-positive detections early on, to terminate their processing. The region selection criteria take advantage of application-specific sensor cues available on the UAV and light-weight computer vision filters, along with prior domain knowledge, essentially specialising the model to the task at hand.

Particularly for the task of UAV-based vehicle detection, altitude-aware information about the expected scale, density and spatial location of candidate vehicle detections on the ground is exploited at runtime, to eliminate outlier detections at an early-stage, reducing the overall workload.

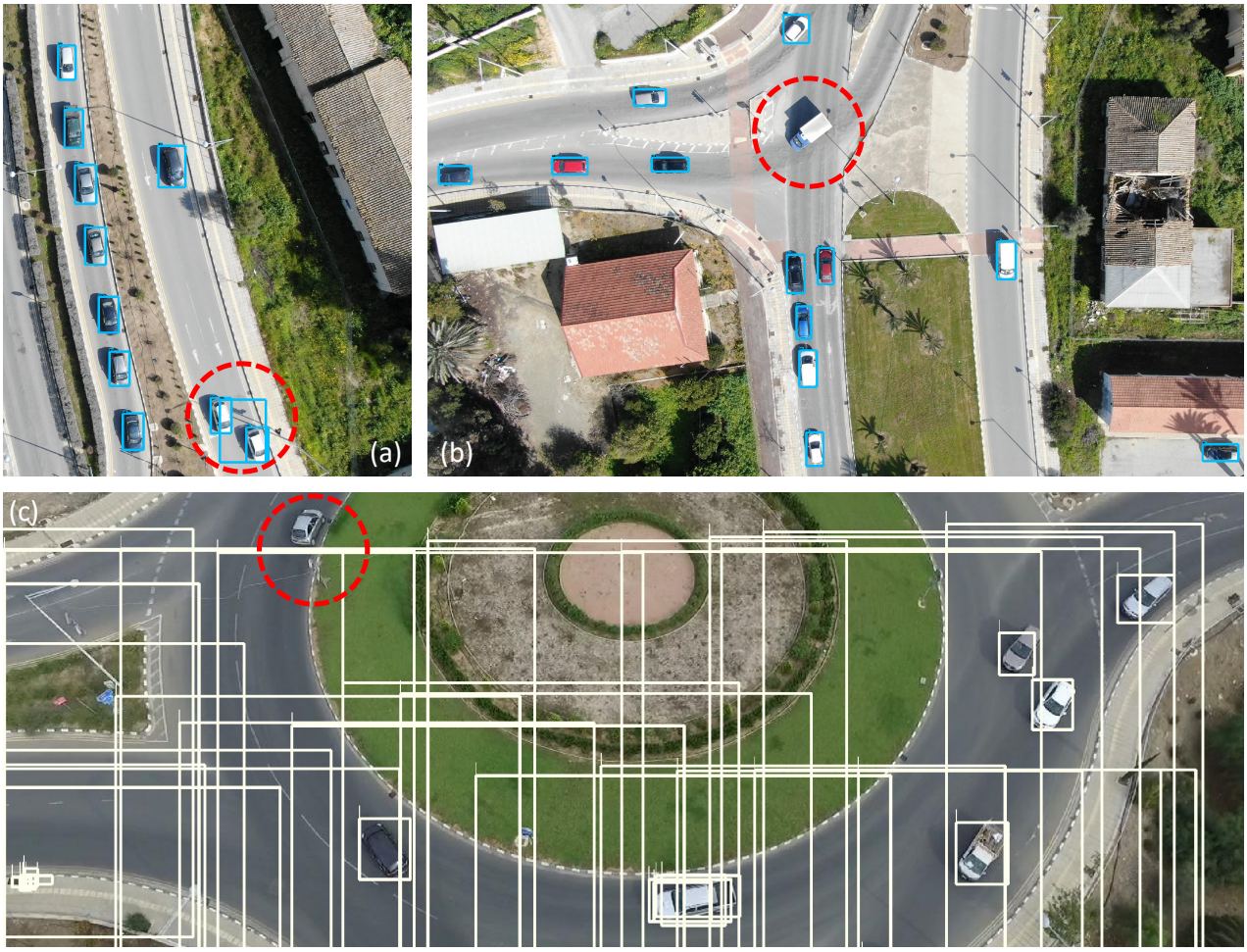
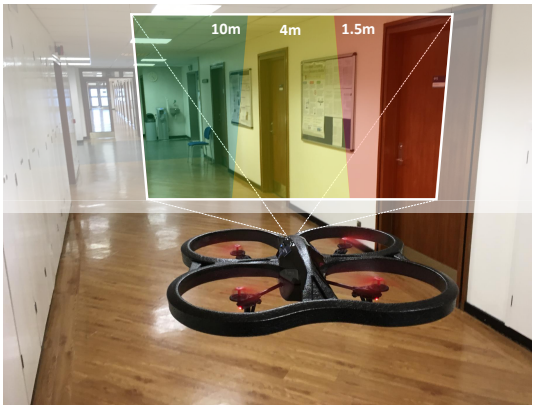


Figure 5.9: Qualitative results: (a) A false-positive detection of the original model, corrected by the proposed methodology. (b) A false-negative detection of the proposed methodology, caused by overly strict thresholding of the expected vehicle scale. (c) False-negative region proposal from a baseline, performing probability-based region selection with a fixed cardinality of 50 proposals.

By employing a dynamic (runtime-adjustable) number of region proposals for each image, the proposed methodology pushes the latency-accuracy optimality frontier by achieving remarkable speed-ups, with little to no compromise in accuracy; enabling the use of state-of-the-art region-based detectors on-board UAVs.

Alongside, a new dataset of real UAV imagery annotated for the task of vehicle detection and altitude-stamped to support the development of data-driven methods is also introduced and made available to the community, featuring a variety of flying heights and traffic scenarios.



6

Learning to Fly by Myself:

A Self-Supervised CNN-based approach for autonomous UAV navigation

Contents

6.1 Overview . . . 164
6.2 Motivation . . . 165
6.3 Related Work . . . 168
6.3.1 Data Collection and Annotation for UAV Navigation . . . 168
6.3.2 DNN-based Approaches for UAV Navigation . . . 169
6.4 Methodology . . . 172
6.4.1 Overview . . . 172
6.4.2 Self-Supervised Data Collection and Annotation . . . 172
6.4.3 The *SelfNav* Dataset . . . 175
6.4.4 CNN Architecture . . . 177
6.4.5 Model Training and Optimisations . . . 180
6.4.6 Local Motion Planner . . . 182
6.5 Evaluation . . . 185
6.5.1 Experimental Setup . . . 185
6.5.2 Model Design and Prediction Accuracy . . . 186
6.5.3 Training- and Inference-time Optimisation Ablations . . . 189
6.5.4 End-to-end UAV Navigation Comparison . . . 191
6.5.5 Qualitative Analysis . . . 192
6.6 Discussion . . . 193

Keywords: *Two-stream CNN, Learnable Spatio-temporal Representations, Self-Supervision, Pre-training, UAVs, Autonomous Navigation, Obstacle Avoidance, SelfNav Dataset*

6.1 Overview

*The focus of this chapter shifts up to the **data layer** of the deep learning deployment stack. The release of large-scale annotated datasets has been a driving force for the emergence of deep learning in several computer vision applications. In robot vision tasks however, the data collection and annotation process can be prohibitively complex, time consuming and expensive. As a result, the scale of machine vision datasets to date is significantly smaller, also featuring sparse annotations; thus limiting the pace of adoption of data-driven approaches across the field. At the same time, complex data modalities such as optical flow, encapsulating spatio-temporal visual information, have proven to be particularly useful in vision-based applications where a continuous stream of camera frames is available, as is common in robotics. However, the excessive computational cost required to explicitly calculate such data cues at runtime restricts their adoption in the resource-constrained and latency-sensitive settings as in several mobile robot applications, where such information could be exploited by DNNs to enhance the descriptiveness of their feature representations.*

This chapter preserves the direction of application-specific optimisation, focusing on the task of autonomous navigation. Learnable navigation approaches can act as an enabler for the safe deployment of mobile robots in a wide range of real-world environments, due to their enhanced generalisation potential to previously unseen domains; under the assumption that a vast amount of training data is available. A self-supervised approach for collecting and annotating robot navigation data, with minimum human intervention through the use of additional distance sensors is initially introduced in this chapter. As a result a new real-flight dataset for autonomous navigation termed SelfNav is created. Subsequently, a CNN-based regression model is designed and trained to approximate these sensor readings directly from monocular visual inputs. Being specially trained to simultaneously process the current and previous input frames, the proposed model is able to implicitly extract and exploit spatio-temporal features that encapsulate both spatial appearance and temporal motion information; resulting to a considerable computational cost reduction and comparable effectiveness to equipping the model with explicitly calculated spatio-temporal cues from the literature.

Employing the proposed model on a UAV, along with a custom motion planner able to exploit the fine-granularity of its predictions to modulate the yaw and linear velocity of the UAV, demonstrated the effectiveness of the proposed approach in autonomous exploration within real-world indoor environments; offering numerous advantages compared to relevant approaches.

This chapter is based on a conference paper co-authored with Prof. Christos Bouganis: (IROS 2018).

6.2 Motivation

Obstacle detection and avoidance form core tasks towards the safe deployment of mobile robots, navigating autonomously in real-world environments. In this direction, several autonomous navigation methods have been proposed in the literature, relying on sophisticated sensors such as RGB-D [403] and LIDAR [404], and complex algorithms such as SLAM [405], to construct a 3D model of the environment. Such information is typically post-processed by subsequent hand-engineered motion planning modules to deduce the navigable space and yield corresponding action commands, closing the sense-process-act loop through a *modular* pipeline.

However, on more resource-constrained platforms like Micro Aerial Vehicles (MAVs), the use of such sensors is not practicable, due to their limited form factor and payload constraints. Recently deep learning has been able to provide estimates of dense depth information in view of monocular images [406], thus substituting the need for such sensors. However, the same reasons also limit the computational power available on board, restricting the real-time deployment of the 3D-reconstruction based planning algorithms. Moreover, these approaches are bound by the inherent limitations of the employed sensors that are challenged by transparent, shinny and textureless surfaces; also used to provide supervisory signals for training their learnable counterparts.

Recently, vision-based navigation has gained a lot of attention in the domain of aerial robotics, owing to its applicability on commercially available quadcopters which are commonly equipped with a forward-looking camera, lacking off-the-shelf integration of other distance sensors. Simultaneously, the tremendous advancement of deep learning can offer great potential to visual navigation, as DNNs enable the development of *end-to-end* learning methodologies [72]. Such data-driven approaches provide enhanced generalisation capabilities compared to their handcrafted counterparts, acting as an enabler for the deployment of mobile robots in real-world environments which typically demonstrate significant variation in visual appearance.

The success of learnable approaches, however, relies heavily on the existence of an abundance of training data. Specifically for the task of end-to-end visual navigation, where a model is fit to correlate raw sensor data directly to robot motion, different flavors of data collection, annotation and training processes have appeared in the literature. These include:

(i) Learning from Demonstration [407], that relies on a human expert executing the target task while their actions are being recorded to act as ground-truth data. This process increases the cost of the data collection, limiting the scale of the resulting datasets.

(ii) Reinforcement Learning (RL) [408], where the robot operates under the control of a learnable policy, that is updated though a reward function. The trial-and-error nature of this process is also slow, requires a human in-the-loop to recover from failures and raises safety concerns when deployed on robots operating in the real world.

(iii) Learning in Simulation (Sim2Real) [409], where policies/ demonstrations can be learned/ provided with the use of a simulator [410]. Although significantly faster and safer, the domain gap between psychical and virtual environments still limits the applicability of such methodologies in the real world [411].

Additionally, end-to-end models learned using the above methodologies are commonly subject to inductive biases, since the provided annotations are tailored to target task and/or the robot/UAV model at hand, as well as the behaviour of the human operator providing the demonstrations. This limits the transferability of the trained models between robots and use-cases, restricting the re-use of the collected data and thus slowing down progress. Not less important, the majority of learnable navigation approaches to date provide independent navigation predictions for each RGB frame, failing to capture the temporal correlations evident in the visual input stream. This contrasts years of study of autonomous navigation through traditional (and bio-inspired [412]) vision-based approaches, which heavily rely on spatio-temporal cues, such as optical flow [413][414][415]. Although several methods are introduced in the literature exploiting such spatio-temporal information in a wide range of computer vision tasks, the lack of adoption in robot vision is attributed to their substantial workload not fitting in the limited computation envelope available on-board UAVs.

This chapter addresses the above challenges in a two-fold manner. Initially, a self-supervised approach for collecting and annotating visual navigation data with minimum human supervision is introduced, exploiting additional sensors instead. Using this methodology, SelfNav, a new dataset for UAV navigation, is collected and shared with the community. Subsequently, a deep learning model is trained to predict these sensor readings, corresponding to *distance-to-collision* values along different directions, directly from monocular visual input. The proposed model's design and training process are carefully crafted, exposing numerous trade-offs to enable the

extraction of spatio-temporal information while avoiding the soaring computational cost of relevant methods. The outputs of the proposed model, form an intermediate representation, that is fed to a local motion planner to modulate the robot's linear and angular velocity, achieving collision-free navigation, while remaining tunable post-training for adaptation to different robots/use-cases.

More specifically, the main contributions of this chapter can be summarised as:

- A self-supervised methodology for collecting and annotating robot navigation data, *through the employment of additional distance sensors mounted on the robot*. Sensor readings are utilised to provide annotations, as well as handle the navigation task during the data collection process, minimising human intervention. Applying the proposed methodology on a UAV, SelfNav, an large-scale navigation dataset with real indoor-flight data is created and made publicly available. SelfNav pairs visual inputs from the UAV's camera to aligned and calibrated sensor readings, signifying the distance-to-collision with the closest obstacle in the robot's path, towards multiple diverging directions within the camera's field-of-view.

- The design of a *two-stream regression CNN*, trained on SelfNav to *predict these continuous distance-to-collision values* in view of the robot's forward-looking camera input, essentially approximating the sensor data. The proposed network architecture, consists of two merging streams that concurrently process the current and previous raw visual inputs. Combined with a *self-supervised pre-training* methodology that exploits information from additional UAV sensors, such as the IMU, to pre-train the model on a proxy task that enhances its representation learning capabilities, allowing the extraction of spatio-temporal features that incorporate both static appearance and robot motion information; while several design optimisations are introduced to alleviate the added computational overhead compared to single-stream baselines.

- A *tunable local motion planning policy* that jointly modulates the robot's yaw and linear velocity, in view of the proposed model's predictions towards three directions (left, forward, right), to accomplish collision-free navigation. Under this formulation, the learnable component of the proposed system is transferable across different UAVs/applications (i.e. with varying minimum allowed distance to obstacles) without the need for re-labelling and training from scratch.

The proposed methodology is designed to equip small-scale and low-cost robots/UAVs (typically carrying a monocular camera), with autonomous navigation capabilities relying on deep learning, without the need to incorporate additional distance sensors. The prediction of continuous distance-to-collision values, as an intermediate representation in place of directly inferring navigation commands, strikes a balance between the modular and end-to-end approaches, enabling the motion planner to remain tunable post-training in order to accommodate varying UAV limitations and/or application-specific requirements. Additionally, the two-stream CNN architecture and its ability to efficiently extract spatio-temporal features significantly boost the prediction accuracy on the task at hand, offering fine-grained information to the proposed motion planner. This allows it to make informed navigation decisions, leading to safe and reliable autonomous navigation through deep learning.

6.3 Related Work

6.3.1 Data Collection and Annotation for UAV Navigation

Across the spectrum of applications, learnable approaches are known to be data-hungry. This becomes a challenge in domains such as autonomous navigation where the data collection process in the real-world is complex, costly, time-consuming and eventually dangerous [70]. For example, the early works of [416] and [417] rely entirely on a human in-the-loop, employing expert pilots to directly fly or progressively supervise-and-correct a UAV respectively, recording <input-image, pilot-command> pairs to act as training data for a DNN model.

To remedy this issue, [72] introduced a transfer learning approach where a CNN trained on data collected by cars and bicycles is successfully applied on a drone flying autonomously in urban environments. In that approach, captured frames were manually labelled offline for classification (as positive or negative with respect to collision danger) based on the bicycle's distance to its surrounding vehicles/objects. The key principles of this concept are closely related to the work of [418], in which a dataset collected by human hikers with head mounted cameras was used to train a DNN for drone navigation on forest/mountain trails.

Towards the same goal, [419] took a different approach, using predicted depth and surface normal data from RGB inputs, along with a 3D cost function to automatically assign ground-truth navigation labels (from a pool of predefined paths) to each image of the dataset.

Self-supervision is exploited across several applications in robotics, through various mechanisms aiming to automate the collection and annotation large-scale datasets for their respective task, spanning from object grasping [420] to road detection [421] and ego-motion estimation [422].

In a relevant (but expensive) endeavour, [423] introduced a UAV navigation dataset where a UAV was able to detect collisions with its environment through its accelerometer data while "blindly" executing straight-line trajectories, and annotate a predefined number of frames captured before each crash as negative samples, for binary collision classification. However, the open-loop navigation behaviour followed in this collection process does not adequately represent realistic UAV deployment scenarios; whereas the coarse annotation rule employed led to a vast imbalance between positive and negative samples.

The approach introduced in this section, adopts several distance sensors mounted on a UAV to provide continuous distance annotations along varying directions (considering the closest obstacle on the robot's path towards each direction), as well as handle its navigation during the data collection process. As a result, the proposed methodology is able to minimise the need for human intervention during data collection and drastically reduce the annotation cost, while providing a significantly finer granularity of annotations, and feature trajectories that re-assemble real-world flight paths.

6.3.2 DNN-based Approaches for UAV Navigation

The majority of recent approaches for autonomous navigation and its highly correlated tasks of obstacle detection and avoidance, employ CNNs due to their state-of-the-art predictive and generalisation capabilities. The adopted processing pipelines in these works can be divided in two types: modular and end-to-end.

The former typically employ deep learning for perception, due to its ability to extract meaningful representation from visual sensory inputs, capturing the state of the robot and its environment. Here DNNs comprise just early blocks within a modular pipeline, also featuring hand-crafted post-processing steps for making navigation decisions. In contrast, end-to-end approaches employ models that learn direct mappings between raw sensor measurements (as input) and control commands (as output).

Both approaches can be characterised by a balance between advantages and disadvantages. Modular approaches being more versatile and interpretatable, suffer from the cost of dealing with the cumulative complexity and noise injected by each module [406]. End-to-end learning

approaches instead are directly optimised to the target task without any reliance to imperfect perception modules; requiring however hard-to-obtain supervisory signals at the application level (e.g. through imitation learning with a human in-the-loop) [416][417], while lacking adaptation capabilities post-training.

Indicatively, following a modular approach, [424] employs multivariate CNN regression (trained on manually annotated data) to predict bounding boxes that identify navigable space in an disparity map, that was generated by a stereo correspondence algorithm from a pair of images obtained after a transnational movement. After post-processing, the largest predicted bounding box is used to determine the next navigation waypoint, fed to the robot planner. Accordingly, in [406], a depth map is predicted for each monocular image captured by the drone's on-board camera, using a CNN trained on RGB-D data. Then, a deterministic arbitration scheme is employed to steer the UAV away from obstacles by controlling its angle on two rotational degrees of freedom (DoF), based on a hand-crafted analysis of the generated depth map.

On the other hand, DroNet [72] introduces an end-to-end approach, reducing autonomous UAV navigation to a regression problem where the desired yaw-velocity of the drone is directly predicted from each input frame by a CNN model. Sharing the same backbone, a classification head is also employed to predict a collision probability, that is used to control the UAV's linear velocity accordingly. Along the same lines, TrailNet [96] focused on autonomous UAV navigation on forest trails, where a CNN is trained to predict the desired drone's orientation and lateral offset with respect to the centre of the trail, used directly to control transnational and rotational speed in order to retain its path on the trail.

The approach introduced in this chapter lies in between these two schools of thought, effectively combining the best of both worlds. A modular approach is employed in the sense that a local motion planner conducts some *tunable* post-processing of the CNN's prediction, allowing for post-training adaptation of the navigation policy that is implicitly learned. This step facilitates the adaptation of the proposed model to different UAV modes and tasks (where different distance to obstacles is required during flight), without the need to re-annotate the dataset and fine-tune the model accordingly. At the same time, the proposed approach is able to yield actionable predictions, in the sense of the proposed intermediate representation of real distance-to-collision towards different directions, instead of primitive perception outputs (as in many modular approaches), minimising the need for hand-crafted data analysis, similarly to end-to-end approaches.

Most closely related to the methodology introduced in this chapter is the state-of-the-art indoor navigation work by Gandhi et al. [423]. In that work, autonomous navigation is casted as a binary classification problem, between collision-free and non-navigable space. A CNN is trained on this classification task and employed upon deployment to yield predictions for different crops of the input image. These predictions are later fused by a deterministic algorithm to conclude on a single yaw-angle control command. The proposed approach differs from this work in two key ways:

Initially, the approach introduced in this chapter treats navigation as a continuous regression problem, exploiting the fine granularity of annotations provided by the newly introduced dataset. This allows to yield more intuitive and interpretable (actual distance-to-collision) outputs, leading to more informed navigation decisions. Additionally, as discussed above, this design choice enables the post-training adaptation of the system to different UAVs and application requirements without the need of data re-annotation and model re-training. Of course, regression has been employed in several computer/robot vision tasks in the past, including human pose estimation [425], 6-DoF camera localisation [426], robot grasping [75]. Most relevant is the approach of [72], that *directly* regresses the desired yaw value for collision-free UAV navigation, in view of the input image; being trained on a task-specific dataset with steering angle annotations. In contrast, the proposed model predicts an *intermediate* distance-to-collision representation towards three directions, that is consumed by a local motion planner to jointly modulate the robot's velocity and orientation. As a result, the strong reliance on UAV and task-specific annotations is waived.

Secondly, the "Siamese" architecture adopted by the proposed model is inspired by two-stream CNNs found in the literature in use-cases such as multi-sensor fusion [427], RGB and Depth data fusion [428], video classification [301] and human action recognition on videos [429]; all coming with a considerable computational and memory overhead. In contrast to the majority of related works in UAV navigation (including [423]) where each prediction solely relies on a single image, through the use of the proposed architecture the introduced model is able to extract and exploit spatio-temporal information. This leads to improved predictive accuracy, as well as a more temporally consistent navigation policy. Furthermore, the careful design choices, along with the analytical examination of their speed-accuracy trade-off undertaken in this chapter, act as an enabler for the efficient deployment of the resulting model in latency-critical applications, such as UAV navigation.

6.4 Methodology

6.4.1 Overview

This chapter investigates the problem of autonomous navigation, focusing on data layer approximations, that allow the efficient estimation of explicit (data) and implicit (feature) information cues, instead of analytically calculating or measuring them. The aim is to equip low-cost robot platforms with autonomy capabilities, relying solely on visual data as an input, alleviating the payload and cost off additional sensing devices.

Focusing on mobile robots/UAVs equipped with a forward-looking camera, this section *provides* a methodology for autonomous navigation that:

- Is able to exploit deep learning to approximate other sensor readings (that are not available on-board) from visual data and effectively incorporate them on the navigation process.
- Is able to implicitly extract and exploit temporal information from the frame sequence, boosting the predictive accuracy of the model, without incurring a computational overhead that would hinder its real-time performance, as in the case of explicitly-calculated optical-flow.
- Does not rely on manually annotated navigation data that are task-specific (i.e. follow a predetermined navigation policy, or incorporate UAV/application-specific biases), incurring significant overheads to adapt to different settings.

As such, in this section autonomous obstacle avoidance is formulated as a modular pipeline, where a deep-learning model is designed (Sec. 6.4.4) and trained (Sec. 6.4.5) to approximate sensor readings that are otherwise unavailable on the target system, followed by a tunable local motion planning module (Sec. 6.4.6), able to yield informed navigational decisions, in view of the model's predictions. The proposed system is trained on a custom dataset (Sec. 6.4.3), created in a Self-Supervised way (Sec. 6.4.2), refraining from the incorporation of use-case or UAV-related annotation biases.

6.4.2 Self-Supervised Data Collection and Annotation

Deep Learning methods require a large amount of data in order to train models that can generalise across different real-world environments. Data collection and annotation, however, is time

consuming and costly, especially when the process involves a real robot interacting with its environment under the control/supervision of humans experts.

With a focus on robot navigation, this chapter aims to minimise the human involvement in this process by introducing a *self-supervised* methodology, where additional distance sensors are mounted on the robot hull, in order to: (i) tackle the robot navigation task during the data collection and (ii) undertake the data annotation task for the collected data. The aim is to create a large-scale dataset for visual indoor navigation, featuring real *distance-to-collision* labels coupled with corresponding visual cues from the robot's forward looking camera. Distance is measured with respect to the closest visible[1] obstacle[2] across multiple diverging directions.

More specifically, a commercially-available UAV is employed[3], and three pairs of Ultrasonic and Infra-Red (IR) distance sensors are mounted on its hull pointing towards different directions within its camera's field of view. As depicted in Fig. 6.1, the sensors are aligned to face towards [-30°, 0°, 30°] with respect the centre of the captured frame across the image width, and on the same height as the UAV with a 15° angle along the image height axis.

The design choice of pairing different distance sensor technologies is made to alleviate the downfalls of each technology. In more detail, *Ultrasonic sensors* based on time-of-flight measurements, support longer range distance sensing at the cost of providing many noisy samples due to scattering and reflection effects when obstacles are positioned too close or on a narrow angle to the sensor. Conversely, *Infra-Red sensors*, based on light intensity measurements, demonstrate better behaviour on a wide range of sensing angles, however their scope is limited to notably shorter distances while being affected by lighting conditions.

Upon deployment for data collection, two navigation strategies are introduced, making use of the provided distance sensor readings:

(i) Open-loop: Initially, the drone executes straight-line trajectories, until the forward-looking distance sensor detects an obstacle at the minimum distance that the drone requires to stop without colliding. Then, a random rotation is performed to determine the flight direction of the next trajectory, which should be towards a navigable area of the environment. This constraint is evaluated by thresholding the distance reading of the same UAV sensor after the rotation. This

[1] In the sense of being within the camera's field-of-view.

[2] Defined as any object being present in the robot's path.

[3] The choice of this platform is made due to its unconstrained kinematic capabilities in the 3D space. This enhances the applicability of the proposed methodology to other robot platforms, with their more constrained kinematic space being expressed as a subset of the one examined in this chapter.



(a) (b)

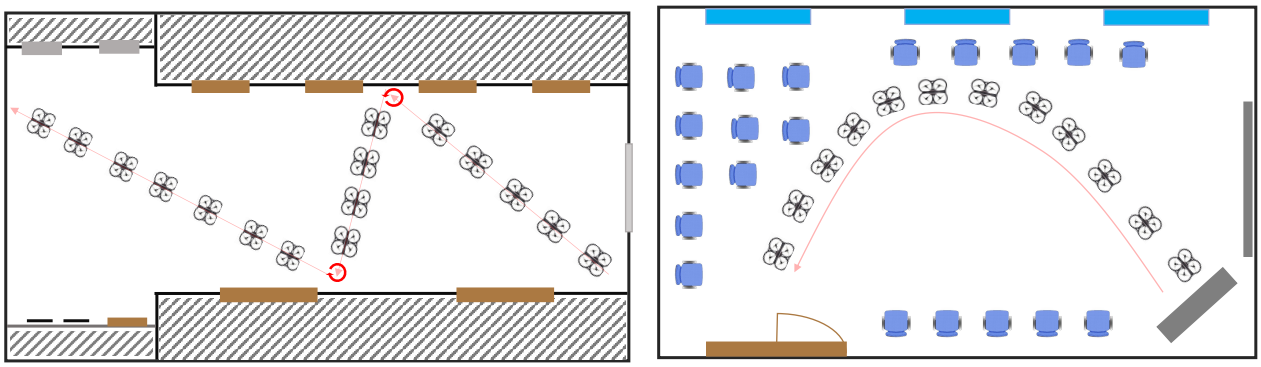
Figure 6.1: (a) Top-view of distance sensor alignment on the UAV used for data collection. (b) Sample frame captured by the UAV camera, along with distance sensor alignment and colour-coded distance readings (green: free-space, red: collision).

process, also illustrated in Fig. 6.2a, replicates the approach followed by [423] offering, however, more fine-grained and informative annotations for each frame.

(ii) Closed-loop: In order to create more diverse and realistic data, a second navigation strategy based on an exploration algorithm is also employed. In this setting, all three distance readings (towards the left, forward and right direction) are fed to the local motion planning methodology introduced later (Sec. 6.4.6). This strategy makes planning decisions towards the most navigable space in the UAV's field of view, when meeting an obstacle in the direction of flight; as illustrated in Fig. 6.2b.

In both strategies, the UAV *autonomously* plans and executes data collection trajectories, making use of the distance measurements provided by the external sensors mounted on its hull. All frames from the UAV's forward looking camera are captured and timestamped, along with raw measurements from all distance sensor pairs sampled at 30Hz, that will act as ground-truth labels after necessary post-processing and fusion. Alongside, raw measurements from other sensors available on-board the UAV, such as IMU and altitude data are captured at the same rate, mainly aiming to act as supplementary data cues that would facilitate different applications, such as Visual-Inertial methods [430]. This self-supervised approach offers an automated and highly-scalable data collection and annotation process, as it eliminates the need for a human expert being in-the-loop and minimises the need for human supervision.

An automated post-processing step is employed, normalising and fusing the measurement from both sensor technologies in each pair. Data from the Ultrasonic sensor are used for samples with large-distance obstacles and the IR data when available due to presence of obstacles in



(a) (b)

Figure 6.2: Navigation policies during self-supervised data collection process: (a) following straight-line trajectories; (b) following distance-sensor based exploration strategy.

close proximity. The fused annotations of each trajectory are also processed by a low-pass filter to eliminate sensor noise and unwanted spikes.

6.4.3 The *SelfNav* Dataset

Following the above methodology, SelfNav, the first large-scale indoor-flight dataset annotated with fine-grained real distance-to-collision labels is created and shared with the community[4]. SelfNav fills a thoroughly discussed gap in the literature [72], offering dense (every-frame) and intuitive annotations, relevant to the autonomous robot navigation task, that are transferable across UAVs and tasks.

The data collection process of SelfNav took place within the Imperial College campus, featuring various real-world indoor environments (including hallways, seminar rooms, office spaces, kitchen areas etc.). A total of more than 300,000 samples, organised in approximately 2,000 trajectories, have been collected and annotated with three distance labels (towards different directions) each, following the self-supervised methodology introduced in Sec. 6.4.2. Every captured image is divided into three overlapping windows, each being annotated with the fused measurements of the corresponding sensor pair. Distance labels range from 20cm to 5m, corresponding to the limits of the sensors. Fig. 6.3 depicts representative data points from SelfNav, across the collected spectrum.

[4]The SelfNav dataset has been made publicly available, under an Open Data Commons (ODC) Attribution License, and can be found at https://www.imperial.ac.uk/intelligent-digital-systems/indoor-uav-data/.

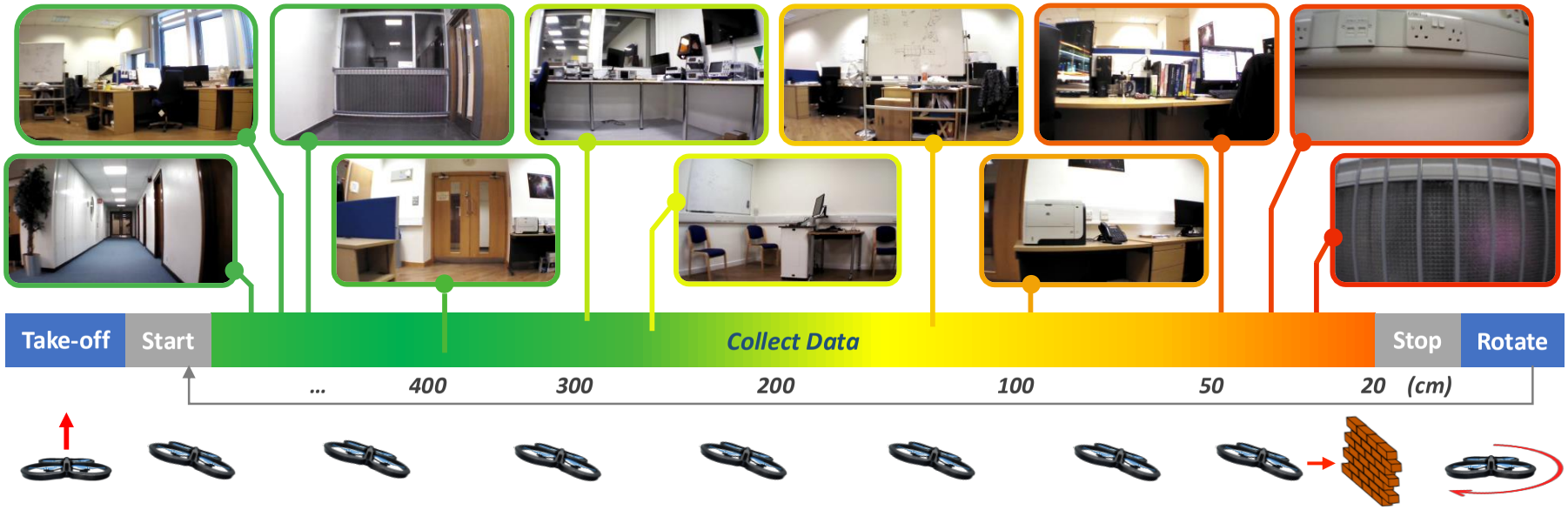


Figure 6.3: SelfNav: an indoor flight trajectory dataset in real-world buildings, automatically annotated with distance-to-collision labels.

6.4.4 CNN Architecture

To enable autonomous navigation on small-scale and low-cost robots/UAVs equipped with monocular cameras, a CNN-based regression model is designed and trained on the newly introduced SelfNav dataset. This model essentially approximates the distance sensor data, relying solely on the input visual cue that is commonly available across robot platforms.

Regression CNN for distance prediction

A vanilla AlexNet [20] backbone is adopted as a starting point for the custom-designed CNN architecture introduced in this section, due to its small computational and memory footprint. Although relevant work has shown improved convergence speed by casting regression as a classification problem and discretising the continuous ground-truth labels into bins [362], this contradicts the motivation of this chapter to predict real distance values, approximating the actual sensor measurements. Thus, taking advantage of the fine granularity of annotations in the proposed dataset, the last layer of the final classifier of the original AlexNet is replaced by a regression unit. This consists of a single-neuron Fully-Connected layer, that directly predicts the desired distance-to-collision value.

Spatio-Temporal Feature Extraction

Although, in robot vision, image sensors provide a sequence of temporally correlated frames, related work has focused solely on learning spatial (i.e. static appearance) features by individually processing each input sample to make independent navigation decisions [423]. In contrast, other computer vision applications such as action recognition [431] commonly exploit spatio-temporal information through the use of two-stream "Siamese" CNNs [432]. These approaches however, require the pre-calculation of explicit temporal feature representations, such as dense optical flow [433][434], that is provided as an input to the model along with the current RGB frame [435]. This methodology introduces a significant cost, and is not computationally tractable within the limited resource envelope available on-board small-scale UAVs, under the latency-critical setting of autonomous navigation. Towards this direction, parallel work [436] exploits data-driven approaches to approximate explicit temporal representations, such as optical flow, through dedicated DNNs [437] instead of analytically calculating. Although highly-accurate, this approximation remains exceedingly computationally expensive for real-time deployment in the embedded landscape; while

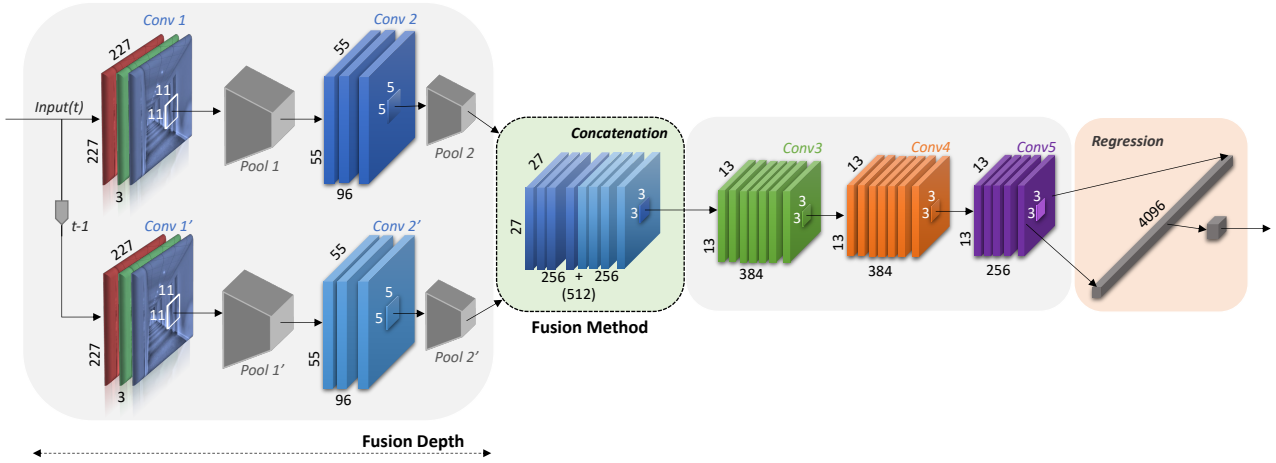


Figure 6.4: Two-stream CNN architecture for spatio-temporal feature extraction on regression tasks.

also requiring fine-tuning on the target domain to generalise robustly in unseen environments [438], limiting its applicability in the real world.

To remedy the above issue, the approach introduced is this chapter aims to *implicitly* leverage the temporal information encapsulated between consecutive frames, by enabling the extraction of spatio-temporal representations within the model itself, at a minimal computational cost. More specifically, two contiguous-in-time frames are directly fed to the proposed model, without any computationally-heavy pre-processing required. The model is then guided at training time (discussed in Sec. 6.4.5) to extract spatio-temporal features; which implicitly capture temporal information such as relative motion and local proximity, that would allow the proposed model to make more informed distance predictions.

From an architectural perspective, the two RGB frames are initially processed concurrently by two parallel streams of convolutional layers that independently extract spatial (appearance) information from both static images. The two streams are fused to a single one deeper in the network, in order to extract temporal features by correlating the two representations, essentially incorporating information regarding the relative motion between the two frames. Hence, both spatial and temporal components of the input sequence are extracted exploited by the trained model and exploited while making distance predictions. Notably, this process is characterised by design choices regarding the proportion of the model's layers allocated for processing each frame independently, compared to the subsequent joint processing stage, as well as the most appropriate fusion method between the two streams.

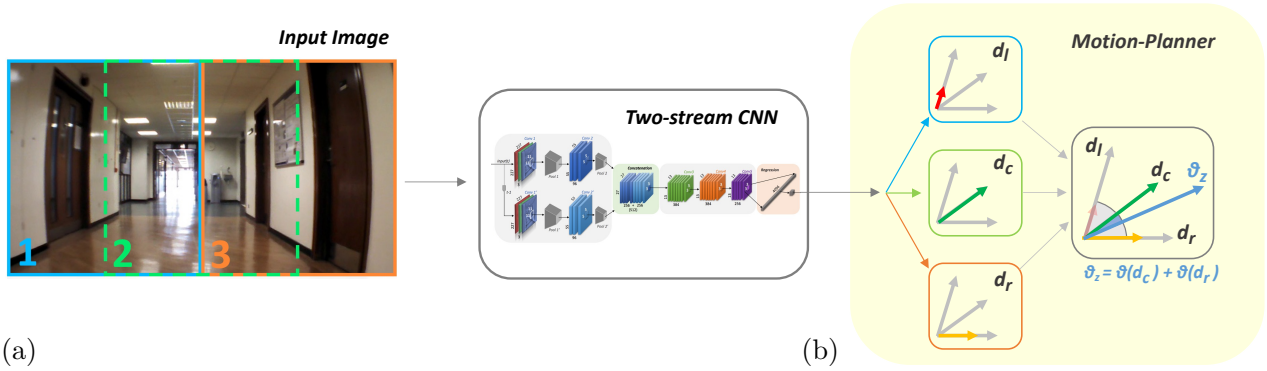
Model Architecture Search

Determining the appropriate *fusion method* and *fusion depth* between the two streams of a Siamese network does not have a "one size fits all" solution. Instead, these design choices are heavily dependent on the input data and target application. As such, this chapter exposes the above design dimensions to a manual grid search [301]. The design space consist of two candidate fusion methods, namely {concatenation, addition} and three fusion depths: {early, mid, late}. Fusion by concatenation essentially appends the feature maps of the two streams across the channel dimension and processes the combined feature volume though the subsequent convolutional layer; whereas addition-based fusion perform element-wise addition between the two feature volumes before propagating to the next layer. Accordingly, early fusion merges the two RGB frames at the input level; while mid/late fusion merges their corresponding feature volumes half-way or at the output of the feature extractor respectively.

The highest performant model accuracy-wise on the target distance prediction task is illustrated in Fig. 6.4. This architecture employs mid-fusion by concatenation, with the two streams being merged after the second convolutional layer of AlexNet. As a result, the third layer needs to be slightly modified, to receive the more channel-rich merged feature volume as input. More details on the findings of this grid search are presented in the evaluation of this chapter (Sec. 6.5.2).

Inference Process

Upon deployment, the input image is initially down-sampled to match AlexNet's input height requirement (227 pixels). This allows the model to capture and exploit as much context as possible from the robot's observations of environment. Subsequently, three rectangular windows are cropped out of each input frame, across the width dimension. The crops are centred so as to match the sensor alignment of the training dataset, yielding overlapping windows as depicted in Fig. 6.5a. Each window is associated with the corresponding crop from the previous frame and processed independently by the proposed CNN. As a result, three independent distance predictions, each corresponding to a different navigational direction, are provided for each frame. Subsequently, the motion planner introduced in Sec. 6.4.6 is responsible to translate these predictions to actionable navigation commands.



(a) (b)

Figure 6.5: (a) Overlapping windows of sample input image processed by the proposed two-stream CNN. (b) Instance of proposed local motion planner.

6.4.5 Model Training and Optimisations

Training Process And Accuracy Optimisations

During training, each cropped window is treated as an independent training sample, as happens at inference time. This way the CNN exploits information from the total of the annotated samples on the training set, holding out a portion of the data to form validation and test sets.

All the convolutional layers of the backbone are initialised to the ImageNet [21] pre-trained weights of the original AlexNet model [20]. In the fusion layer the weights are replicated twice, in order to match the appended feature volume of the two merging streams. Due to the variations introduced in the architecture of the model, its feature map distributions are expected to demonstrate a heavy drift during inference. Additionally, extracting correlations between the two merged streams may not be an intuitive task for the model, in its initial state, as per its previous training.

As such, it is beneficial to pre-train the proposed model in a way that would *guide* the two-stream feature extractor towards excerpting (spatial) correlations between frames, which describe several characteristics about the implied (temporal) motion of the camera. For this purpose, a *self-supervised pre-training* [439][440] methodology[5] is introduced. In this setting, a proxy-task that heavily relies on temporal features, and for which ground-truth labels can be trivially generated from raw data or external sources, is employed to pre-train the weights of a proposed model. This process would implicitly push the model to search for weights that are able to extract such correlations, without explicitly specifying their structure.

[5] Also referred to as Unsupervised Pre-training [143] in the literature.

In the context of robot navigation, in this chapter the IMU data captured in SelfNav along with the visual inputs, are exploited to pre-train the proposed two-stream CNN. A proxy task of identifying the flight direction of the UAV is established. As this prediction relies completely on understanding the correlations between the two frames, training in this task guides the models to extract a temporally descriptive feature representation from each input frame pair, restricting it from taking "short-cuts" such as focusing solely on spatial features. Subsequently, the model can take advantage of this learned representation, by further fine-tuning on the target task.

During the pre-training phase, two consecutive frames for the dataset are fed to the proposed regression model along with a ground-truth label, corresponding to the angular difference as captured by the IMU data between the two timestamps. The model is trained through Stochastic Gradient Descent (SGD) with Momentum for 60 epochs, to predict this continuous direction value. To achieve that, back-propagation forces the backbone to extract strong correlations between the two frames, that implicitly capture temporal characteristics of the robot motion, effectively initialising the model for the navigation task at hand.

Finally, training of the proposed regression CNN on the target distance-prediction task is performed using the SelfNav dataset, with SGD over another 30 epochs, employing a mini-batch size of 128. The training set is shuffled on each epoch to form diverse batches. At the same time, multiple data augmentation techniques are applied on, including random rotations, horizontal flipping and lighting variations (focusing mainly on brightness, contrast and saturation). However, no augmentations affecting the scale of the captured images are performed, to avoid distorting the implicitly learned camera model correlating the 2D image plane with the depth (distance) predictions.

Performance Optimisations

The two streams of the proposed CNN are simultaneously processing two input frames, aiming to extract spatial appearance features from both, that are later correlated to figure out temporal correspondences, encapsulating motion information. The addition of the second stream, however, adds a computational and memory overhead on the resulting model, penalising its inference latency.

To address the memory issue, a weight re-use approach is employed. Driven by the fact that in the proposed model both streams are processing inputs sampled from the same distribution, in contrast to work employing an explicit temporal representation such as optical flow on the second stream [435], a single set of weights can be re-used between the corresponding layers of

both streams. Apart from reducing the memory footprint of the resulting model, this technique can enhance the re-use of the weights loaded on-chip, by pushing a larger amount of data through each layer at once by batching the two inputs. This essentially makes the model less memory bounded, improving its attainable inference latency.

Additionally, since the proposed model operates on a streaming fashion, under some assumptions[6], the "current frame" during timestep[7] t can also act as "previous frame" during timestep $t+1$. This approach allows the re-use of intermediate feature maps for a single frame, across two successive timesteps. From a systems perspective, this is enabled by a feature map memoisation approach, that saves the intermediate feature maps of the top ("current frame") stream right before they are merged at the fusion layer on the global memory, and re-uses them by fetching them back on chip during the next timestep, as an input to the fusion layer corresponding to the bottom ("previous frame") stream this time. Hence, the proposed optimisation approach essentially trades computation for memory transactions, exploiting the large off-chip bandwidth available on GPUs.

It is noteworthy that as deeper layers of the proposed model yield feature volumes with reduced spatial dimensions and increased number of channels compared to shallower ones, as also happens in most CNN architectures to date, a balance is stricken between the memory transactions required for fetching the second input image or the intermediate feature volume. This avoids excessive memory overheads that could be caused by the proposed feature memoisation technique if blindly applied under different settings.

Finally, since three windows are extracted from each input image and become available at the same time, batching is also employed across these image crops, to facilitate further weight re-use upon deployment.

6.4.6 Local Motion Planner

During inference, the trained model yields three distance-to-collision values predicted in view of the robot's visual input. A hand-crafted motion planner is proposed to reduce these three distance values to motion commands towards navigable space (i.e. avoiding collisions with obstacles).

[6]Such assumptions rely on maintaining a fixed latency between successive inferences to avoid stochasticity in the sampling rate between each pair of frames; and preserving a meaningful time difference between two consecutive inferences such that the two subsequent inputs maintain a high overlap that allows the extraction of temporal correlations across them.

[7]Timestep here refers to an complete inference of the proposed model on a new data sample.

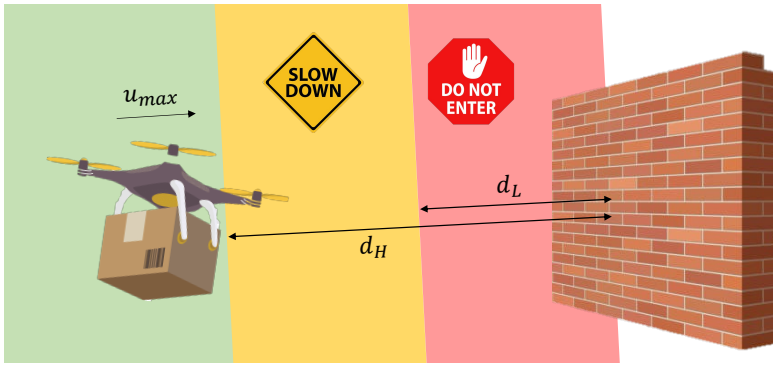


Figure 6.6: UAV- and Application- specific tunable parameters of the proposed motion planner.

Towards this goal, the three continuous-value predictions are projected to 2D vectors on the flying-height plane[8], considering the direction of the corresponding distance sensors used in the training data. Thus, let $|\vec{d_l}|, |\vec{d_c}|, |\vec{d_r}|$ denote the magnitude of the resulting distance-to-collision vectors towards the left, centre and right directions, corresponding to an [-30°, 0°, 30°] angle with respect to the heading of the robot, respectively. In view of these vectors, the local motion planning algorithm produces a motion command that modulates the robot's forward linear velocity (u^x_{lin}) and rotational (yaw) velocity (u^z_{rot}). Importantly, under the formulation the model's predictions remain fully decoupled from the robot's kinematics and dynamics. Hence, adapting the proposed method to a different robot platforms, solely requires tuning the motion planner's parameters.

More precisely, the *linear velocity* of the robot $u^x_{lin} \in [-u_{max}, u_{max}]$ is assigned to be inversely proportional to the predicted distance-to-collision for the central window of the visual input (Eq. 6.1), in order to avoid collisions in the direction of flight of the UAV.

$$u^x_{lin} = \begin{cases} \dfrac{|\vec{d_c}| - d_L}{d_H} \cdot u_{max} & \text{, if } |\vec{d_c}| < d_H + d_L \\ u_{max} & \text{, otherwise} \end{cases} \tag{6.1}$$

where d_L, d_H form tunable, task- and drone-specific parameters representing the minimum distance that the UAV is allowed to get close to obstacles, and the minimum distance from an obstacle that the drone is capable to decelerate from to avoid collision when flying on its maximum allowable velocity u_{max}, accordingly (Fig. 6.6).

Accordingly, the UAV's *rotational velocity* across the yaw-axis $u^z_{rot} \in [-1, 1]$ is adjusted to guide the UAV towards the direction that is predicted to display the largest amount of navigable space within the current frame's FoV. This direction is determined by the angle of the resultant

[8]Plane being parallel to the ground, at the current flying level of the UAV. This can generalise to the floor plane for ground robots.

distance vector after fusing all the consecutive distance vectors that exceed a tunable distance threshold value d_H. An example of this process is illustrated Fig. 6.5b; while Algorithm 6 describes the above strategy in more detail.

Algorithm 6 Rotational Yaw Velocity

Input: $\{\vec{d_l}, \vec{d_c}, \vec{d_r}\}, d_H, d_L$
Output: u^z_{rot}

if $\{|\vec{d_l}|, |\vec{d_c}|, |\vec{d_r}|\} > d_H + d_L$ **then** ▷ Free space
  $\theta_z = \theta(\vec{d_l} + \vec{d_c} + \vec{d_r})$
else if $|\vec{d_l}| \geq |\vec{d_r}|$ **then**
  $\theta_z = \theta(\vec{d_l} + \vec{d_c})$ ▷ Close to Obstacle; with free space towards left
else if $|\vec{d_l}| < |\vec{d_r}|$ **then**
  $\theta_z = \theta(\vec{d_c} + \vec{d_r})$ ▷ Close to Obstacle; with free space towards right
end if
$u^z_{rot} = \frac{\theta_z}{\theta(\vec{d_r})}$ ▷ Resulting yaw velocity

where $\theta(\cdot)$ indicates the angle w.r.t vector $\vec{d_c}$.

Low-pass filtering is employed on the calculated velocities to eliminate undesirable oscillations in robot motion, before being fed to the on-board controller. Hence, under this planning policy, the UAV instantly reacts to changes in its visual input, based on the trained model's distance prediction. Two key advantages of the proposed approach, over the classification-based methodology followed by [423] and [72] distinguishing between navigable and non-navigable space, are worth mentioning here:

Firstly, the information-rich predictions of the proposed regression CNN provide a fine-grained and accurate distance estimation, gradually escalating across a wide range of real distance-to-collision values. The proposed motion planning policy can leverage this information to perform continuous fine adjustments of the robot's motion. This allows commanding timely manoeuvres that result to smoother navigation and prompt interaction with the environment, as well as insightful longer-range planning decisions by selecting to move towards the direction that is considered to contain the largest amount of traversable space.

Additionally, the proposed regression approach offers enhanced tunability of the navigation policy as a result of the more primitive nature of information that is fed to the motion planner. As discussed, the behaviour of the proposed policy can be adjusted through a number of tunable thresholds whose values may differ significantly between different UAV platforms and/or application scenarios. For example, different drones demonstrate varying levels of manoeuvrability, leading to

unalike reaction time and required stopping distance (d_H). Moreover, tuning can be performed to meet task-specific requirements, such as keeping the drone on a large safety distance from obstacles (d_L), e.g. in scenarios where failures are unrecoverable. Furthermore, the maximum flying speed of the UAV (u_{max}), determined both form the platform specifications and application requirements, is also considered in the behaviour of the proposed motion planner. Conversely, the classification-based setting followed in the literature is less condescending in such tuning as its behaviour is mainly dictated ahead of the training of the model, that implicitly encapsulates the above parameters. As a result, time-consuming re-annotation and re-training of the model is required for making equivalent adjustments, in contrast to the proposed methodology, where thresholds can literally be adjusted on-the-fly.

6.5 Evaluation

6.5.1 Experimental Setup

In the experimental evaluation of this chapter, a Parrot AR-Drone[9] 2.0 was used for the data collection and autonomous flight experiments. This drone is equipped with a 720p forward-facing camera, with a wide-angle lens (92°) that captures images at 30 fps. The maximum speed u_{max} is bound to 1m/s, while distance thresholds d_L and d_H are set to 20cm and 100cm accordingly. Upon deployment, the Ardrone_autonomy package[10] and ROS Kinetic Kame were used. Moreover, during the data collection process, external distance sensors (GP2Y0A60SZLF Analogue IR Sensor and HC-SR04 Ultrasonic Sensor) were attached on the UAV's hull connected on an Arduino Pro Mini micro-controller. For the data collection process, data were transferred and recorded to a laptop, through a direct 2.4GHz Wifi link.

The proposed methodology is deployed on an Nvidia Jetson TX2 board, featuring a 256 CUDA-Core embedded GPU (Pascal Architecture), 8GB of RAM and an Quad ARM A57 CPU. The TX2 module fits the payload capability of AR-Drone 2.0 and is connected with the on-board controller using the methodology presented in the supplementary material of [436].

CNN design and training are performed using MATLAB R2017b and the Neural Network Toolbox on a desktop server equipped featuring a 2560 CUDA-Core Nvidia GTX1080 GPU (Pascal Architecture), 64GB RAM and an Intel Xeon E5-2630 CPU. At training time, the SGD

[9] https://www.parrot.com/us/drones/parrot-ardrone-20-power-edition
[10] http://wiki.ros.org/ardrone_autonomy

optimiser was used, with momentum set to 0.9, and a starting learning rate of 0.001 being reduced by a factor of 10 after every 10 epochs (step scheduling). Moreover, regression prediction and ground-truth values were normalised (in the range of [0,1]) to prevent an exploding behaviour of the gradient values that affects the convergence of the training method. For this purpose the maximum distance prediction is saturated at 5m, which is also the nominal limit of the long-range ultrasonic sensor employed during the data collection process.

6.5.2 Model Design and Prediction Accuracy

This section focuses on evaluating the distance predictions of the proposed model on a previously unseen test-set of the SelfNav dataset. This test set comprises 20,000 real-flight pictures from a variety of indoor environments, covering the whole spectrum of distance values captured in the dataset.

Comparison to Baselines

Initially a comparison with *single-stream* CNN baselines, relying solely on spatial features from the current frame on each prediction is conducted. Two baselines are employed, both based on the AlexNet [20] architecture and trained on the SelfNav dataset following the same practises as in the proposed model:

- The first baseline follows a *regression* approach, adopting an identical regression unit with the one proposed in this chapter; built on top of a vanilla single-stream AlexNet backbone;
- The second baseline is based on the *classification* approach of [423], featuring the original AlexNet classifier and trained after binarising the SelfNav dataset labels to navigable and non-navigable space using the (equivalent of a) $d_H = 100cm$ threshold value from Sec. 6.4.6.

Figure 6.7 summarises the overall picture of this experiment. Focusing on the left y-axis, it can be seen that the proposed two-stream model consistently demonstrates a considerably narrower confidence interval compared to the single-stream regression approach, across the spectrum of actual distance values. This suggests that the predictive accuracy of the model is enhanced by the incorporation of spatio-temporal features, allowing for more informed predictions and potentially contributing towards the disambiguation of conflicting information extracted based on visual appearance (e.g. by also considering the relative motion information between frames).

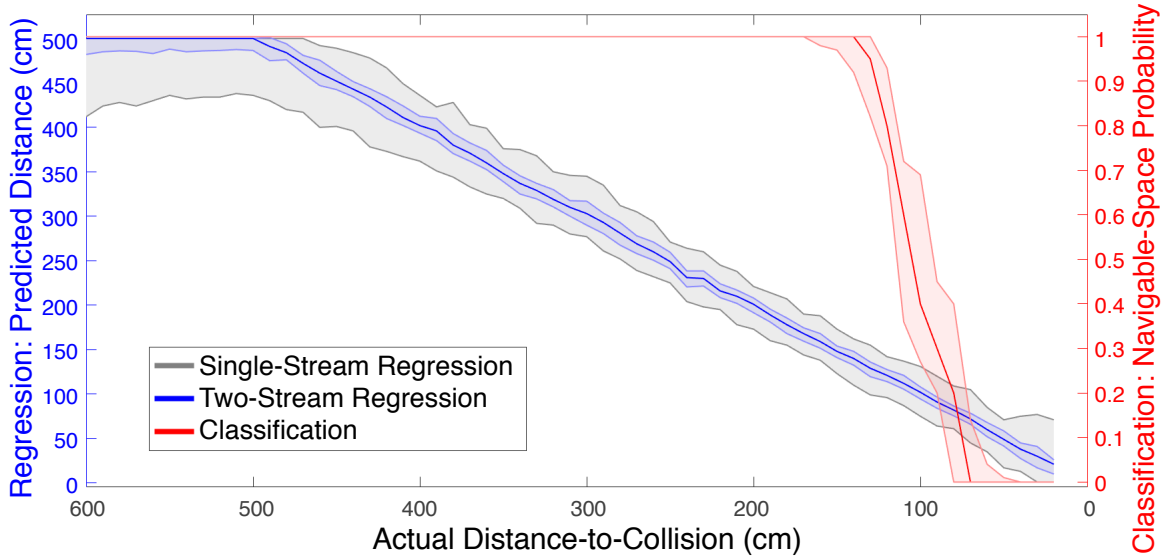


Figure 6.7: Actual vs Predicted distance-to-collision from proposed and baseline CNNs.

On the right y-axis, the output of the binary classification model of [423] is also illustrated[11]. Although that approach remains effective and demonstrates strong predictive accuracy on the classification task, it can be seen this formulation of the navigation task is severely less informative. For the greatest expanse of the actual distance range the model yields and overconfident navigable-space probability. Across the short range of 70-120cm distance-to-collision labels, a highly escalating behaviour with overly wide error range is observed. This area is positioned around the labelling threshold of the two classes, which is determined ahead of training and cannot be adapted afterwards. Conversely, the proposed regression-based approach demonstrates a more information-rich and smoothly escalating behaviour across the whole range of the examined distances with significantly lower error range, bounded only by the maximum sensing range of the external distance sensors used during data collection (500cm). As demonstrated later in this section, this complementary information can be effectively exploited by the motion planning policy to make more informed longer-range decisions for autonomous navigation.

Comparison between Model Design Variants

In this section an empirical analysis between various configurations of the proposed two-stream network architecture is presented. More specifically, different variants of fusion depth and fusion method are cross-examined, as discussed in Sec. 6.4.4; with the highest-performant instance

[11]The predicted probability of facing navigable space p_{nav} is shown here, with the probability of collision being trivially deduced as $1 - p_{nav}$.

Table 6.1: Distance-to-collision Prediction Evaluation (and Model Design Analysis)

Network Architecture			**RMSE** (Val)
Streams	*Fusing Depth*	*Fusing Method*	*(meters)*
Single-stream	-	-	0.167
Two-Stream	Early (*input*)	Concatenation	0.050
Two-Stream	**Middle (*Conv2*)**	**Concatenation**	**0.046**
Two-Stream	Late (*Conv5*)	Concatenation	0.051
Two-Stream	Early (*input*)	Addition	0.138
Two-Stream	Middle (*Conv2*)	Addition	0.086
Two-Stream	Late (*Conv5*)	Addition	0.087

in terms of accuracy being adopted as the proposed architecture (Fig. 6.4). All models are trained on SelfNav following the same process and prediction accuracy is evaluated in terms of Root-Mean-Squared-Error (RMSE) between the predicted and ground-truth values of each frame of the held-out validation set also comprising 20,000 samples.

The results of this comparison are listed in Table 6.1; where the single-stream baseline from the previous experiment is also included. Initially, it is noteworthy that all variants of the proposed two-stream CNN architecture radically overperform the single-stream approach, in terms of predictive accuracy. This re-affirms the argument for exploiting spatio-temporal information on the visual navigation task.

With respect to the *two-stream architectures*, the instance employing mid-fusion by concatenation outperforms all other variants and is selected for deployment. This configuration initially extracts spatial features of an intermediate level of abstraction from both input frames, before fusing the two streams to extract temporal relations between them. It can be presumed that this configuration strikes a good balance in the allocation of the model's learning capacity between spatial and temporal feature extraction. It is also observed that, network configurations employing channel concatenation as a fusion method generally achieve higher accuracy on the regression task, compared to those using channels-wise addition. This can be explained by the clear separation between the feature maps of the two streams that is preserved by the concatenation operation, and combined with the enhanced flexibility offered by the subsequent convolution, essentially weighting the contributions of each channel on the fused representation through learnable parameters.

6.5.3 Training- and Inference-time Optimisation Ablations

This section evaluates the speed-accuracy trade-offs arising as a result of different design-choices and optimisations applied on the proposed model during inference and training. The results of this analysis are summarised in Table 6.2.

Inference Optimisations

Due to the latency-critical nature of the visual navigation task, real-time performance (>30fps) is crucial for making mission-critical decisions in a safe and robust way upon deployment. As reported on the top block of rows in Table 6.2, a vanilla two-stream implementation adopting the architecture, discussed in the previous section, falls considerably short of this requirement (row (ii)). This was expected as the replication of the first two layers forming the additional CNN stream considerably increases the number of parameters and workload of each forward pass. However, the proposed weight-sharing (row (iii)) and feature memoisation (row (iv)) optimisations almost fully alleviate this overhead, reducing the workload to the same level as the single-stream baseline. Additionally, batching the forward passes of the three windows cropped from each frame considerably improves the inference efficiency (row (v)). This is achieved by increasing the re-use of the loaded weights across the model, but most importantly on the memory-bounded Fully-Connected layers at the regression unit, resulting to a better utilisation of the available resources on the target GPU platform.

The interplay between all the above optimisations enabled the proposed approach to achieve real-time performance on-board, without any compromise in accuracy. Instead, it is noteworthy that the proposed weight-sharing approach across the two streams (row (iii)) unexpectedly provided a significant boost in accuracy, compared to the original two-stream approach (row (ii)); and even more the single-stream baseline (row (i)). This can be attributed to the better alignment between the intermediate representations extracted for both frames, guaranteed by processing them through the same filters, that presumably allows easier matching of correspondences between them and thus the extraction of more accurate temporal information.

Implicit vs Explicit Representation and Training Optimisations

In the second block of rows of Tab. 6.2 the design choices behind the representation adopted by the proposed methodology are ablated. A two-stream CNN relying on an *explicit* temporal

Table 6.2: Ablation Study of Training and Inference Optimisations

	Model Configuration	**RMSE** (Test) *(meters)*	**Latency** *(ms/inf)*	**End-to-end rate** *(fps)*
(i)	Single-stream	0.182	14.57	22.87
(ii)	Vanilla Two-stream	0.094	19.32	17.25
(iii)	+ Weight-sharing	0.049	16.92	19.00
(iv)	+ Feature Memoisation	0.049	15.10	22.07
(v)	+ Window Batching	0.049	9.08	36.71
(vi)	Explicit[1] Representation [435]	0.045	51.25	6.50
(vii)	Implicit[2] (w/out pretraining)	0.109	9.08	36.71
(viii)	Implicit[2] (with pretraining)	0.049	9.08	36.71

[1]Baseline two-stream model with [frame(t) | $Farneback(t, t-1)$] as input.

[2]Proposed two-stream model with [frame(t) | frame($t-1$)] as input.

feature representation is used as a strong baseline (row (vi)). Following the methodology of the flagship action recognition work introduced in [435], optical flow is analytically calculated for each frame[12] and fed as an input to the additional stream.

The comparison indicates that a vanilla-trained *implicit* representation (row (vii)) cannot fully exploit the added benefits of the provided temporal visual cue, given the scale of the examined dataset, although notably improving the latency compared to the single-frame baseline. However, when feature extraction is guided by the proposed self-supervised pre-training scheme (row (viii)), the two-stream model implicitly extracting a spatio-temporal representation is able to meet the accuracy of the strong baseline. The latter however, is severely slower at inference time, due to the excessive additional workload required for the explicit calculation of the dense optical flow expected as an input to its second stream; whereas the proposed approach managed to keep the single-frame inference latency at the same levels of a single-stream model.

It should be noted that optical flow can be estimated through a plethora of different methods, dense [433] and sparse [441], iterative [442], or DNN-based [437]. Consequently, the results in this subsection are by no means a one-to-one comparison between the two selected instances. Rather, the point to be made is that implicit representations are also able to capture strong temporal information for the UAV-related task at hand, that can even be equally powerful to the explicit ones through the adoption of insight-rich training strategies, while demonstrating better efficiency and lower-complexity due to their single-model end-to-end learning paradigm.

[12]Optical flow is calculated once per frame, before cropping the three windows. This optimises the speed of the baseline by eliminating duplicate computation on the overlapping windows. At the same time, that design choice provides the baseline with enhanced context information for the extraction of temporal information.

Table 6.3: Comparison with Related Work on Vision-based Autonomous Navigation

Method	Mean time between collisions (s)	
	Seminar Room	Hallway
Straight Path Policy	4.8	9.2
Human Pilot	96.2	107.9
Classification-CNN [423]	27.4	38.2
Regression-CNN (proposed)	51.4	68.0

6.5.4 End-to-end UAV Navigation Comparison

In this section the proposed methodology is evaluated in an end-to-end manner on the task of autonomous UAV navigation. Experiments are conducted within two different indoor scenarios, namely *traversing a hallway* and *exploring a seminar room*, with real-world obstacles (such as chairs, boxes, poster-stands, bins etc) present in a realistic physical configuration. The UAV's motion is constrained within a fixed-altitude plane, with all methods essentially controlling the linear and angular velocities of the drone. The proposed approach is compared with:

- A Straight-Path navigation policy, acting as a *weak baseline* to offer a quantitative measure of the complexity of each undertaken task.
- A human pilot controlling the drone using a joystick, based solely on its forward-looking camera inputs (i.e. without having the drone on sight), acting as a *strong baseline*.
- A state-of-the-art learning-based visual navigation approach, also employing an AlexNet-based classification CNN, along with a hand-crafted motion planner [423], trained on the proposed dataset.

The above policies are compared in terms of the *mean time between collisions* while navigating autonomously for a flight time of 5 minutes in each scenario. The results of this comparison, listed in Table 6.3), demonstrate that the proposed approach outperforms the current state-of-the-art, managing to navigate without coming in contact with obstacles for 1.78× more time (on average) compared to the CNN classification-based work of [423]. The proposed methodology also reaches up to 63% of the collision-free flight time of a human pilot, and overpassing by up to 10.7× the weak baseline following straight-line trajectories.

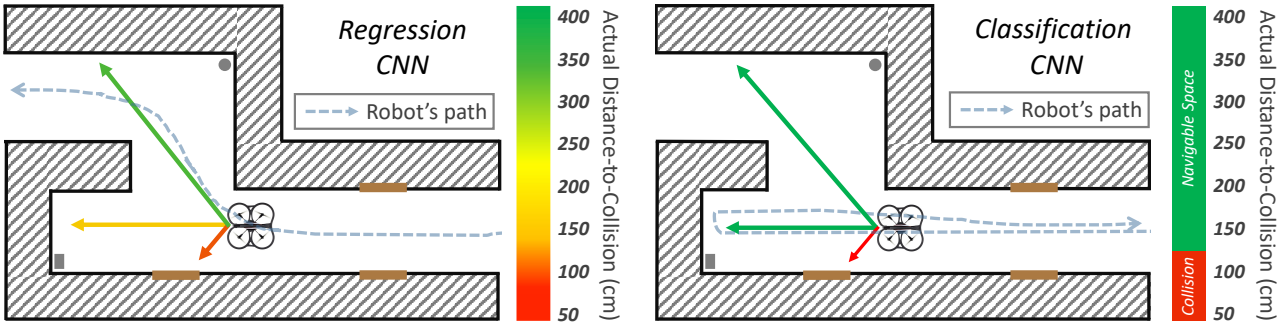


Figure 6.8: Case study comparison of regression- and classification-based approaches on a representative autonomous navigation scenario.

6.5.5 Qualitative Analysis

Navigation Policy

The superior navigation quality of the proposed approach is attributed to the fine-granularity of distance predictions offering more insightful environmental awareness, which is effectively leveraged by the proposed motion planner to yield more informed motion commands. Conversely, the classification-based approach demonstrates more impulsive navigational behaviour as result of the discrete nature of the model's outputs, leading to less smooth flight paths.

This section provides a representative case study that further demonstrates the benefits of employing a regression model to obtain fine-grained distance-to-collision predictions across a trajectory in the implicit navigation policy learned. In this context, the UAV is tasked to navigate autonomously in the environment depicted on Fig. 6.8, using both the classification [423] and regression CNN models. As illustrated by the flight-paths of both approaches, the proposed motion planner is able to exploit the richer distance information provided by the regression model, to make longer-range planning decisions, effectively considering the ratio of the predicted distances between the front and right direction, following a path towards the navigable corridor. In contrast, the classification based approach, although successfully avoiding collisions, made shorter-range planning decisions, based on the less informative collision probabilities, eventually getting trapped on the dead-end.

It is worth iterating that the proposed approach, also demonstrates enhanced post-training adaptability, as tunable distance thresholds can be adjusted on the planner without any need for fine-tuning. This renders the proposed methodology easily transferable across different target tasks (e.g. with different allowable distances to obstacles), as well as different UAV models (e.g. with

different dynamic behaviour, such as deceleration capability). In contrast, in order to modify the behaviour of the classification-based approach (e.g. re-defining obstacles as objects present in a larger distance, to resolve the issue of the previous example) would require a time-consuming and costly complete re-annotation of the dataset and re-training of the model.

Failure cases

From the experiments described in the previous sections, it is deduced that the proposed two-stream regression model generally manages to make more insightful decisions in cases of high ambiguity by utilising the learned, richer in information, spatio-temporal representation of the robot's state in its environment. In the extreme case that a *texture-less surface* covers the camera's field of view, resulting to a complete loss of features, the proposed model loses any reference of scale and is thus not able to predict distances-to-collision accurately. In such cases, a small distance value is consistently provided by the model due to its annotation bias (textureless inputs mostly occur when being in extremely close proximity to objects), that although inaccurate allows the motion planner to avoid collisions, making a safe navigational decision.

Additionally, in the case of flying in excessively close proximity to glass or highly-reflective surfaces, the accuracy of the distance prediction demonstrates a significant drop. This is attributed to the corresponding annotation error due to IR sensor failures in these cases. This can be remedied by selecting a larger minimum distance threshold d_L within the operating range of the Ultrasonic sensor that corrected such irregularities during the data collection process.

6.6 Discussion

This chapter focused on the data layer of the deep learning deployment stack where, in the context of vision-based robot navigation, approximations were employed to: (i) estimate distance sensor readings in view of solely visual inputs and (ii) incorporate an implicit representation of spatio-temporal features instead of explicitly calculating them upon deployment. The above contributions, enhance the applicability of autonomous navigation methodologies to a wider range of robot/UAV platforms without the need for expensive and heavyweight sensors, and offer the benefits of processing dense temporal information at a fraction of the computational cost, respectively.

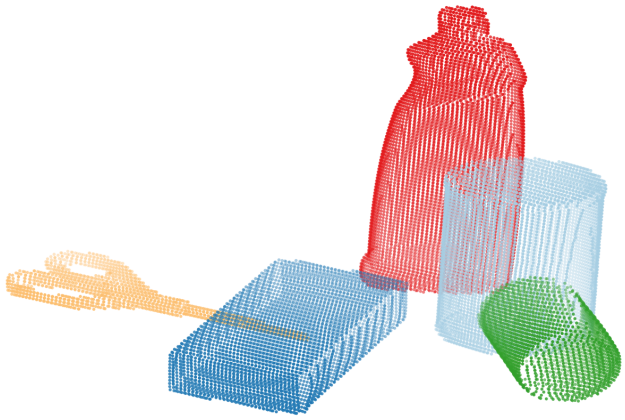
More specifically, a custom motion planning methodology for autonomous navigation was introduced, relying on distance-to-collision values towards diverging directions. This chapter showed

that such distance measurements can effectively be approximated by the use of a regression-based CNN, relying solely on visual input of a forward-looking camera, commonly available on commercial small-scale robots/drones. Training such a neural network typically requires an abundance of training data. To remedy this issue, a self-supervised data collection and annotation methodology is introduced, resulting to the creation of SelfNav: a large-scale real-flight dataset for UAV indoor navigation. The main motivation behind the proposed methodology is to minimise the need for human in-the-loop across the data collection and annotation process, drastically reducing the complexity and cost.

A thorough investigation was conducted to optimise the architecture of the proposed regression CNN, as well as its training methodology. A key finding is that by incorporating spatio-temporal representations for the task of visual distance-to-collision estimation, the predictive accuracy of the underlying model is significantly improved. Additionally, self-supervised pre-trained can be employed to initialise the CNN backbone towards extracting an implicit spatio-temporal representation in view of the RGB stream, eliminating the need to explicitly pre-calculating computationally-heavy data cues, such as optical flow. Furthermore, design choices such as weight-sharing and feature-map memoisation can significantly boost the inference efficiency of the resulting model upon deployment.

Real-world evaluation on a UAV demonstrated a considerable performance improvement across multiple indoor navigation scenarios by the proposed approach, compared to state-of-the art methodologies from the literature. Additionally, it was shown that the regression-based formulation of the visual navigation problem generates a more insightful prediction of the actual distance to the closest obstacle in the robot's path. This information-richer approach provides enhanced environmental awareness and enables the development of a more informed local motion planning policy, which not only allows it to conduct smooth collision avoidance manoeuvres autonomously, but also considers the longer-range traversability of space in the robot's environment to produce motion commands.

Not less important, by avoiding the incorporation of UAV- or application-specific characteristics/requirements in the data annotation and model training processes, the proposed approach remains flexible; through a number of tunable parameters allowing for post-training adaptation to fit the needs different navigation scenarios and/or robot/UAV platforms.



7

Seeing the Unseen:

A learning-based occlusion handling approach for 3D reconstruction

Contents

7.1 Overview 196
7.2 Motivation 197
7.3 Related Work 202
7.3.1 Visual Simultaneous Localisation and Mapping 202
7.3.2 Learnable 3D Reconstruction 203
7.3.3 Shape Completion and Occlusion Handling 204
7.4 Methodology 210
7.4.1 Overview 210
7.4.2 3D Shape Representation through Vis-à-Vis Camera Pairs 213
7.4.3 Occlusion Handling Model Design and Training 217
7.4.4 The *UnScene* Dataset 222
7.4.5 3D Reconstruction and Multi-view Fusion 224
7.5 Evaluation 230
7.5.1 Experimental Setup 231
7.5.2 Model Accuracy 232
7.5.3 Time-to-Accuracy Evaluation in 3D Reconstruction 234
7.5.4 Qualitative Analysis 237
7.5.5 Case study on Robot Grasping 243
7.6 Discussion 245

Keywords: *Encoder-Decoder CNN, RGB-D Data, Vis-á-vis Camera Representation, Depth and Silhouette Completion, 3D Reconstruction, Occlusion handling, Multi-frame fusion, UnScene Dataset, Robot Grasping*

7.1 Overview

*This chapter reaches the **task layer**, positioned on the top of the deep learning deployment stack. At this level, application-specific deep learning models can be used to approximate the output of the target task as a whole, in situations where it would otherwise be overly time-consuming to compute, or infeasible to accomplish due to other limitations. In this context, this chapter focuses on the task of building a detailed 3D reconstruction model of a scene, based on a RGB-D data stream. 3D scene models (representations), often capturing both geometric and semantic information, play an integral role on high-level robot applications such as object grasping and task-driven navigation. As real-world 3D scenes are only partially observable through a single camera view, traditional solutions actuate the camera and fuse information from multiple poses to a persistent 3D representation. In robotics, such approaches are typically based on visual Simultaneous Localisation and Mapping (SLAM). However, obtaining a complete 3D reconstruction through traditional SLAM can become severely time-consuming in cluttered real-world environments, where occlusion calls for a thorough camera scan of the scene, which can also be undermined by physical constraints and sensor noise.*

As such, this chapter introduces an occlusion-handling pipeline for 3D reconstruction, approximating the structure of unseen parts in partially observable objects, through a learnable approach. The proposed system is able to deal with both self- and inter-object occlusions through a unified formulation, offering time-efficient 3D reconstruction compared to traditional approaches. A new vis-à-vis camera representation is introduced, encoding the 3D shape of an object in the form of two occlusion-free depth and silhouette frames, captured from two opposite-view virtual cameras. At runtime, the proposed representation is predicted by a specially-trained encoder-decoder CNN model, in view of a real depth map for each partially observable object in the scene. By essentially hallucinating novel depth-camera frames for known camera poses, the proposed methodology is able to seamlessly operate as part of most RGB-D SLAM systems to date. Towards this direction, a novel multi-view fusion approach is proposed, that distinguishes between measured and predicted information, to enable consistent reconstruction and facilitate the long-range operation of the system. As a result, the proposed pipeline initially provides a rough approximation of the 3D reconstruction of a scene and progressively refines it in an informed way, as more real camera views become available. This approach improves Time-to-Accuracy (TTA) in 3D reconstruction compared to state-of-the-art approaches, while being able to capture approximations of otherwise unobservable information (e.g. due to limitations in camera motion).

In order to facilitate the development of the proposed methodology, the UnScene Dataset, a hybrid (real input, synthetic output) extension of the YCB-Video annotations offering occlusion-free depth and silhouette labels for each object, is created and will soon be made publicly available. Furthermore, incorporating the proposed 3D reconstruction methodology in-the-loop of a state-of-the-art object grasping pipeline demonstrated considerable improvement to the grasp-prediction quality, by exploiting its more complete scene representation.

The work presented in this chapter was jointly advised by Dr. Stefan Leutenegger and Prof. Christos Bouganis. A manuscript is in preparation for submission to a conference.

7.2 Motivation

Although several robot tasks such as autonomous exploration (Ch.6) can be performed relying solely on primitive perception [443], more complex embodied AI applications like target-driven navigation [411], object grasping [99] and manipulation [444], usually require more advanced state estimation, capturing a detailed 3D representation (map) of the robot's environment [445] along with its pose (position and orientation) within it. Consumer robot platforms, due to their form factor, payload, cost and energy constraints, typically rely solely on visual sensors to build such representations. Map-based approaches facilitate smooth, seamless and consistent interaction between the autonomous agent and its surroundings, by exploiting detailed information about the geometry, structure and semantics of the scene, as well as its relative state to the robot. Nonetheless, 3D shape is probably the richest property of an object, offering insights about its category, affordance and functionality [446]. As such, 3D representations have also proven to be equally useful in relevant domains, including Augmented and Virtual Reality (AR/VR) [447][448].

Simultaneous Localisation and Mapping (SLAM) [449] refers to the widely studied task of incrementally building a map of an unknown environment, while jointly tracking the *pose* of the sensing device within it [450]. The SLAM problem, being one of the fundamental challenges towards robot autonomy, has been formulated in a large number of different ways, making varying assumptions and offering very diverse solutions [451]. Visual SLAM [452], relying solely on cameras [453] and nowadays most commonly on RGB-D sensors [454], has been in the centre of the community's attention over the past fifteen years.

With the advent of deep learning for visual 3D understanding [133], SLAM approaches rapidly evolved [455] by incorporating both geometric and semantic information in a unified representation

[456], unlocking new potentials towards intelligent interaction between the robot and its environment. This progressively led to more complex but flexible object-centric [457] or hierarchical [458] representations reflecting the needs of the target application, while becoming able to handle dynamic scenes [459]. As a result, several restrictions on the scale and consistency of the environment have been waved, enabling the applicability of SLAM in a multitude of real-world usecases.

With applications such as grasping and manipulation heavily relying on a *complete* and *detailed* 3D representation of the scene, SLAM solutions can become excessively time consuming, both task-wise and computationally [460]. This is due to the extensive scan of the scene required (striking a wide variety of camera poses) and the burden of fusing the resulting multitude of high-resolution observations to a consistent multi-dimensional representation, respectively. These challenges are further aggravated when operating in real-world environments, where complex object geometries (Fig. 7.1a-top) and clutter (Fig. 7.1a-bottom) result to self- and inter-object occlusions, respectively; severely restricting the observability of the scene from each camera view. As a result, and in combination with sensing noise and outliers/artifacts, the underlying task-level Time-to-Accuracy and end-quality of the reconstruction process is undermined [461].

Recent literature resorted to deep learning for exploring learnable alternatives for *3D reconstruction*, driven by the increasing availability of 3D model datasets of objects [462] and scenes [463][464] and emergence of 3D perception models [465]. Commonly, these approaches rely solely on RGB data, and employ a DNN to predict dense depth information in the form of depth-map [466], point-cloud [467], volumetric [468] or surface [446] representations. The aim of such approaches is to entirely replace traditional 3D reconstruction methodologies through learnable counterparts. As a result, the ability to incorporate measured depth data (when available) is lost, leading to inferior reconstruction accuracy and limited generalisation capabilities, while restricting their applicability in conjunction with well-established SLAM systems for multi-view reconstruction.

Towards the same goal of obtaining a complete and detailed 3D representation of a target scene, a growing body of work explores learnable *3D completion* techniques, at the object [469] or scene level [456]. Such methodologies typically operate on measured 3D representations, being directly applicable to SLAM pipelines. Their aim is to apply a learnable refinement on the reconstruction, to complete missing information caused by sensor noise, occlusion or incomplete scanning. Although proven to provide accuracy benefits on downstream tasks [470], commonly these methodologies rely on 3D CNN architectures [471], tailored for multi-dimensional

spatial data [472]. However, due to the cubical memory complexity of such representations and corresponding computational and memory overhead of 3D CNNs, the scale and/or resolution of the reconstruction is heavily restricted, while the online deployability of the proposed approaches on resource-constrained environments is limited. Instead, such methodologies are frequently used as an offline post-processing step [473], aiming to increase the quality of the obtained 3D map; under the assumption of a static environment.

In real-world settings, apart from high accuracy, the applicability of 3D completion methodologies in tasks such as robot grasping [474] and multi-object tracking [475] also requires *near real-time* inference latency, to enable the agent to timely react to changes in the environment. To remedy the challenges of the above approaches and enable efficient runtime occlusion-handling, recent work took inspiration from literature on amodal perception[1] [476][477] and object inpainting [478]. The resulting methodologies aim to represent 3D information into 2D (image-like) representations, such as Layered Depth Images (LDI) [479] or Spherical Projections [480], and exploit computationally- and data-efficient commodity vision CNN architectures to predict unseen 3D shape information. Relevant work has proven this concept, focusing on efficient handling of either inter-object occlusions at a scene level [481] or self-occlusions at an object level [482], through generative formulations.

Scene-level approaches, however, still suffer from hallucination challenges causing the creation of artifacts on depth predictions [483], while object-level methods typically rely on strong symmetry [484] or convexity [482] assumptions and fail to capture global context and resolve inter-object occlusions. Furthermore, such occlusion-handling models are mainly trained on synthetic data, due to the inability of sensors to obtain real-world annotations. At the same time, the every-pixel nature of their predictions, limits the opportunities for applying domain-randomisation methodologies [485], with current approaches residing on photo-realistic rendering methods [486], leaving a heavily un-bridged sim-to-real gap. Nonetheless, current literature in this direction has focused on single-view reconstruction, leaving several open challenges on multi-view fusion for persistent 3D reconstruction through SLAM, as noise/errors on inpainted pixels may contradict future measured information, breaking the consistency of the map.

Inspired by human's ability to hallucinate occluded parts of objects and complete their geometric model, this chapter introduces a new formulation for unified handling of both self- and inter-object occlusions, in the context of 3D reconstruction. The proposed method, focuses on table-top

[1]The term *amodal perception* refers to the task of perceiving the whole shape of an occluded physical structure.

Figure 7.1: Illustrative comparison between (a) a real perspective depth measurement (b) a virtual perspective camera with LDI/thickness representation and (b) the proposed orthographic vis-á-vis camera representation; for the case of *self-* (top) and *inter-* (bottom) *object* occlusions.

scenes, and employs an object-centric and view-dependent pipeline. Objects of interest are initially cropped-out of each RGB-D observation, based on the predictions of a segmentation model, and projected through 3D geometry to a novel object-centric depth and silhouette representation. In this domain, a learnable module independently completes the silhouette and front-view depth and predicts opposite-view depth of each object, effectively resolving occlusions in the 3D space through an efficient CNN architecture borrowed from dense 2D tasks. The model's predictions reassemble an object-centric (virtual) vis-à-vis camera pair (Fig. 7.1c) being co-linear to the real camera's principle axis while adopting orthographic projections and depth standardisation.

In contrast to vanilla LDI (Fig. 7.1b), this formulation allows for direct pixel correspondences between all depth and silhouette layers of an object, while eliminating learning-processes complexities relevant to scale and alignment invariance and exploiting global context information. The proposed model is trained through a custom set of annotations, extending the real-world YCB-Video dataset with occlusion-free depth and object-mask labels, following the proposed representation. Furthermore, the proposed occlusion-handling approach can be seamlessly integrated to SLAM pipelines, in the form of "revised" camera measurements from known poses. This allows providing a rapid approximation of the 3D reconstruction of a scene, and progressively refining it as more views become available. For this purpose, a novel fusion methodology is introduced, enabling the informed accumulation of multi-view data, distinguishing between observed and hallucinated information, to aid consistency and long-term operations of the underlying SLAM system.

The main contributions of this chapter can be summarised as:

- A unified self- and inter-object occlusion-handling methodology, based on an object-centric depth and silhouette inpainting CNN model, trained to operate on a novel vis-à-vis camera representation, meeting near real-time constraints.
- A multi-view pipeline for 3D reconstruction, fusing the measured and predicted depth and silhouette cues to a persistent 3D representation that is progressively refined in an informed way, as more observations become available. The proposed pipeline can be easily integrated to SLAM approaches, to improve task-efficiency on downstream robot-vision applications.
- The creation of the *UnScene* dataset, an extension of YCB-Video comprising hybrid (real input, synthetic output) occlusion-free object-centric depth and silhouette annotations.

7.3 Related Work

Since the early years of Computer Vision research, recovering 3D shape information from one or more 2D projections (images) has been one of the fundamental challenges in the field [487][131]. As outlined below, a variety of methods have been proposed to obtain, represent and augment such *3D reconstructions* under different assumptions, while the deep learning revolution also played a transformative role on the capabilities of such systems.

7.3.1 Visual Simultaneous Localisation and Mapping

In the simplest form of the 2D-to-3D information recovery problem, fundamental camera projection properties, such as the fact that during (translational) camera motion objects closer to the camera demonstrate larger displacement in their 2D projection compared to more distant ones, can be used to calculate pixel disparity maps [488] and subsequently infer the 3D structure of the scene through Structure-from-Motion (SfM) approaches [489][490], as well as the camera motion through Visual Odometry (VO) [491][492]. This information remains in a relevant scale, unless further knowledge about the actual camera movement is incorporated, e.g. in the form of ego-motion in Visual-Inertial odometry (VIO) [493][494] or baseline distance in stereo camera pairs [495].

During long-term operation, vanilla SfM yields a sequence of 3D reconstructions, which remain independent and sparse due to the limited observability of the scene from each particular view, and can only be fused offline by analysing the whole sequence [496]. Since mid-1980s, Simultaneous Localisation and Mapping (SLAM) aims to tackle reconstruction and odometry in conjunction, in an online manner through a joint formulation [449] that enables the fusion of multiple observations from different viewpoints to a persistent representation [133], while tracking the pose of the sensor within it by registering each observation [497]. Specifically, Visual SLAM refers to the task where a moving camera (mounted on a robotic agent in this context) jointly estimates its $SE(3)$ (Eq. 2.6) pose over time and progressively constructs a 3D model of its (previously unseen) environment [498]. The joint and iterative formulation of localisation and reconstruction remedies the issues of mapless approaches such as dead reckoning and visual odometry, which suffer from catastrophic drift on medium to long-range trajectories [499]. This is achieved through a combination of computer vision, geometry, graph theory, probabilistic estimation and optimisation methods found in full-blow SLAM systems, which are typically also equipped with loop-closure [500] and bundle adjustment [501] methodologies to enforce consistency and reduce drift.

Over the years SLAM solutions adopted the progress in camera technology and computer vision algorithms, as well as the increasing availability of computation power [502], to evolve from relying on monocular [453][503][504] to depth (RGB-D) sensors [454][505][506], from sparse [507] to dense tracking and reconstruction [508], from rigid [501] to deformable [459] object support, and from metric [509] to semantic [77][510], metro-topological graph [457] and hierarchical [511] representations, while encapsulating significantly more information and enabling a plethora of robot-vision tasks.

Deep learning has unlocked new potentials for SLAM, such as the ability to associate semantic concepts with the captured geometric information [512], facilitating the transition to scene understanding, environmental awareness and task-level planning for robot agents [513]. This has been driven by the ability to handle dynamic objects by separating moveable and static objects [514][515] or establishing object-centric representations [516][517][518], with complex hierarchies [101][519] offering a tunable level of abstraction. The resulting reconstruction encodes the information of the world that is important to the target task, at the appropriate granularity (also called task-driven perception paradigm [502]). Along these lines and most relevant to this chapter is *object-level SLAM*, that focuses on the reconstruction and tracking of specific objects that are directly related to the underlying application. In this paradigm, objects-of-interest are first detected by a DNN-based model, segmented out of the original scene and individually reconstructed [520], with their relative spatial relations captured in a graph structure, as in Fusion++ [457]. Multi-view integration is supported by selectively and independently fusing information on each object's dedicated representation, while dynamicity in the environment is solely affecting the relative poses between objects, leading to unprecedented scalability and efficiency.

7.3.2 Learnable 3D Reconstruction

SLAM-based 3D reconstruction requires a mobile camera sensor to provide a comprehensive scan of the target scene. In many cases, obtaining a complete reconstruction through this process relies on the assumption that control of the camera trajectory is (to a degree) provided to the agent, and that the majority of camera poses are kinematically reachable. As these assumptions are not satisfied by the majority of computer vision applications, a parallel line of work explored (learnable) alternatives to predict 3D representations directly from a single monocular camera frame.

Early works on *single-view monocular 3D reconstruction* exploited the analytical properties of visual metrology [521], usually relying on strong assumptions about object structure and

alignment, such as symmetry [522] and adoption of canonical camera poses [522]. Deep learning allowed to partly waive such assumptions through learnable solutions, adhering to different design choices relevant to the selection between volumetric [468], surface [523] and point-cloud output [467] representations, being scene- [524] or object-centric [446], acting on a canonical [525] or arbitrary [483] camera frame, and focusing on category-specific [526] or -agnostic [527] objects. These methodologies aim to replace the whole 3D reconstruction pipeline with a learnable counterpart. In their current form, however, the above approaches rely solely on RGB data and fail to incorporate and exploit any depth observations available through RGB-D sensors, leading to limited generalisation and subpar reconstruction accuracy, especially when scaling the task to scene-level view-dependent reconstruction.

Additionally, the above methods are mainly focused on single-view observations, with only a few works providing a formulation for *multiple-view monocular 3D reconstruction*. The approaches of [528][529][530] form representative examples, focusing on single-object reconstruction with the ability to accumulate information from different viewpoints. Even so, the proposed methods remain offline and require joint optimisation of all poses; or in the case of [531] employ a stateful 3D-LSTM model to handle fusion of a limited number of RGB object views. Due to its excessive memory requirements, however, the latter approach is only applicable on very small volumetric representation spaces. Orthogonally, relevant methods aim to fine-tune CNN models to construct an internal representation of the whole scene [532], or generate novel views [533].

Overall, the above approaches are built under the assumption that a single image or one-off scan of the scene is available, and provide solutions that remain incompatible with the online and iterative process of a SLAM pipeline, where multi-view information is progressively fused to a persistent representation at scale. Furthermore, in many of the above cases the underlying prediction task assigned to the learnable model is *generative* by nature, introducing significant challenges in training and generalisation and heavy reliance on synthetic datasets [462] that often introduce a distribution shift to the real-world data.

7.3.3 Shape Completion and Occlusion Handling

In real-world machine vision applications, where 3D reconstruction forms an intermediate (or even the end) goal of the underlying task, while RGB-D data are frequently available, obtaining a complete 3D reconstruction through traditional visual SLAM remains excessively time-consuming.

This is attributed to *self-occlusions* between the front and rear view of each object, as well as clutter of real-world scenes causing complex *inter-object occlusions* between frontal (occluder) and backward (occludee) objects, that require a large number of diverse camera poses to resolve, aggravating the complexity of the scanning process [461]. Even so, in some cases, a comprehensive scan with the robot circumnavigating the scene [533] may not be possible due to physical constraints on camera reach, such as walls or other obstacles obstructing the robot motion on specific poses, leading to *partial observability* of the scene. Additionally, RGB-D technology is far from perfect, suffering from *sensor noise* for example in cases of transparent or reflective objects [534]. Consequently, these challenges often lead to incomplete 3D reconstruction, even when numerous *multi-view RGB-D observations* are available. Different approaches have been proposed to address this challenge across the 3D and 2D domains, with several representative cases described in this section.

Targeting volumetric 3D representations

Since their early days, SLAM systems pursued to estimate the structure of unseen areas of the target scene, and incorporate such predictions to their 3D representation. One stream of works took the form of trying to fit shape priors into the observed data, essentially decomposing the 3D shape of the object [535]. At the extreme, works like [536][537][538] incorporate entire 3D CAD models of known objects to the map, essentially reducing the SLAM problem to pose estimation. This comes at the expense of generalisation to novel objects, as CAD models are often extremely abstract to meet the needs of detailed representations of specific objects, even within the same class, for downstream tasks such as object grasping and manipulation [470].

Approaches from a different line of work proposed the completion of 3D structures with partial 3D shape/scene information, aiming to recover a complete and detailed reconstruction by post-processing the obtained 3D representation, by essentially "filling in the gaps". Such works mainly target volumetric representations [539][469][456][540], applying a learnable refinement directly on the 3D volume to complete missing information, sometimes along with inferring semantics [473]. To achieve that, the above works rely on 3D CNN architectures [471][541] tailored for regular input representations of multi-dimensional spatial data, such as voxel grids [472]. However, the cubical memory complexity of such representations, along with the computationally expensive and weight-heavy 3D convolution operation involved, limit the scalability of these

techniques to detailed representations or large scenes, compromising the adopted reconstruction quality, while being impractical for online processing with realistic latency. As a result, these approaches are commonly applied as an offline post-processing step, rather than as part of an iterative SLAM-based multi-view pipeline; aiming to improve the end-quality of the obtained 3D reconstruction, under the assumption of a static environment. Although memory-efficient representations such as Octrees [542] have proven to be viable alternatives for 3D reconstruction [540][543], their irregularity over-complicates the feature propagation, prohibiting DNN-based approaches to directly consume such representations efficiently [544]. Nonetheless, apart from the computational and memory challenges, the large number of learnable parameters of 3D CNNs also make them prone to overfitting, unless vast training datasets are adopted. Once more, the solution to this is to resort to synthetic data domain, where adequate amount of data are available, at the expense of limited generalisation to the real-world and challenging adaptability to different environments, due to the cost of data generation and training process at this scale.

Image-space 2D approaches

Aiming to alleviate the computational and memory burden of 3D CNNs, along with the resulting reconstruction scale and resolution limitations and inference latency overhead, as well as their corresponding training and generalisation challenges, a few works proposed alternative formulations for complete 3D reconstruction, pushing the prediction of missing information to the primitive 2.5D representation of each captured depth map. This approach is relevant to the *amodal perception* task, that aims to resolve visual occlusions on RGB data and predict the unobserved appearance [478] or semantic [545] information, by inpainting the corresponding 2D RGB image or semantic mask representation to provide complete outputs from partially visible inputs. Similarly, the *amodal completion* task tackles object de-occlusion in a slightly different setting, where modal silhouette masks for each object are also provided as input [546]. This chapter generalises the amodal completion formulation to the 3D reconstruction problem, through a view-dependent and object-centric pipeline, independently resolving occlusions for every detected object-of-interest in a scene, as captured by a particular camera view.

An important benefit of amodal completion methodologies operating on image-like 2D input representations (either RGB, or depth maps and silhouette masks) is that they are able to re-use commodity CNN architectures for mainstream CV tasks, with minimal modifications. Due to

their remarkably better memory efficiency compared to their 3D counterparts (demonstrating squared vs cubical memory complexity), image-space representations can also adopt considerably higher resolution, leading to superior representation accuracy of spatial information, and thus more detailed reconstruction even on thin objects where coarse volumetric representations completely fail [446]. However, owing to the augmented information on the de-occluded output, vanilla 2D depth and silhouette representations, comprising a single view-dependent depth and class annotation for each pixel, cannot adequately represent the desired output.

To remedy this challenge during *self-occlusion handling*, MarrNet [547] introduced a hybrid deep learning architecture, comprising a traditional CNN encoder followed by a 3D CNN decoder. As such, the proposed shape estimation model receives observed or learnably approximated depth, silhouette and surface normal information as input, in the form of 2D channels, and directly predicts a volumetric 3D representation of the target object. Due to the 3D decoder part, however, this approach remains prone to all the issues coming with the use of 3D CNN architectures. Instead, GenRe [548] adopts an alternative spherical map representation [549] of the observed or estimated depth map. This image-like representation is subsequently processed by an inpainting CNN model to recover missing information, with its output being equivalently represented on the same spherical map. Although this representation is able to capture geometric information of both visible and non-visible surfaces in a 2D output format, due to projection challenges along the sphere's radius the provided reconstruction remains incomplete. To remedy this, GenRe relies on post-processing the resulting reconstruction by a 3D CNN operating on the voxel domain, to effectively deal with self-occlusions. Furthermore, the above works operate on single objects, assuming a well-centred and unobstructed frontal camera view, consequently leaving the challenge of inter-object occlusion handling unexplored.

Layered Depth Images (LDI) comprise a more intuitive output representation for the reconstruction task at hand, inspired by the field of computer graphics [479], addressing many of the above challenges. LDI essentially stacks more than one depth-map channels associated to each RGB image, progressively capturing different layers of depth information in the scene, with each channel denoting deeper intersection between the camera rays and object surfaces (Fig. 7.1b). Due to its simplicity, the LDI representation is adopted by several learnable 3D reconstruction approaches in relevant literature. The work of [483] for example, employs a CNN to directly predict an LDI representation from RGB-only inputs, offering a view-dependent scene-level

prediction of 3D structure; while [466] is also able to de-occlude the appearance (RGB) cue too, extending LDI to a holistic Layered Scene Representation. When RGB-D data are available through depth measurements or prediction, the frontal depth layer of LDI is actually observable. Hence CNN-based approaches can be employed to predict the subsequent depth maps of this representation, dealing with *inter-object occlusions* as in [481] that successfully in-paints the occluded areas of background depth. The main benefit of these approaches is their ability to re-use widely studied and optimised encoder-decoder segmentation CNN architectures [6] due to the similarity in the structure of their input and output domains, while being able to provide dense and complete 3D representations. However, learnable scene-level LDI-based approaches are susceptible to creating object artifacts as the number of depth channels increases. Furthermore, their heavy reliance on RGB inputs leading to limited generalisation capabilities, while the every-pixel nature of scene-level reconstruction outputs does not facilitate the applicability of domain adaptation methodologies, as in the case of object-centric tasks [550].

X-Section [482] introduces a relevant formulation for view-dependent 3D-reconstruction, decomposing the scene to a set of detected objects of interest, that are initially masked out of the input image and then processed independently by a thickness[2] prediction CNN. By appending the predicted thickness channel to the measured depth map, a 2-layer LDI representation effectively resolving self-occlusions is formed, for each object-of-interest. Additionally, a modified Truncated Signed Distance Function (TSDF) representation with the ability to capture the notion of thickness along each ray, is introduced in order to facilitate re-projection of the de-occluded object shapes into 3D representations, that can then be combined to compose a scene-level reconstruction. This approach demonstrates enhanced accuracy on self-occlusion handling independently of the scene, under the assumptions of convex objects at each camera view (i.e. each camera ray intersects exactly two times with the object's surfaces). Additionally, the adopted formulation heavily relies on obtaining unobstructed observations of the objects' front view and completely deprives context information in the object's environment, making it inconceivable to deal with inter-object occlusions.

Open Challenges

Across formulations, CNNs are set out on a quest to hallucinate unobserved geometric information from partial observations. Expectedly, ground truth annotations for such a dense and ambiguous

[2]Thickness is defined as the depth span between the first and next surface of a 3D object, as intersected along a camera ray.

task are laborious to obtain and thus excessively costly, hard to scale and subject to noise and biases [551]. At the same time, injecting artificial obstructions on complete data to facilitate a self-supervised formulation, although effective in some tasks [476], can only cover a limited subset of realistic situations relevant to self- and inter-object occlusions in the wild. As such, relevant literature relies heavily on completely synthetic datasets for training amodal completion models [546], introducing an inevitable domain gap to real-world scenes. Instead, this chapter augments the widely used real-world YCB-Video dataset [126] with a set of synthetic, object-centric occlusion-free depth and silhouette annotations, obtained through camera rendering of well-aligned 3D models of the objects-of-interest, following the proposed representation. The newly created *UnScene* dataset, comprises synthetic annotations that remain aligned with real-world YCB-Video inputs, mitigating both the domain gap and data annotation challenges.

Another common challenge across the above works arises from the adoption of the perspective camera model for all geometry transformations between the 2D and 3D spaces. Notably, 3D shape recovery from 2(.5)D observations already constitutes an overly complex and ill-posed problem, as geometric hallucinations can be ambiguous by themselves, let alone the case of partial observability due to inter-object occlusions. As such implicitly incorporating geometric and camera projection models to the prediction task undertaken by learnable approaches further complicates the inductive process, in combination with the enhanced variability in the depth that each object can be met, evident in view-centric approaches (i.e. when not adopting canonical views). Instead, the proposed approach adopts a novel representation for the undertaken predictive task, comprising virtual object-centric orthographic camera pairs that can be trivially derived from real observations through 3D geometry based pre-/post-processing steps, mitigating the unnecessary representational complexities of end-to-end learnable approaches that undermine the learning process.

Finally, existing literature addresses either self- or inter-object occlusions independently, with the adopted representations impeding the direct generalisation of these methodologies to both challenges. It is also noteworthy that these work inherit several design choices from single-view 3D reconstruction literature. As such, although 3D representations independently obtained by applying these methodologies on different camera views can be naively fused to a single reconstruction, current methods completely overlook uncertainty aspects differentiating measured and predicted information, as well as multi-view consistency aspects that are necessary to secure the long-term operation of the mapping system. Instead, the proposed methodology uniformly tackles self-

and inter-object occlusions through a joint formulation. Additionally, a novel multi-view fusion approach is introduced enabling the robust integration to SLAM approaches, while preserving the ability to distinguish between observed and predicted depth information, facilitating the consistency of long-range 3D reconstruction frameworks.

7.4 Methodology

7.4.1 Overview

This chapter explores the concept of high-level task efficiency, in the context of detailed 3D reconstruction of real-world environments. More specifically, *given* a stream of RGB-D information, comprising a progressive camera scan of a table-top scene with the aim to incorporate the observed information on a persistent 3D representation; this chapter *introduces* a methodology that is able to deal with occlusions on objects-of-interest through an inpainting CNN, and provide a more complete approximation of the reconstruction early on, that gets progressively refined as more camera views of the scene become available (i.e. hallucinated information is replaced by more recent/real observations). As such, the Time-to-Accuracy of the reconstruction task at hand is improved; while the final representation is able to (approximately) capture geometric information that was never observed (e.g. due to incomplete scan or physical limitations). The operation of such a system implies its near real-time execution, as well as its seamless integration with multi-view mapping (SLAM) pipelines, with the ability to fuse predicted data in an informed way. Furthermore, the proposed pipeline is crafted to strike a good balance between 3D geometry-based and learnable approaches, aiming to reduce the complexity of the undertaken predictive task (that remains inherently ill-posed).

In more detail, the methodology introduced in this chapter adopts a modular structure, similar to X-section's pipeline [482]. As such, each scene observation is decomposed into a set of objects, that are independently reconstructed through a novel object-centric view-dependent formulation for occlusion handling, before being recomposed to a scene-level 3D representation. This process is visually illustrated in Fig. 7.2. The scene decomposition is conducted based on an off-the-shelf instance segmentation model [202], that predicts bounding boxes and semantic masks for each object instance that appears in the RGB input frame. The bounding-box predictions are used to crop the *depth* information of each object-of-interest out of the scene-level input and process it

independently, while the object *silhouette* is obtained by binarising the instance mask. As such, an object-centric view-dependent *depth and silhouette representation* is obtained, carrying all necessary spatial and geometric information for each object in the scene [552].

To deal with occlusions, a new vis-à-vis virtual-camera representation is introduced in Sec. 7.4.2. Under this scheme, the 3D shape of each object is represented by a pair of depth maps, captured by opposite-view *virtual cameras* along a shared principle axis. As illustrated earlier (Fig. 7.1), this representation shares common benefits with Layered Depth Images (LDI), in the sense that it remains within the 2D domain. However, the formulation adopted in this chapter is able to eliminate many limitations of existing literature: First, the proposed representation is able to handle both self- and inter-object occlusions in a unified and structured formulation. Additionally, the adoption of virtual orthographic cameras enables to preserve direct one-to-one pixel correspondences across all input and output channels, reducing the complexity of the occlusion-handling task, while preserving necessary context information on the input to guide the generalisation of the approach to inter-object occlusion handling. Finally, the adopted front- and opposite-view formulation is able to provide enhanced volumetric occupancy estimation, even in objects with complex geometry exploiting the compactness and efficiency of image-space representations.

Through 3D geometry it is trivial to transform real camera observations to the proposed representation. However, due to occlusion and partial observability of the scene from a single camera view, this representation initially remains incomplete. As such, a CNN is trained to undertake an object-level predictive task, essentially inpainting the front-view depth and silhouette channel to handle inter-object occlusions, and hallucinating the opposite (artificial) view depth map to resolve self-occlusions and remedy incomplete scene scanning. This process is detailed in Sec. 7.4.3. Since in the proposed representation the front and opposite camera views adhere to pixel-level correspondences, a shared object silhouette channel adequately represents both views. Furthermore, operating on the image space, the inference process can re-use widely-studied dense CNN architectures [197], benefiting from their impressive predictive capabilities and enhanced data and inference-time efficiency.

To facilitate the development of the proposed pipeline, a new dataset termed *UnScene* is created. As will be discussed in Sec. 7.4.4, *UnScene* enhances the annotations of the YCB-Video dataset [126] with occlusion-free object-centric annotations, adopting the proposed vis-á-vis camera representation. Furthermore, UnScene comprises a hybrid dataset, adopting

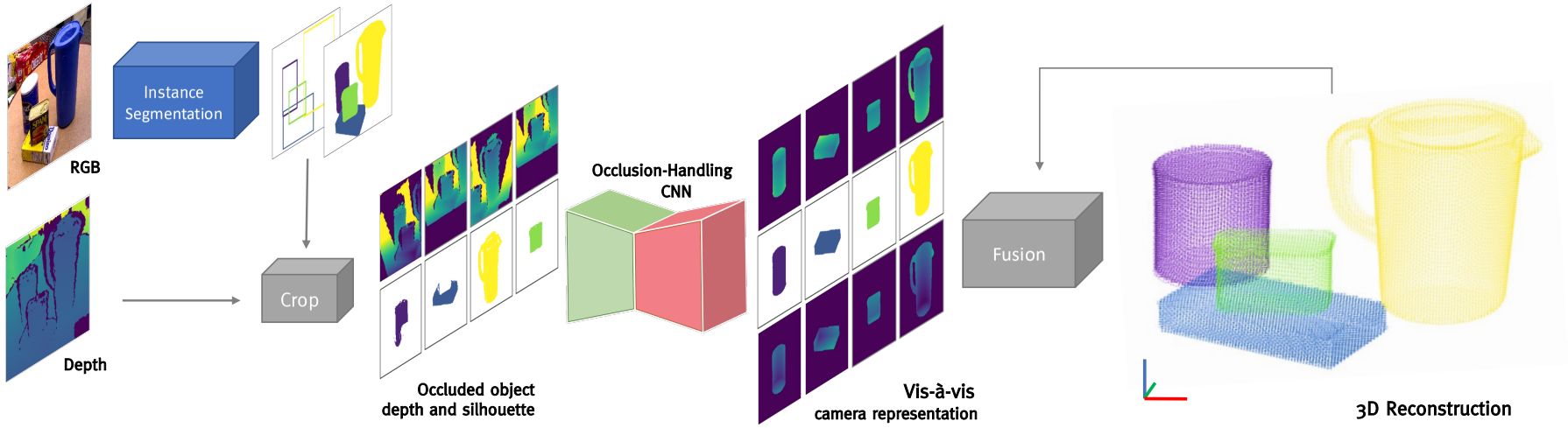


Figure 7.2: Proposed occlusion-handling pipeline for 3D reconstruction, adopting the proposed object-centric camera representation.

real-world RGB-D inputs and synthetic depth and silhouette outputs. This structure aids generalisation to real-world data, while tremendously reducing the complexity, and resulting noise and cost of the annotation process.

Finally, the proposed occlusion-handling system is designed to facilitate its adoption by most multi-view 3D reconstruction pipelines. Aiming to effectively support the integration of multiple camera-view information on a persistent representation [457], a tailored fusion approach is introduced in Sec. 7.4.5, distinguishing between measured and predicted information and taking into account multi-view consistency aspects of the reconstruction process, in an informed way.

7.4.2 3D Shape Representation through Vis-à-Vis Camera Pairs

Measured [454] or estimated [553] 2.5D data (RGB and Depth map) are able to capture the geometry of visible parts of objects in the wild, but lack the ability to recover occluded regions. To remedy this limitation and obtain a complete 3D reconstruction of the scene, a learnable approach is introduced in this chapter, aiming to predict the missing information of such partial observations. To support this task, a more complex representation alternative is required, able to portray the de-occluded output of the model.

As discussed in the previous section, due to computation and memory constraints of 3D representations and corresponding DNN models, image-space shape representations comprise an attractive alternative offering enhanced efficiency and reconstruction resolution. Even in the image domain, where CNN architectures have been widely studied offering impressive capabilities on dense prediction tasks, use-cases like amodal perception, 3D reconstruction and occlusion handling remain inherently ill-posed. Despite progress in the field, reconstruction accuracy remains subpar for many applications requiring highly-detailed shape representations of objects, unless strong assumptions are imposed on the adopted camera pose and object structure.

Many approaches formulate occlusion-free 3D reconstruction as a novel-view synthesis task, for appearance and/or depth [554][446][533][555], where virtual observations from artificial camera poses can be fused into a 3D representation. This is by nature a generative task, relevant to adversarial image synthesis [556][532], and requires the model to learn an implicit representation of the whole scene and render novel views by inferring camera pose and projection models. Although much simpler, the LDI representation [466][483] equivalently relies on implicitly learning a camera projection model, to be able to render deeper depth channels of hallucinated structure information.

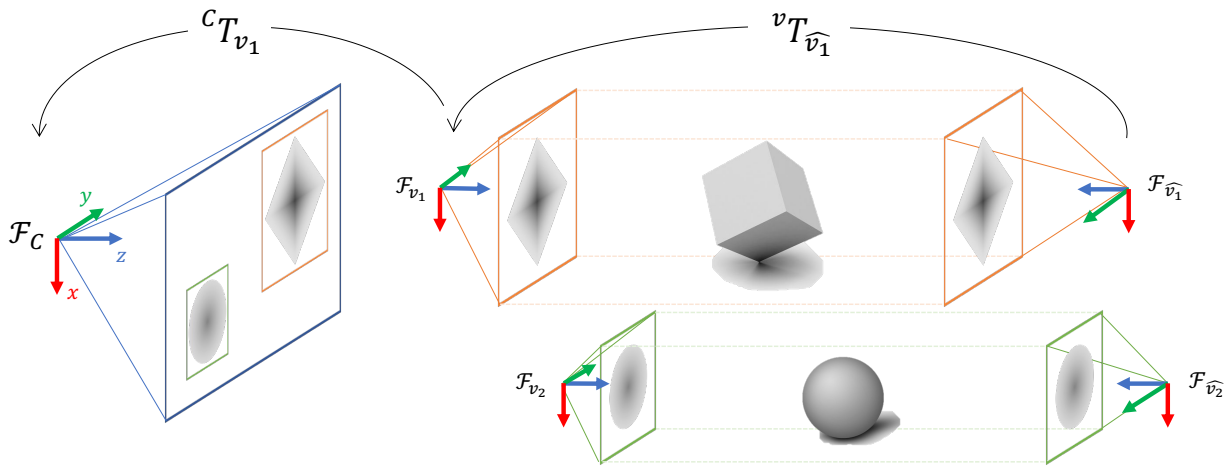


Figure 7.3: Illustration of the proposed vis-à-vis camera representation for occlusion handling.

This chapter strives to reduce the complexity of the occlusion handling task, by explicitly incorporating geometric and camera modelling in pre- and post-processing stages of the pipeline, to constrain the predictive task undertaken by learnable components. By eliminating the need to infer the camera pose and projection models arising from the adoption of perspective modelling, as well as robust scale and shift invariance caused by the view-dependent formulation, several unnecessary complexities are eliminated from the learning process and undertaken by other components, facilitating its predictive accuracy as well as data-efficiency and generalisability.

Although the overall pipeline remains view-dependent (i.e. preserving the original camera view $\mathcal{F}_C \in SE(3)$) and achieves scene-level reconstruction, a new vis-á-vis camera representation is introduced allowing the occlusion-handling model to operate on standardised object-centric camera views. For each object i detected in a camera frame by the adopted segmentation model, the proposed representation assumes a new pair of virtual opposite-view cameras $\{\mathcal{F}_{v_i}, \mathcal{F}_{\hat{v}_i}\}$, as depicted in Fig. 7.3. These artificial camera-views share the same principle axis, which is defined to be co-linear with the center of the observed object's bounding box. The distance between the two cameras d_v is fixed, while their poses are defined so that the two cameras are equidistant to the observed object surface. This reduces the depth and shift variability in which each object can be met, independently of its pose in the scene, simplifying the predictive task.

Consequently, in the proposed representation each object is represented by a pair of depth and silhouette frames, with well-defined camera poses. Apart from the enhanced simplicity of

this approach, this representation is capable to resolve both self- and inter-object occlusions (in contrast to LDI, thickness, ray-tracing or circular map representations), by hallucinating the opposite and inpainting the front view respectively (Fig. 7.1c). Most importantly, the *orthographic projection model* is adopted for both virtual cameras. This design choice ensures direct pixel-level correspondences across both camera frames, further reducing the complexity of the predictive task across several dimensions. First, the complex parameters of the perspective projection model are no longer needed to be inherently learned (and satisfied) by the model predictions. Furthermore, front and opposite view predictions are tightly coupled, offering more coherency and sharing a common silhouette mask. Not less important, input and output channels are aligned, allowing for direct propagation of the observed information from the input to the output, as well as the exploitation of context information. This facilitates the case of inter-object occlusions, where the depth and spatial information of the occluder object that is visible in the input becomes available to the model during inference, aiding the occludee inpainting process. Finally, the virtual cameras can model translational motion across their z-axis to adopt a standardised object-centric view, by simply applying a distance bias term across their observed depth values, facilitating the standardisation of the input depth, eliminating further ambiguities from the training process.

More formally, given an input image $\mathbf{I} \in [0, 255]^{H \times W \times 3}$ and corresponding depth map $\mathbf{D} \in [0, 255]^{H \times W}$, captured from a camera C modelled by the perspective projection model with intrinsic matrix $\mathbf{K}_C$ and pose $\mathcal{F}_C$, the adopted instance segmentation model $f_{seg}(\cdot)$ outputs a set of object predictions in the form of:

$$\{\mathbf{b}_i = [x_i, y_i, h_i, w_i], \mathbf{p}_i, \mathbf{M}_i\}_{i=1}^{N} = f_{seg}(\mathbf{I}) \tag{7.1}$$

where (x_i, y_i) denote the top-left corner of the object bounding box in the image frame and (h_i, w_i) its height and width expressed in pixels; $\mathbf{p}_i$ is a vector containing the probability distribution of the detected object across all possible object classes and $\mathbf{M}_i \in \{0, 1\}^{H \times W}$ a binary semantic mask with the same dimensionality as $\mathbf{D}$, for the i-th out of the total of N detected objects with detection score above a tunable threshold $\max(\mathbf{p}_i) \geq p_{th}$.

In the proposed representation (Fig. 7.3), a pair of dedicated virtual cameras $\{v_i, \hat{v}_i\}$ is generated for each detected object i and placed at $\{\mathcal{F}_{v_i}, \mathcal{F}_{\hat{v}_i}\}$ respectively. The proposed vis-á-vis representation comprises depth frames, virtually "catpured" by this camera pair. More formally, each object is represented by two object-centric depth images $\{\mathbf{D}_{v_i}, \mathbf{D}_{\hat{v}_i}\} \in [0, 255]^{h_i \times w_i \times 2}$

accordingly. The transformation between the front-view virtual camera v_i and the actual camera C frame is derived, considering the object bounding-box information, as:

$$ {}^{c}\mathbf{T}_{v_i} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & (x_i + h_i/2 - c_x) \cdot (\overline{(\mathbf{D} \odot \mathbf{M_i})} - d_v/2)/f_x \\ 0 & 1 & 0 & (y_i + w_i/2 - c_y) \cdot (\overline{(\mathbf{D} \odot \mathbf{M_i})} - d_v/2)/f_y \\ 0 & 0 & 1 & \overline{(\mathbf{D} \odot \mathbf{M_i})} - d_v/2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{7.2} $$

where f_x, f_y and c_x, c_y are extracted form the intrinsic matrix of C (Eq. 2.2); $\overline{(\cdot)}$ denotes the calculation of the $median(\cdot)$ function, and $\odot$ the element-wise multiplication operation. As such, the front-view camera adopts the same viewing direction with C, while it's principle axis passes through the center of the object's bounding box as back-projected in the 3D space, and stays $d_v/2$ meters away the median object depth, as observed from this new viewing pose.

Accordingly, the pose of the opposite virtual camera $\hat{v}_i$ can be expressed with respect to its front-view counterpart as:

$$ {}^{v_i}\mathbf{T}_{\hat{v}_i} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & d_v \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{7.3} $$

essentially sharing the same principle axis with the front-view camera, while posing towards it and being d_v meters further away from the original camera C.

Both virtual cameras follow the orthographic projection model, and share the same intrinsic matrix denoted as $\mathbf{K}_o$. Consequently, the observations from the front- and opposite-view virtual camera can be transformed to be directly represented by equivalent depth maps captured from the original camera pose, through the following transformation chains:

$$ {}^{c}\mathbf{D}_{v_i} = \pi({}^{c}\mathbf{T}_{v_i} \cdot \pi_o^{-1}(\mathbf{D}_{v_i})) \quad \text{and} \quad {}^{c}\mathbf{D}_{\hat{v}_i} = \pi({}^{c}\mathbf{T}_{v_i} \cdot {}^{v_i}\mathbf{T}_{\hat{v}_i} \cdot \pi_o^{-1}(\mathbf{D}_{\hat{v}_i})) \tag{7.4} $$

where the virtual-camera depth map is essentially first back-projected ($\pi^{-1}(\cdot)$) to a point-cloud representation, and then re-projected ($\pi(\cdot)$) to the new camera angle; through the respective projection models, and intrinsic parameters of each camera.

In the context of this chapter, focusing on table-top scenes (as the one in the earlier example of Fig. 7.2), the distance between the two cameras is set to $d_v = 2m$. By tuning the scaling factor s of the orthographic projection model (Eq. 2.4), the camera frustrum can be adjusted to capture arbitrarily large objects, as long as their depth fits between the two virtual cameras. As expected, a trade-off exists between the area and level of detailed captured by each frame. This representation

budget can however be increased by adopting higher resolution images in each camera, at the expense of additional memory and computation requirements. A potential extension of the proposed methodology on room scale scenes would require an appropriate adaptation of these values.

It is noteworthy that this representation is still a "single-frame" approximation of the 3D shape, since the internal structure of concave objects with complex self-occlusions cannot be fully recovered in either frame. Nevertheless, in contrast to 2-layer LDI representations as the one of X-Section [482], the proposed vis-á-vis camera representation effectively prioritises the reconstruction of outmost object shell, that is more informative for task such as grasping and occupancy estimation (Fig. 7.1). Notably, during multi-view reconstruction, resolving self-occlusions requires antipodal camera views, which take up the longest camera trajectory of circumnavigating the object. It is therefore fair to assume that concavity-caused occlusions are usually easier to resolve in multi-view settings.

7.4.3 Occlusion Handling Model Design and Training

Having re-formulated occlusion-handling as a predictive task from a 2D object-centric depth and silhouette representation to the proposed vis-á-vis camera representation, a CNN-based architecture is employed to undertake this prediction. This section discusses the design and training process of the proposed model.

Depth and Silhouette Inpainting CNN Architecture

Given the input depth map $\mathbf{D}$ and silhouette masks $\{\mathbf{M}_i\}_{i=1}^{N}$ for each object of interest obtained after processing the input image through the segmentation model (Eq. 7.1), occlusion-handling is formulated as the predictive task of obtaining the object-centric front opposite depth maps of the proposed vis-a-vis representation, denoted as $\mathbf{D}_{v_i}$ and $\mathbf{D}_{\hat{v}_i}$ respectively. Since these camera views are not directly observable, the proposed system solves this problem independently for each object, through a CNN, offering corresponding approximations of the two views, denoted as $\tilde{\mathbf{D}}_{v_i}$ and $\tilde{\mathbf{D}}_{\hat{v}_i}$.

Similar to other dense prediction tasks [199], an encoder-decoder CNN architecture is employed. As depicted in Fig. 7.4, the proposed model follows the U-Net architecture [197], featuring symmetric encoder and decoder blocks. Initially the encoder acts as a feature extractor reducing the spatial dimensionality of the input depth and silhouette to an three-dimensional embedding. The encoder blocks are adopted from a ResNet-50 backbone [3], with the exception of the first layer which is slightly modified to accommodate the reduced number of input channels (depth and silhouette

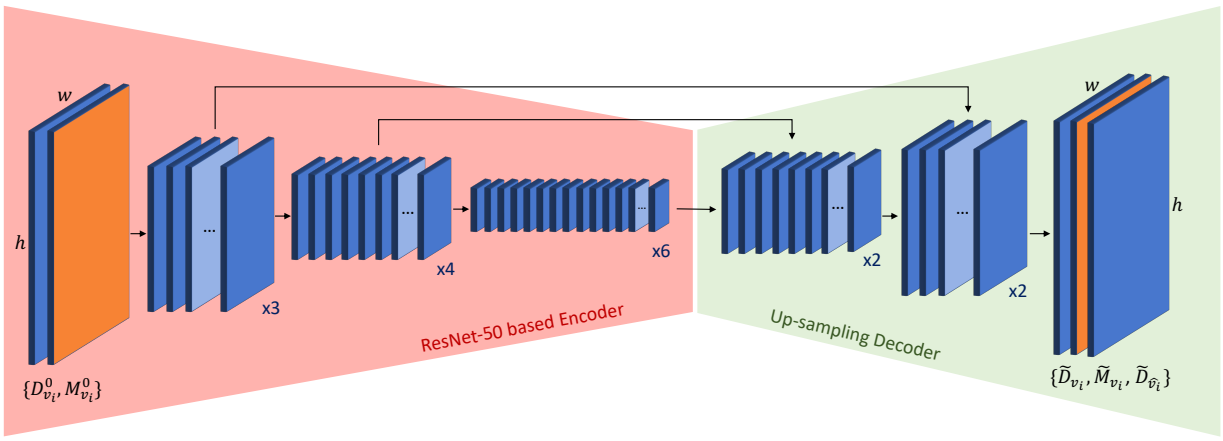


Figure 7.4: Architecture of the encoder-decoder CNN model for object-centric occlusion handling.

(2), instead of RGB (3)). The decoder comprises a corresponding number of custom blocks, initially conducting a spatial bi-linear up-sampling operation on the feature volume by a factor of two, followed by element-wise accumulation with skip-connection feature maps from the corresponding resolution encoder block, and a pair of 3x3 convolutional layers with a ReLU non-linearity.

Intuitively, skip connections propagate the observed information from the input towards the output, taking advantage of the one-to-one pixel correspondences offered by the adoption of the orthographic projection model; while learnable layers undertake the inpainting task of occlusion-handling. During training, this implicitly enforces consistency between the observed and predicted values on visible areas of objects, acting as a regularisation factor across the depth and silhouette channels.

The inputs to the model are extracted from the observed depth map $\mathbf{D}$ and predicted silhouette mask $\mathbf{M}_i$, independently for each object. As naive cropping would violate the perspective projection properties of the real camera, a sequence of transformations is undertaken instead. The aim is to obtain the equivalent (occluded) observations through the lens of the front virtual-camera, which is derived according to the bounding-box coordinates of the corresponding object $\mathbf{b}_i$ based on Eq. 7.2. As such, the occluded depth and silhouette observations from the real camera can be re-projected to the virtual-camera pose, as:

$$\mathbf{D}^0_{v_i} = \pi_o \left({}^{v_i}\mathbf{T}_c \cdot \pi^{-1}(\mathbf{D}) \right) \tag{7.5}$$

where the real-to-virtual camera transformation matrix can be calculated as the inverse of the virtual-to-real transformation (${}^{v_i}\mathbf{T}_c = {}^{c}\mathbf{T}^{-1}_{v_i}$). Essentially, the observed depth-map $\mathbf{D}$ is

back-projected to a point cloud using the perspective camera model of the original camera and re-projected to the virtual front-view camera through an orthographic model. Due to the placement of the virtual camera, the resulting artificial observation is object-centric, but still preserves context information from its surrounding objects. A corresponding transformation is conducted to the object mask:

$$\mathbf{M}_{v_i}^{0} = \mathbb{1}\left(\pi_o\left({}^{v_i}\mathbf{T}_c \cdot \pi^{-1}(\mathbf{D} \odot \mathbf{M}_i)\right) \neq 0\right) \tag{7.6}$$

which is additionally post-processed by an indicator function $\mathbb{1}(\cdot)$ to get re-binarised. In this way, an object-centric depth and silhouette representation $\{\mathbf{D}_{v_i}^{0}, \mathbf{M}_{v_i}^{0}\}$ for object i, expressed with respect to the front virtual camera frame $\mathcal{F}_{v_i}$ is obtained.

Notably, the virtual camera v_i adopts an image plane with the same dimensionality as the respective object bounding box, enhanced by a tunable zero padding of p pixels to facilitate the prediction of occluded object regions outside of the originally observed bounding-box area. As such, the virtual camera frames are dimensioned as $h_{v_i} \times w_{v_i} = (h_i + p) \times (w_i + p)$. This object-centric depth and silhouette representation remains occluded, and is provided as a 2-channel input to the proposed de-occlusion model, that estimates the corresponding (de-occluded) depth and silhouette frames, along with a novel opposite-view depth map, as a 3-channel output:

$$\left[\tilde{\mathbf{D}}_{v_i}, \tilde{\mathbf{M}}_{v_i}, \tilde{\mathbf{D}}_{\hat{v}_i}\right] = f_{occlusion}(\left[\mathbf{D}_{v_i}^{0}, \mathbf{M}_{v_i}^{0}\right]) \tag{7.7}$$

Importantly, due to its fully-convolutional architecture, the proposed model is able to deal with arbitrary input-image resolutions. Additionally, the adoption of orthographic projections and depth standardisation facilitates enhanced generalisation capabilities, by limiting the variability on the size and scale of objects appearing in the wild, and disentangling complex camera projection modelling from occlusion-handling. Furthermore, due to the CNN's encoder-decoder architecture, the output of the model features the same spatial dimensions as the input, which preserves the same camera projection properties as the input. Finally, adopting the same input and output representation in a model, allows for more regular mapping, simplifying the predictive task, in contrast to alternative 2D-to-3D formulations.

Loss Function and Training Process

Assuming the availability of corresponding occlusion-free depth and silhouette ground-truth data (discussed in the next section), the proposed model is trained in a supervised fashion. Given the dense (every-pixel) format of the output, the training process re-assembles that of semantic segmentation models. However, custom loss terms are required for the predicted depth and silhouette channels.

Silhouette data comprise binary masks, hence their prediction can be viewed as a set of binary classification tasks (one per pixel). In this formulation, the silhouette loss term consists of the binary cross-entropy loss across all pixels, reduced through averaging:

$$\begin{aligned}\mathcal{L}_{silhouette}(\tilde{\mathbf{M}}_{v_i}, \mathbf{M}_{v_i}) &= BCE(\tilde{\mathbf{M}}_{v_i}, \mathbf{M}_{v_i}) \\ &= -\frac{1}{h_{v_i} w_{v_i}} \sum_{x=1}^{h_{v_i}} \sum_{y=1}^{w_{v_i}} \Big(\mathbf{M}_{v_i}(x,y) \cdot \log \tilde{\mathbf{M}}_{v_i}(x,y) + (1 - \mathbf{M}_{v_i}(x,y)) \cdot (1 - \log \tilde{\mathbf{M}}_{v_i}(x,y)) \Big)\end{aligned} \tag{7.8}$$

A regression loss term is also required for the prediction of the front- and opposite-view *depth* maps. An L1 loss term[3] is therefore applied for this purpose. As the predictive task follows an object-centric formulation, depth predictions are subsequently masked by the predicted object silhouette, to eliminate background information from the representation. Equivalently, only parts of the predicted depth map that correspond to the object-of-interest contribute to the depth loss term. During training this is implemented by masking the depth predictions with the ground-truth silhouette masks, to avoid a collapse of the training process.

$$\mathcal{L}_{depth}(\tilde{\mathbf{D}}_{\{v_i,\hat{v}_i\}}, \mathbf{D}_{\{v_i,\hat{v}_i\}}) = \frac{1}{||\mathbf{M}_{v_i}||_0} \sum_{v \in v_i, \hat{v}_i} \sum_{x=1}^{h_{v_i}} \sum_{y=1}^{w_{v_i}} \Big(\mathbf{M}_{v_i}(x,y) \cdot |\mathbf{D}_v(x,y) - \tilde{\mathbf{D}}_v(x,y)| \Big) \tag{7.9}$$

Finally, the depth and silhouette loss terms are combined to a single cost function by a weighted summation, and accumulated for all detected objects in a frame as:

$$\mathcal{L} = \sum_{i=1}^{N} \Big(\mathcal{L}_{depth}(\tilde{\mathbf{D}}_{\{v_i,\hat{v}_i\}}, \mathbf{D}_{\{v_i,\hat{v}_i\}}) + \alpha \cdot \mathcal{L}_{silhouette}(\tilde{\mathbf{M}}_{v_i}, \mathbf{M}_{v_i}) \Big) \tag{7.10}$$

where α denotes a configurable parameter. The process of calculating the loss terms for each channels is also illustrated in Fig. 7.5.

[3] Since L1 norm is not directly differentiable, while the common alternative of L2 can become destabilise the training process when many outliers are present in the data; a smoothing approximation of L1 was actually applied, in the form of: $||a||_{1+\epsilon} = \left(\sum_i |a|^{1+\epsilon}\right)^{\frac{1}{1+\epsilon}}$ for a vector a.

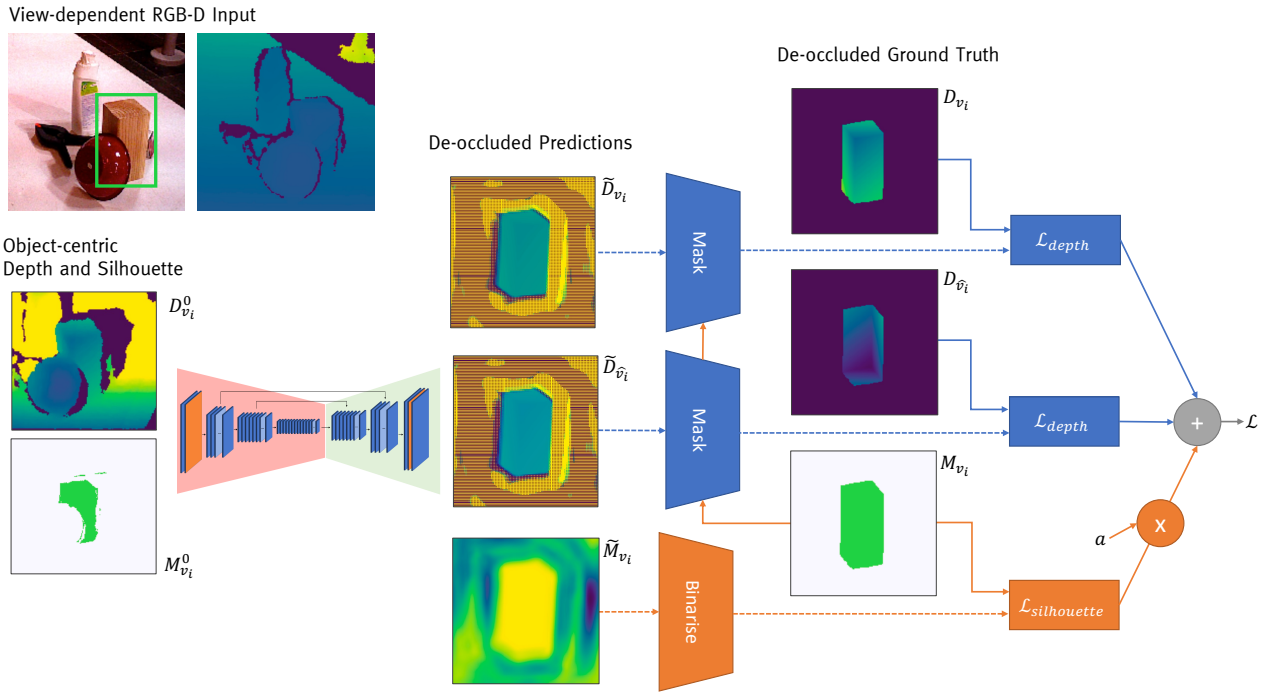


Figure 7.5: Calculation of depth and silhouette loss terms, for training the proposed model.

Notably, the opposite-view depth map is predicted in a vertically-mirrored representation. This is explicitly imposed by reflecting the ground-truth data during training. As a result one-to-one pixel correspondences are achieved between the two-views, simplifying the predictive task by eliminating the complexity of generative-type predictions. The adopted depth and silhouette input, offers pixel-level information about the shape of the object-of-interest leading to the adoption of a single shared silhouette mask across both views. Furthermore, by avoiding to explicitly mask the depth information on the *input* within the detected bounding box, context information is preserved. This is particularly important for the case of inter-object occlusions, as the model can exploit information about the existence and shape of occluder objects in the scene, by jointly processing the complete depth and object-specific mask.

Upon deployment, both predicted depth channels are masked by the *predicted* silhouette mask, and then the opposite-view prediction is reflected vertically back to its original form, corresponding to the opposite virtual-camera frame $\mathcal{F}_{\hat{v}_i}$.

Driven by the design choice not to mask out background information from the input depth maps, data augmentation is required to facilitate generalisation to different domains. As such, during a percentage of training epochs the object depth sketches are superimposed over randomly

selected depth frames from the NYU-V2 dataset [557]. This preserves the necessary context information hinting about inter-object occlusions, while cancelling out the overfitting effect of the repetitive background information, commonly met in table-top scene datasets.

Finally, although the proposed model is designed to operate separately on each object, all detected objects of an input image are used to form a batch, facilitating stability during training. Further zero-padding is required on some samples to enable this batching, given the variability of their spatial dimensions. Although a naive implementation of this approach introduces a computational overhead, the heavily structured nature of this padding allows for alleviating most of the computational burden through the use of out-of-the-box sparse convolution kernels [558]. Optionally, the same approach can be employed at inference time, if the efficiency gains from batching are justified.

7.4.4 The *UnScene* Dataset

The majority of RGB-D datasets are either collected through depth-sensors in real world environments, thus lacking the ability to capture the complete geometry of 3D objects, or rely on synthetic data generation processes, introducing a domain gap with the real-world that remains hard to bridge, as well as noisy labels being subjective to each annotator's interpretation. As such, in order to train the proposed methodology to predict occlusion-free views of objects in the scene, a new *hybrid dataset* is introduced in this section, taking advantage of the benefits of both approaches to remedy each others nuisances.

By lining up 3D models of objects, generated through CAD tools [462] or obtained through multi-view reconstruction [559], with the pose that these objects are met in individual frames of a video, synthetic occlusion-free ground-truth data can be generated and aligned to their corresponding real-world RGB-D inputs. Therefore, the resulting hybrid dataset minimises the annotation cost and resulting noise on the output, while preserving inputs to the real-world domain avoiding the introduction of a domain gap.

More specifically, in the context of this chapter the *UnScene* dataset is introduced, featuring an extended set of annotations for the YCB-Video [126] camera sequences. Originally designed for object pose estimation, YCB-Video consists of 92 long RGB-D videos, scanning table-top scenes with 3 to 9 objects each from a total of 21 categories, totalling more than 130K frames.

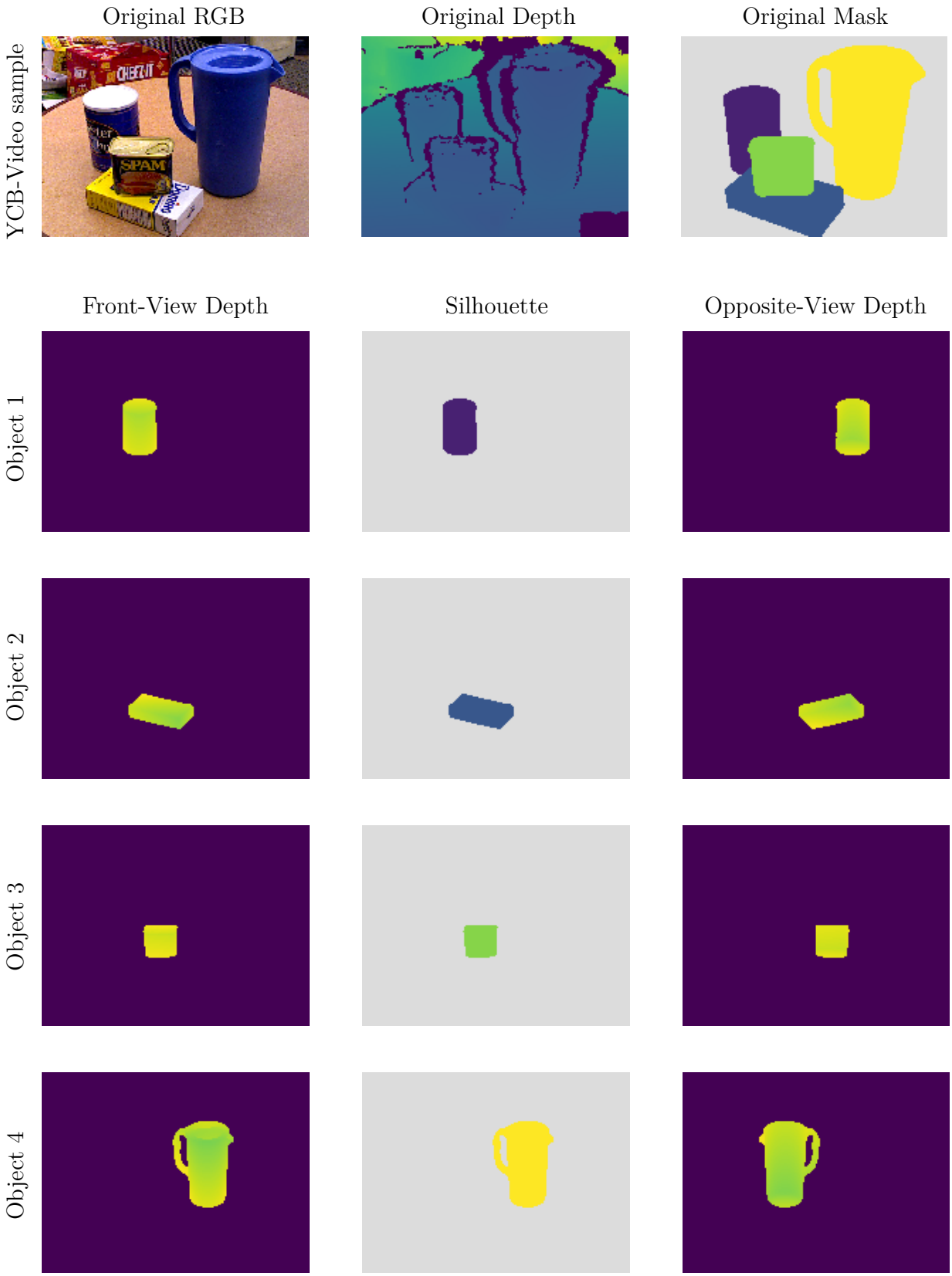


Figure 7.6: Occlusion-free annotations from the *UnScene* dataset; for a frame of a YCB-Video scene.

For each YCB-Video scene, the watertight meshes of the YCB dataset [559] are employed and placed in an empty virtual scene, according to their pose, as captured by PoseCNN [126]. Subsequently virtual cameras are placed at an identical pose to the actual camera in each YCB-Video frame, from which new depth and silhouette data can be obtained for each object independently. This effectively resolves inter-object occlusions, whereas corresponding opposite-view frames are rendered by a symmetric virtual camera along the same principle axis, positioned equidistantly to the middle of the scene[4], effectively resolving self-occlusions of each objects. It is noteworthy that, in contrast to training time, no information about the pose of each object is required at any stage upon deployment. Representative annotation samples obtained through this process are depicted on Fig. 7.6.

Importantly, by directly rendering ground truth via the orthographic projection model, a depth denoising effect is achieved (leading potentially to denser reconstruction), in contrast to the orthographic input that are sparsified by the the back- and re-projection process at runtime, as well as the common "bleeding" effect around the object boundaries. Furthermore, owing to the adoption of the orthographic projection model along with the design choice to preserve an identical rotation matrix between the real and virtual cameras, depth-normalised views from the object-centric virtual cameras can be rendered at training time by cropping the generated ground-data using the detection bounding box and adding/subtracting a bias term from their depth values according to the median depth of the observed object.

7.4.5 3D Reconstruction and Multi-view Fusion

The proposed methodology is able to offer dense shape predictions for all identified objects in a scene. To make these predictions available to other down-stream tasks, the model's output should be translated to commonplace 3D representations such as point-clouds or voxel-grids. Driven by the direct correlation of the predicted depth maps to (virtual) camera poses, this integration is trivial as predictions can be interpreted as additional depth measurements obtained from a known camera frames via a camera with a well-established projection model.

However, such single-view 3D reconstructions remains largely hallucination driven, and can contain inaccurate predictions about unseen areas of objects, particularly in the case of previously unseen instances of asymmetric items. Additionally, in some cases, non-convex shapes require depth

[4]The middle of the scene is calculated as the mean position of all the tracked objects within it.

measurements from more diverse viewing angles to capture their internal visible structure, that may feature complex self-occlusions under some views. Considering the above limitations, the proposed methodology is not designed to operate in a standalone setting, but rather as part of a multi-view reconstruction pipeline; while a novel fusion scheme is introduced below to facilitate the distinction between observed and predicted information, stabilising the long-term operation of the system.

Integration to a SLAM System

To avoid an inflation on the memory requirements caused by the adoption of mainstream 3D representations, the integration of the proposed methodology with SLAM systems is designed to preserve the object-centric approach of the rest of the proposed pipeline, that decomposes the scene into individual objects, abstracted out of the background and processed independently, while remaining spatially correlated to each other. As such, atomic 3D representations are adopted for each object, the locations of which are defined at a scene-level to provide a complete reconstruction. In this chapter, this is realised though the pose graph representation of Fusion++ [457], where each object is assigned to a dedicated 3D representation, while relative spatial placement of different objects are captured in a topological manner. This design choice further facilitates dynamic behaviour of rigid objects that can be individually displaced or even removed from the representation affected solely their topological connections, while also being markedly more scalable to pure volumetric representations in terms of memory requirements and captures substantially more information than pure topological ones. Employing such object-centric representations Fusion++ comprises a complete volumetric object-level SLAM system, offering multi-view fusion, camera tracking, loop-closure detection, re-localisation etc.

However, Fusion++ only incorporates information from real observations from a single moving camera. Instead in the proposed approach, several object-specific cameras are instantiated for each frame taking variable poses in scene, offering an abundant mix of real and hallucinated information. From a technical view-point, the proposed approach remains well-positioned to be integrated with such systems, as each virtual camera features a depth observation and known pose, and thus can be treated as an independent observation. In practise though, such a source-agnostic fusion of information can quickly introduce ambiguities on the reconstruction, and eventually corrupt the consistency of the whole SLAM system. To remedy the above challenges, a novel informed fusion approach, tailored the proposed occlusion-handling formulation is proposed below.

Upon deployment, information from multiple views is fused to the respective dedicated 3D representation of each object, following a semantic and topological registration of new observations to the existing reconstruction, while the representation remains scalable and new (previously unobserved) objects can be added at anytime. For each new camera measurement, the proposed CNN-based approach is employed as a mechanism to recover missing geometric and semantic information due to the incomplete observation of the scene. In the context of this chapter, the focus remains on the reconstruction part of SLAM systems. As such, it is assumed that camera poses can be accurately be obtained through the off-the-shelf localisation approach of the corresponding SLAM system (Fusion++ in this case).

More formally, a scene is scanned by a moving camera providing a sequence of RGB-D observations, denoted consistently with previous sections as $\mathbf{I}^{(t)}$ and $\mathbf{D}^{(t)}$, where $t = 0, 1, 2, ...$ signifies the frame count. Each object in the scene is assigned its own 3D representation $\mathcal{V}_i$, i.e. a voxel-grid parametrised by a fixed dimensionality of V^3 and a tunable voxel resolution parameter V_R. For simplicity, in this chapter it is assumed that all objects of a scene are at least partially observable on the first frame, therefore after the execution of the instance segmentation model on the first frame, N such voxel representations are instantiated. To scale up to scene-level representations, a world coordinate frame is arbitrarily defined identical to the camera pose at the first frame as $\mathcal{F}_W = \mathcal{F}_C{}^{(t=0)}$. All voxel grids are aligned to be perpendicular to the world frame, while their position in 3D space is defined to be spatially in the middle between the two corresponding virtual-camera poses as:

$$ {}^{W}\mathbf{R}_{\mathcal{V}_i} = \mathbf{I}^{3\times3}, \qquad {}^{W}\mathbf{t}_{\mathcal{V}_i} = \frac{{}^{W}\mathbf{t}_{v_i} + {}^{W}\mathbf{t}_{\hat{v_i}}}{2} \tag{7.11} $$

While Fusion++ offers advanced initialisation, re-sampling and re-localisation functionalities that accommodate more efficient and dynamic allocation of voxel size, it is considered outside the scope of this chapter, and the initial volumes are considered adequate to fully represent each object.

A camera localisation module (e.g based on the Iterative Closest Point (ICP) algorithm [560] between current depth observation and accumulated map), provides the 6-DoF pose of the camera for time-frame t, either directly w.r.t. the world frame as ${}^{W}\mathbf{T}_{C^{(t)}} \in SE(3)$, or w.r.t to the camera pose at the previous time-frame, which can recursively be projected back to the world frame as:

$$ {}^{W}\mathbf{T}_{C^{(t)}} = {}^{W}\mathbf{T}_{C^{(t-1)}} \cdot {}^{C^{(t-1)}}\mathbf{T}_{C^{(t)}} \tag{7.12} $$

Initially focusing on the *observed depth* information, each pixel observation is first back-projected to a disposable point-cloud representation w.r.t. the world frame, which can subsequently discretised to a voxel-grid through a projection operation denoted as $\mathrm{q}_{\mathcal{V}}(\cdot)$, and fused to the corresponding persistent object volume:

$$\mathcal{V}_i^{(t)} = \text{fuse}\left(\mathcal{V}_i^{(t-1)},\ \mathrm{q}_{\mathcal{V}_i}\left({}^{W}\mathbf{T}_{C^{(t)}} \cdot \pi^{-1}(\mathbf{D}^{(t)} \odot \mathbf{M}_i^{(t)})\right)\right) \tag{7.13}$$

The next section discusses the internal workings of this fusion process in more detail.

Notably, the *predicted depth views* provided by the proposed occlusion-handling system for each detected object, adopt a very friendly representation to the above formulation. This allows for their out-of-the-box integration to most SLAM back-ends[5], by considering them as additional observations obtained from known camera views. Hence, measured and predicted information can contribute to the 3D reconstruction process, through a technically homogeneous process.

Nonetheless, the aim of the proposed occlusion-handling model is not to replace the scene parsing process as a whole, but rather to work concurrently to provide a rapid approximation of the geometry covering the unseen parts of the scene, which would progressively be refined as measurements from more views become available. As such, a robust multi-view fusion pipeline remains essential, independently of the completeness of the proposed model's predictions.

Informed Multi-view Fusion

During the long-term operation of 3D reconstruction systems, plenty of views become available, each contributing a cumulative source of error due to camera noise, as well as drifting on the localisation etc. Hence, naive fusion (e.g. by averaging the current and previous volumetric representations) can degrade the quality of reconstruction and corrupt the system, due to subsequent localisation failures. This problem gets aggravated when hallucinated depth information is naively integrated into the 3D representation, in equal terms to the real measurements.

In fact, as the scanning of a scene progresses, more observations become available which may eventually capture previously unseen parts of objects, that have earlier been hallucinated by the proposed model to resolve occlusions. Accordingly, previously measured information may become unobservable from specific views, prompting the model to hallucinate structures that have already

[5]Back-end refers to the multi-view optimisation stage of SLAM systems tackling the state estimation problem of expressing the uncertainty of the localisation and mapping predictions; in contrast to front-end that is considering the visual perception tasks.

been adequately observed. Therefore, a need arises to distinguish between measured and predicted information when fusing both cues into a persistent representation, along with a formulation that would prioritise observed over predicted information and enforce consistency in long-term operation.

To facilitate such a probabilistic formulation of multi-view fusion, the Truncated Signed Distance Function (TSDF) -based approach introduced in [561] is considered to enhance the information captured on each object-centric volumetric representation $\mathcal{V}_i$. Signed Distance Fields (SDFs) constitute a representation of the distance between a set of examined points along a ray from their closest intersection to an object surface. Positive distance values indicate points outside the object surface, while negative values are used to represent unseen enclosed space within or behind objects. TSDFs, the truncated version of SDFs, simply saturate the distance values to a normalised fixed range [-1,1], where the zero value indicates the object-ray intersection. Its discretised version stores a continuous distance value for each voxel of a grid, instead of a binary occupancy indicator previously suggested, offering sub-voxel resolution for surface reconstruction. As a result of their more informative representation, (T)SDFs are broadly used for mapping and planning applications in robotics [562].

Observations from multiple views can be fused by a weighted average operation between the resulting TSDF volumes over multiple time instances, allowing for an implicit consideration of consistency and uncertainty that stabilises the long-term operation of the system. In traditional formulations, the fusion weights for each voxel are stored alongside the distance values, and are typically incremented as more observation become available, while they are also scaled proportionally to the cross product between camera rays and the observed depth-map's normal vectors [563], under the intuition that surfaces perpendicular to the camera are more accurately captured by a depth sensors compared to those being co-linear to its rays.

To formalise the above idea, for each new observation of an object i from the RGB-D camera $\mathbf{D}_i^{(t)} = \mathbf{D}^{(t)} \odot \mathbf{M}_i^{(t)}$, every pixel of the object-centric depth map $\mathbf{u} = (u_x, u_y) \in \mathbf{D}_i^{(t)}$ is back-projected to the 3D space as a point $\mathbf{p} = (p_x, p_y, p_z)$, and expressed with respect to the corresponding object's voxel grid coordinate frame as:

$$^{\mathcal{V}_i}\mathbf{p} = {}^{\mathcal{V}_i}\mathbf{T}_W \cdot {}^{W}\mathbf{T}_{C^{(t)}} \cdot \mathbf{K}^{-1} \cdot \mathbf{D}_i(\mathbf{u}) \cdot \mathbf{u} \tag{7.14}$$

Then, considering the centroid coordinates of each voxel in the object's allocated 3D representation $\overline{\mathbf{v}} = V_R \cdot (\mathrm{v}_x, \mathrm{v}_y, \mathrm{v}_z) \in \mathcal{V}_i$, a TSDF value can be produced for each voxel as:

$$\mathrm{TSDF}(\mathbf{v}, \mathbf{p}) = \begin{cases} 1, & \text{if } ||\overline{\mathbf{v}}|| - ||\mathbf{p}|| < -\gamma \\ -1, & \text{if } ||\overline{\mathbf{v}}|| - ||\mathbf{p}|| > \gamma \\ \frac{||\mathbf{p}|| - ||\overline{\mathbf{v}}||}{\gamma}, & \text{else} \end{cases} \tag{7.15}$$

where $2 \cdot \gamma$ is the effective distance range of the representation centred on the intersection of the ray with the point, outside which the TSDF value is truncated[6]. The object surface can therefore be represented by the zero-crossing of the stored SDF values, which can be interpolated to achieve sub-pixel accuracy. Usually, TSDFs are view-dependent, in the sense that only voxels that lie along the ray starting from the camera C and intersect the point p contribute to the calculation of the TSDF for each point, where the set of voxels that satisfy this condition are usually calculating through the process of ray-casting [564]. Additionally, if two points contribute to the calculation of the TSDF value of the voxel, the lowest distance is preserved, as by definition TSDF aims to capture the distance of every point in the volume with the nearest surface:

$$\mathbf{v} = \min(\mathrm{TSDF}(\mathbf{v}, \mathbf{p})), \quad \forall \mathbf{p} \in \mathbf{P}^{(t)} = {}^{\mathcal{V}_i}\mathbf{T}_{C^{(t)}} \cdot \pi^{-1}(\mathbf{D}_i^{(t)}) \tag{7.16}$$

Consistently with the KinectFusion algorithm [454], each voxel stores both the SDF value $\mathcal{S}_i^{(t)}(\mathbf{v})$ and its corresponding weight $\mathcal{W}_i^{(t)}(\mathbf{v})$, as these evolve over time. In the proposed approach, the adopted representation is enhanced, to distinguish between observed and predicted information. To accommodate this, without sacrificing the out-of-the-box portability of the proposed methodology to existing SLAM systems, separate vanilla Distance Fields are instantiated for the observed and predicted depth cues, as $\mathcal{S}_i$ and $\mathcal{S}_{v_i}$ respectively, which are associated with the same volumetric representation of each object $\mathcal{V}_i$, with their corresponding weights being truncated to [0,1].

All measured information from the depth camera is fused directly on $\mathcal{S}_i$ using the perspective projection model and the intrinsic parameters of the real camera sensor, obtained through traditional calibration [565]:

$$\begin{aligned} {}^{\mathcal{V}_i}\mathbf{P}_i^{(t)} &= {}^{\mathcal{V}_i}\mathbf{T}_C^{(t)} \cdot \pi^{-1}(\mathbf{D}_i^{(t)}) \\ \mathcal{S}_i^{(t)} &= \mathcal{S}_i^{(t-1)} + \mathcal{W}_i^{(t)} \odot \mathrm{TSDF}({}^{\mathcal{V}_i}\mathbf{P}_i^{(t)}, \mathcal{V}_i) \end{aligned} \tag{7.17}$$

Accordingly, predicted information from the front and opposite artificial views is fused on the corresponding $\mathcal{S}_i^{pred}$ of the same object, through the analytically obtained orthographic model,

[6] γ is often called cut-off value.

essentially treating them as observations of an additional sensor pair:

$$\begin{aligned} {}^{\mathcal{V}_i}\mathbf{P}_{v_i} &= {}^{\mathcal{V}_i}\mathbf{T}_C \cdot {}^{C}\mathbf{T}_{v_i} \cdot \pi_o^{-1}(\mathbf{D}_{\mathbf{v_i}}) \\ {}^{\mathcal{V}_i}\mathbf{P}_{\hat{v_i}} &= {}^{\mathcal{V}_i}\mathbf{T}_C \cdot {}^{C}\mathbf{T}_{v_i} \cdot {}^{v_i}\mathbf{T}_{\hat{v_i}} \cdot \pi_o^{-1}(\mathbf{D}_{\hat{\mathbf{v}}_\mathbf{i}}) \\ \mathcal{S}_{v_i}^{(t)} &= \mathcal{S}_{v_i}^{(t-1)} + \mathcal{W}_{v_i}^{(t)} \odot \text{TSDF}({}^{\mathcal{V}_i}\mathbf{P}_{v_i}, \mathcal{V}_i) + \mathcal{W}_{\hat{v_i}}^{(t)} \odot \text{TSDF}({}^{\mathcal{V}_i}\mathbf{P}_{\hat{v_i}}, \mathcal{V}_i) \end{aligned} \tag{7.18}$$

Subsequently, a unified representation of each object can be obtained by fusing the two volumes (observed and predicted), through a weighted fusion methodology, that ensures to prioritise observed information over predicted when ambiguities exists as:

$$\mathcal{V}_i^{(t)} = \mathcal{S}_i^{(t)} \odot \mathcal{W}_i^{(t)} + \beta \cdot (1 - \mathcal{W}_i^{(t)}) \odot \mathcal{S}_{v_i}^{(t)} \tag{7.19}$$

where the adopted weighting scheme in this fusion step reduces the contribution of the predicted depth information as the observed cue builds confidence on its representation, while $\beta < 1$ is a tunable parameter responsible to down-weight the overall confidence in representations heavily relying on the estimated depth cue. The resulting representation incorporates both real and estimated depth measurements, taking advantage of the dense predictions of the proposed model, while protecting the temporal consistency of the representation and fully compensating for the information loss introduced by the adoption of the orthographic model on the proposed representation.

Finally, although not studied in the context of this chapter, more sensitive back-end tasks of the SLAM system, such as camera tracking, loop-closure detection and bundle adjustment operations, can be performed relying solely on the more stable reconstruction of observed information $\mathcal{S}_i$, while any resulting corrections to estimated camera trajectory can be reflected back to the 3D representation of the predicted information.

7.5 Evaluation

This section presents the experimental evaluation of the proposed methodology. After the experimental setup is introduced in Sec. 7.5.1, the proposed occlusion-handling model is evaluated initially in isolation (Sec. 7.5.2) and then within a 3D reconstruction system employing the proposed fusion approach (Sec. 7.5.3), in a quantitative manner. Subsequently a qualitative analysis is performed to demonstrate the quality of 3D reconstruction results obtained, as well as showcase the limitations of the proposed method (Sec. 7.5.4). Finally, in Sec. 7.5.5 the effectiveness of the

proposed approach is demonstrated at the application-level, being employed within a robot-grasping pipeline. A 3D reconstruction approach relying on vanilla Depth Fusion [454] to an object-level semantic-aware representation [457] is used as a baseline; along with a state-of-the-art occlusion-handling approach from the literature. Namely, X-Section [482] is selected as a strong baseline, due to its high relevance and comparable computational demands with the proposed approach.

7.5.1 Experimental Setup

The proposed methodology is implemented in python3, using Tensorflow 2.1 (and the Tensorpack library [566]) to build and train the proposed model (and baselines). Offline 3D operations, such as the UnScene dataset rendering are implemented in Open3D [567], while online 3D reconstruction is mainly implemented via CUDA kernels that are integrated with the rest of the pipeline using the PyCUDA library [568]. The whole pipeline is executed on a workstation, equipped with an NVIDIA RTX2080 GPU.

An off-the-shelf Mask-RCNN [202] model trained on the MS-COCO [190] dataset is used for instance segmentation, based on a ResNet-50 backbone [3]. Experiments are conducted on the YCB-Video dataset [126], captured by an Asus Xtion Pro Live RGB-D camera, using the provided camera tracking annotations. The proposed model and examined baselines are trained using the UnScene dataset annotations across the YCB-Video scenes, a subset of which comprising 10 scenes are left out for validation and testing. The proposed model's backbone is pretrained on ImageNet [21]. Subsequently, training on the UnScene dataset takes place for 30 epochs, using SGD with momentum set to 0.9, a batch size of 32, an initial learning rate of 0.01 and a poly schedule [569] with power set to 0.9, and a weight decay term with coefficient set to 0.001.

Following an empirical search, the configurable parameters of the proposed reconstruction system are defined as follows: the loss function weight α is set to 2; the TSDF cut-off value γ is set to 0.5m; and the fusion weight for predicted depth data β is set to 0.7. The virtual orthographic cameras of the proposed representation, are parameterised by a uniform scaling factor s, set to 0.75. The padding term of each object centric representation is set in accordance with the detected object dimensions as $p = 2 \cdot max(h_i, w_i)$. Each object, when detected with a probability greater or equal than $p_{th} = 0.5$, is allocated a volumetric representation $\mathcal{V}_i$ with dimensionality V^3 where V is set to 128 voxels. The resolution of each voxel is qualitatively tuned to $V_R = 0.005$m, aiming for a reconstruction quality that would be able to take advantage of the detailed shape

information captured by the depth camera, while avoiding to reflect the white noise introduced by the sensor. This results in a memory consumption of approximately 32MB per object (to accommodate TSDF integration of both observed and predicted information, including fusion weights). Notably, this linear scaling comprises a remarkably efficient and most importantly *scalable* representation, compared to scene-level volumetric reconstruction counterparts (e.g. voxel-grids), typically demonstrating cubical memory scaling with extreme representation redundancy. However, this design choice can be adjusted accordingly, in-view of application-specific requirements on reconstruction quality, or constraints on the available memory.

7.5.2 Model Accuracy

This section focuses on the intermediate results produced by the proposed occlusion-handling CNN architecture. A quantitative evaluation with respect to ground-truth data is presented, along with an ablation study of different design choices and comparison with X-Section [482], the state-of-the-art work on image-space occlusion handling for 3D reconstruction. Both models are trained and evaluated on previously unseen YCB-Video scenes, employing the object-centric annotations of the newly created UnScene dataset.

For object silhouettes, the predicted mask of each object $\tilde{\mathbf{M}}_{v_i}$ is compared with the respective ground-truth $\mathbf{M}_{v_i}$ in terms of the commonly employed Intersection-over-Union (IoU) metric:

$$\text{IoU}(\tilde{\mathbf{M}}_{v_i}, \mathbf{M}_{v_i}) = \frac{\tilde{\mathbf{M}}_{v_i} \cap \mathbf{M}_{v_i}}{\tilde{\mathbf{M}}_{v_i} \cup \mathbf{M}_{v_i}} \tag{7.20}$$

Accordingly, front- and opposite-view depth predictions are compared with occlusion-free ground-truths in terms of Root Mean Squared Error (RMSE), considering solely the pixels within the ground-truth silhouette of each object[7]. Following a consistent notation to previous sections:

$$\text{RMSE}(\tilde{\mathbf{D}}_{v_i}, \mathbf{D}_{v_i},) = \sqrt{\frac{1}{||\mathbf{M}_{v_i}||_0} \sum_{x=1}^{h_{v_i}} \sum_{y=1}^{w_{v_i}} ||\mathbf{M}_{v_i}(x,y) \cdot \left(\mathbf{D}_{v_i}(x,y) - \tilde{\mathbf{D}}_{v_i}(x,y)\right)||^2} \tag{7.21}$$

The results are listed in Table 7.1. Overall, the proposed framework (row (iii)) achieves considerably good performance in terms of absolute error, reaching an RMSE of less than 1.4cm on the front-view depth and 2.9cm on the opposite-view predictions. This is significantly lower (up to 7.3×) to X-Section in row (i), showing an average error of 21.4cm on the opposite-view.

[7] This is done in order to disentangle the errors from the two predicted cues.

The main sources of error of X-Section, as identified in the experimental process are discussed below, along with the solution that the proposed framework provided.

First and foremost, X-Section's thickness-prediction based formulation does not allow for inter-object occlusion handling, introducing large prediction error in cluttered scenes where the object-of-interest may be surrounded by other items, causing occlusions from some camera views. Instead, the proposed methodology offers a unified formulation for self- and inter-object occlusion handling, allowing for robust shape recovery across the front and opposite view depth and silhouette cues.

Additionally, X-Section is heavily reliant on obtaining highly-accurate semantic segmentation masks for all objects-of-interest, as thickness is only predicted for the pixels within these masks. Hence, poor segmentation quality due to noise, complex object structures, poor generalisation, as well as the impact of occlusions on the segmentation task itself, severely degrade the framework's predictive capabilities. This is evident in (row (ii)) where ground-truth segmentation masks are employed, resulting to a significant improvement in X-Section's accuracy, which however remains $5\times$ lower to the proposed methodology. In contrast, the proposed model demonstrates remarkable robustness to noisy or inaccurate segmentation, as the use of ground-truth segmentation mask brings insignificant improvement to its accuracy (row (iv)). This is due to the preservation of context information on the model's input that allows the proposed CNN to extract further information on the object geometry, essentially refining the provided mask in the process of de-occluding the front-view depth and silhouette information and providing a more complete output.

As discussed, the occlusion-handling problem is inherently ill-posed. The proposed methodology took the approach of eliminating as many unnecessary redundancies as possible from the predictive task undertaken by the CNN, by employing 3D geometry-based pre- and post-processing steps. These steps essentially introduce inductive biases, aiming to reduce the variability of the input in the position and scale of the object, as well as the need to implicitly infer camera projection models. When removing these design choices (row (v)), i.e. by using a perspective projection model, the accuracy of the proposed model drops notably. This is due to the resulting lack of direct pixel-level correspondences between the input and output channels, as well as the increased dimensions towards which the model needs to develop an implicit invariance, over-complicating the learning process.

Finally, it is noteworthy that the adoption of RGB information on the model's input showed very limited impact on its predictive accuracy in the general case (row (vi)). Although one would expect that the appearance information of RGB can provide important clues e.g. on the occlusion

Table 7.1: Evaluation (and Ablation) of proposed Occlusion-handling Model.

	Method		RMSE↓			IoU↑
			Front	*Opposite*[1]	*Overall*	*Silhouette*
(i)	**X-Section** [482]	**with Mask R-CNN**	-	0.214	-	-
(ii)	X-Section [482]	with Mask labels	-	0.144	-	-
(iii)	**This work**	**with Mask R-CNN**	0.014	0.029	0.022	0.93
(iv)	This work	with Mask labels	0.011	0.025	0.018	0.95
(v)	This work	Perspective	0.032	0.120	0.071	0.85
(vi)	This work	+RGB Input	0.014	0.034	0.024	0.93

[1] Denotes cross-sectional thickness in the case of X-Section.

relationship between different objects in the scene, it is observed that the attention of the model during training drifts away from the strictly geometric focus of depth-and-silhouette representations, limiting its ability to exploit knowledge extracted from one type of object to another. Additionally, as RGB data demonstrate a significantly larger domain-adaptation challenge than depth maps and corresponding challenges in data augmentation and generalisability of the model, in the context of this chapter the use of RGB data is limited only to the initial segmentation model. Nevertheless, on a case-by-case examination, benefits were witnessed on some over-represented objects in the dataset, where appearance information may have helped their identification, and subsequent inpainting, even under heavy occlusion.

7.5.3 Time-to-Accuracy Evaluation in 3D Reconstruction

Nonetheless, the predictions of the proposed model do not always provide a complete 3D reconstruction (e.g. when encountering concave objects, such as mugs, from certain views). Instead, the proposed methodology is designed to operate as part of a multi-view 3D reconstruction system, where the occlusion-handling model's predictions provide a rapid approximation of the 3D shape of each observed object in the scene, which is progressively refined as more real observation become available. In this context, this section aims to evaluate the impact of the proposed methodology in the overall 3D reconstruction accuracy of such systems, and the underlying time-to-accuracy curve it provides at the application level, compared to baselines.

To measure 3D reconstruction accuracy in an intuitive way, the object-centric TSDF volumes hosting the measured and predicted information from multiple camera views are binarised, to form volumetric occupancy grid for each object. Negative SDF values, meant to represent internal object areas are casted to 1 (occupied), while positive SDF values to 0 (free space). Each object-centric

grid is then compared to the corresponding ground-truth 3D volume, obtained from the 3D models provided in the YCB dataset, which are placed in accordance to the ground-truth poses available for YCB-Video scenes. Accuracy is reported by means of Intersection-over-Union between the two volumetric representations, averaged across all objects present in the scene:

$$\text{3DIoU} = \frac{1}{N}\sum_{i=1}^{N}\frac{\tilde{\mathcal{V}}_i \cap \mathcal{V}_i}{\tilde{\mathcal{V}}_i \cup \mathcal{V}_i} \tag{7.22}$$

The temporal progression of the reconstruction accuracy is monitored by conducting the above comparison using the fused 3D information across several time instances, during the scan of each scene occurring within each YCB-Video sequence. The results of this experiment are reported in Figure 7.7.

Vanilla depth fusion approaches, such as the reconstruction pipeline of Fusion++ [457], demonstrate a slow growth in 3D reconstruction accuracy over time, as consecutive camera-views demonstrate large overlap and only a few previously-unseen pixels are newly observed on each frame. However, due to the fact that only observed information is fused in the map, accuracy monotonically increases over time, demonstrating a stable behaviour. In contrast, both occlusion-handling pipelines examined in this experiment (namely X-Section [482] and the approach introduced in this chapter) are able to provide superior reconstruction accuracy, even from the very first frame. This is due to the ability of such pipelines to hallucinate shape information from novel artificial views, that may not become observable at all during the scan, owing to complex occlusion, as well as physical constraints (e.g. surfaces being in contact to each other, or obstacles prohibiting scanning from certain viewing angles). Notably, most YCB-Video sequences only reach poses within an octant of a virtual sphere surrounding the scene, leaving many of the "opposite-views" predicted by the proposed model largely unobserved.

Furthermore, the proposed approach, owing to its superior predictive accuracy and more informative representation is able to surpass the reconstruction accuracy of X-Section by up to 17 p.p. on the first frame, and consistently maintain an analogous edge throughout. Additionally, the proposed methodology demonstrates a significantly more stable behaviour, with reconstruction accuracy increasing almost monotonically. This is due to its informed fusion approach, allowing for progressive refinement that seamlessly replaces predicted information with measured, as more views become available, correcting potential prediction errors. In contrast, X-Section (as well as the ablated baseline employing vanilla TSDF fusion with the proposed model), does



Figure 7.7: 3D Reconstruction accuracy (3D IoU) over the progression of time-frames during a multi-view scan of the scene, using different reconstruction pipelines.

not distinguish between observed and predicted information, demonstrating notable fluctuations in accuracy and resulting in a cluttered representation caused by persistent object artifacts, that lead to quality degradation in reconstruction during long term operation of the system (e.g. after the first 150 frames in the examined case).

Lastly, albeit the strong reconstruction performance of the proposed model, a large gap to ground-truth 3D models remains. This is partly due to the sub-centimetre resolution of the adopted representation, capturing highly-detailed shape information even in smaller objects, but being overly demanding for the achieved accuracy of the proposed model, as shown in the previous section. For real-world application where a less detailed 3D reconstruction is required [570], adopting a lower-resolution representation is likely to increase the accuracy gap between the proposed method and vanilla depth fusion baseline, in favour of this work. Nonetheless, albeit progress in the field, the occlusion-handling problem still poses several open challenges.

This chapter focuses primarily on application-level efficiency, essentially trading the achievable reconstruction accuracy w.r.t. the runtime of the overall system (i.e. scanning time of a scene). In this context, the proposed system dominates the underlying speed-accuracy trade-off across the application runtime. Through this lens, the actual inference latency constitutes a less important parameter to this trade-off, as additional factors (such as the lack of control in scanning speed and pattern) dominate the attainable results. For reference, a breakdown of the inference latency across

Table 7.2: Inference Latency and Throughput of each module in the proposed pipeline.

Model	Latency (s)	Avg. Throughput (fps)
Faster R-CNN	0.058 ± 0.004	17.2
Occlusion-handling	0.090 ± 0.050	11.1
TSDF Fusion	0.110 ± 0.040	9.1

different components of the proposed system is listed in Table 7.2. Although not explicitly optimised for performance, the proposed CNN model is able to meet near real-time inference requirements, when deployed on the target device. However, due to the large variability in the number and size of objects in the examined scenes of YCB-video, large latency fluctuations appear. In any case, the resulting operating throughput is not able to meet the frame-rate provided by the actual camera (typically 30 fps). However, due to the large information overlap between consecutive frames, experiments indicate significant robustness to the frame drop required for the system to operate in an online fashion. Nonetheless, there is room for several deployment optimisations, targeting lower layers of the deployment stack, to further push the efficiency of the each component.

7.5.4 Qualitative Analysis

In this section, some indicative qualitative results are presented and analysed, aiming to showcase the effectiveness of the proposed approach, as well as its limitations.

Initially, the analysis focuses on the object-centric intermediate results provided by the proposed model. Fig. 7.8, illustrates the results for all detected objects within a single-frame of a previously unseen YCB-Video scene. The observed (occluded) and predicted (de-occluded) depth and silhouette representation of each object is provided, along with the re-projected input to the model adopting an orthographic projection model, while combining the benefits of object-centric and view-dependent approaches. The predictions of the proposed CNN demonstrate its strong ability to resolve complex self (*all objects*) and inter-object (*obj:1,2*) occlusions, and provide noise-free depth data across all objects. Additionally, Fig. 7.9 indicates that the proposed model was able to complete missing depth and silhouette information that was not observed in the original frame (*obj:3,4*) due to the limited field-of-view and bad camera pose adopted, instead of a "traditional" occlusion. Furthermore, a strong denoising effect is evident on the front-view of observed objects, that compensates for noise introduced by the sensor, as well as the re-projection process of the object to the proposed virtual camera frame.

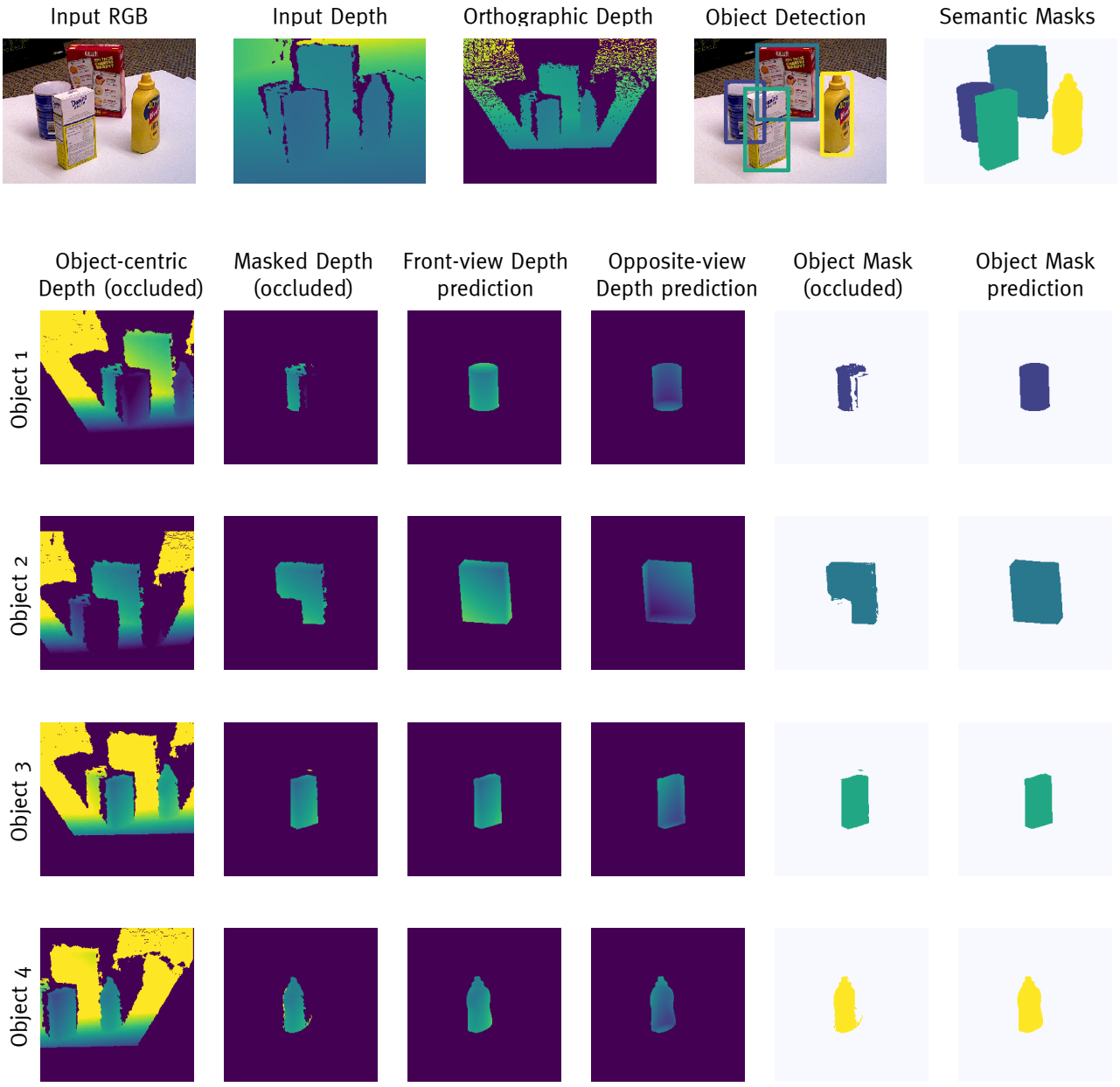


Figure 7.8: Qualitative analysis on a single-frame of a YCB-Video scene (shown on the top). The proposed model's depth and silhouette predictions, along with occluded input representations are illustrated for all objects in the scene.

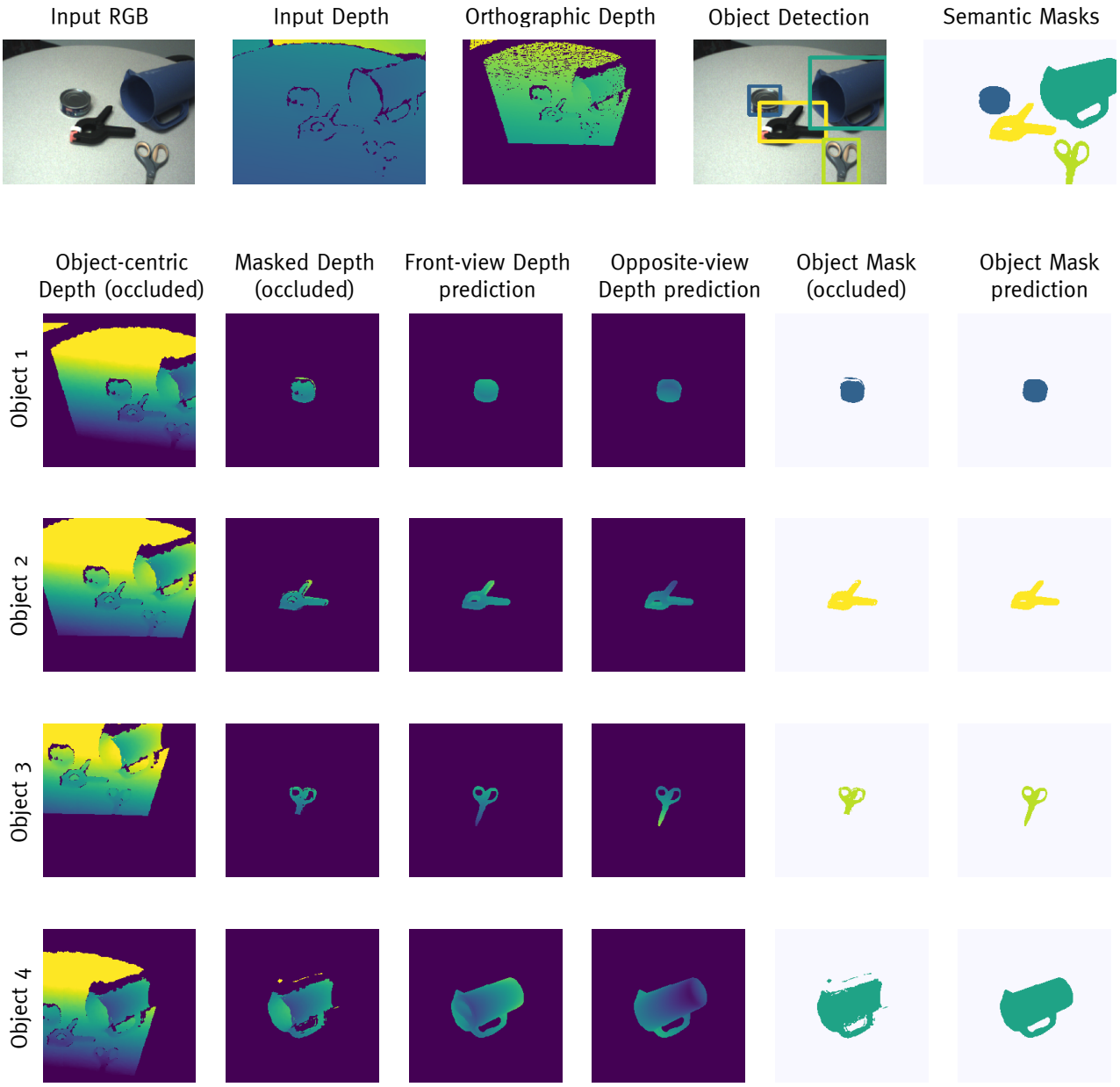


Figure 7.9: (contd.) Qualitative analysis on a single-frame of a YCB-Video scene (shown on the top). The proposed model's depth and silhouette predictions, along with occluded input representations are illustrated for all objects in the scene.

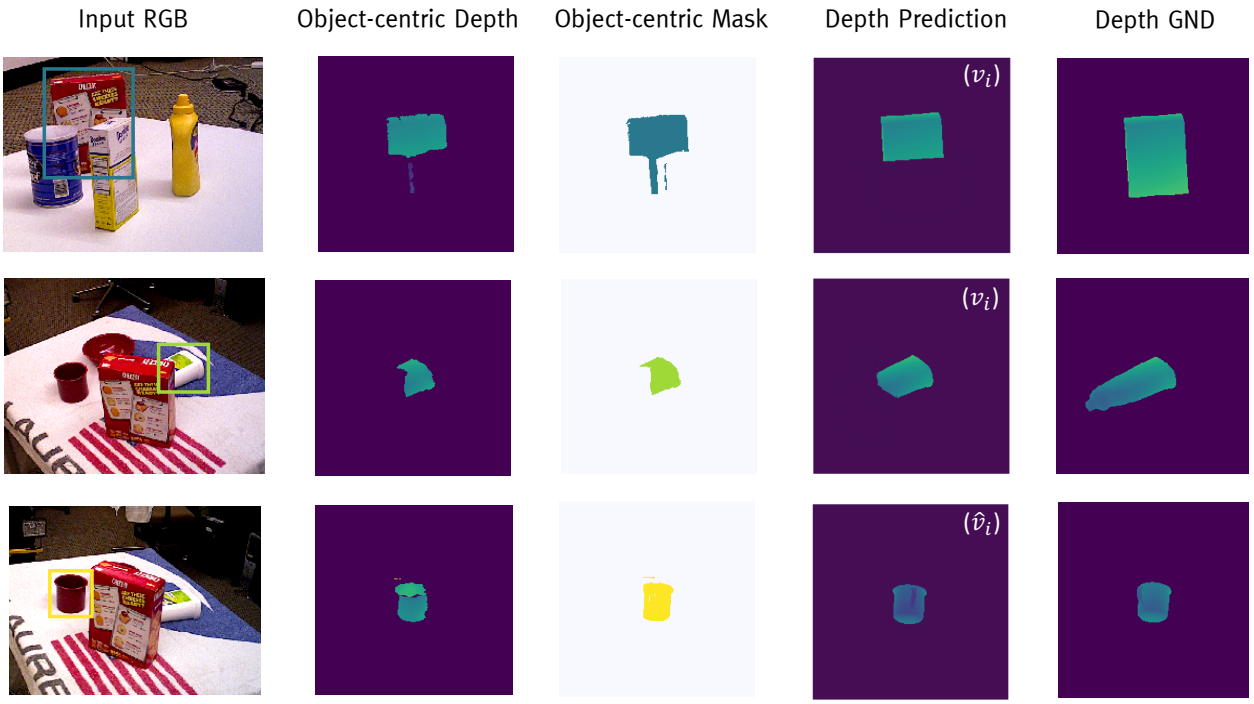


Figure 7.10: Illustration of indicative failure cases of the proposed model.

Subsequently, Fig. 7.10 illustrates some failure cases, indicating the limitations of the proposed approach. Focusing on inter-object occlusions, in the first case, the backward object is heavily occluded, while an incorrect depth reading is provided for its lower-part (caused by the known bleeding effect of depth sensor near object edges). This combination triggers the model to miss-interpret the scene structure, providing a partial depth re-construction that correspond only to the top (visible) part of the object. Accordingly, the second example showcases how heavy-occlusions of previously unseen objects can have equally catastrophic effects on their own. Finally, focusing on the self-occlusions, the third example demonstrates the ambiguity that is harbouring in complex objects. The model has been able to correctly reconstruct a depth structure that reassembles the handle of the mug, but lacking any further clues, the positioning of this prediction was wrong. Although this limitation can subsequently addressed by the proposed multi-view pipeline due to its novel fusion scheme, incorporating multi-view consistency in the training process of the model itself can potentially remedy such situations on-the-fly.

Transferring the same analysis to the 3D space, Fig. 7.11 renders the 3D reconstruction of a scene, encapsulating information obtained solely from a single depth image (green), as well as the prediction of the proposed approach (red). Evidently, the proposed formulation uniformly handles self- and inter-object occlusions with satisfying reconstruction accuracy. Furthermore, the sparsity

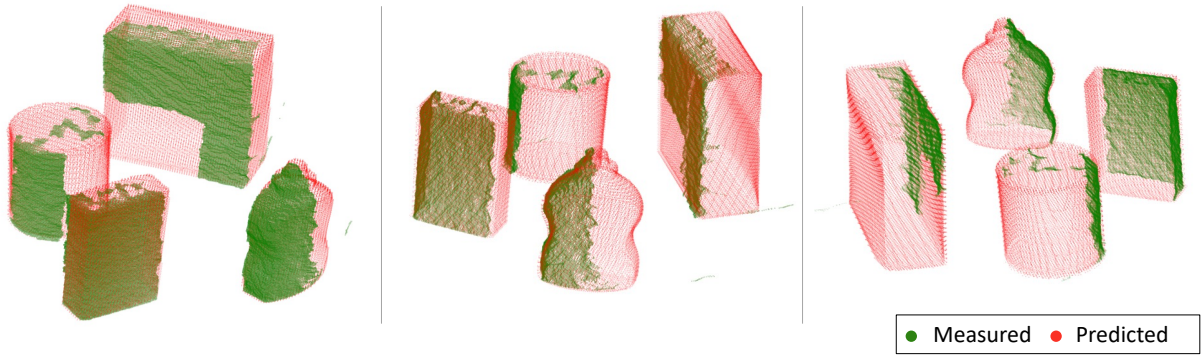


Figure 7.11: Single-view 3D reconstruction of a scene; indicating the difference between observed (measured) and predicted depth information from the proposed approach.

of the observed information contrasts the completeness of the predicted cue, demonstrating the effectiveness of the proposed vis-á-vis camera representation, that is able to encode dense 3D information with remarkable detail within a significantly compact format.

Notably, in the examined case, the selected camera pose is very friendly to the proposed representation (i.e. no object surfaces are being parallel to the camera axis). Nonetheless, the hallucinated information can get far from perfect, as illustrated earlier. Furthermore, the proposed representation remains in-between the 2.5D and 3D domains, suffering from its own limitations e.g. when dealing with objects with complex geometry, as well as when facing objects from particular (relative) views. As an example, Fig. 7.12 illustrates how the proposed approach is not able to capture dense information for surfaces that are parallel to the camera principle axis, limited in this case by its adopted representation. In such scenarios, single-frame reconstruction leads to a severe accuracy degradation on some objects (although still surpassing vanilla baselines). These limitations, however, are addressed by multi-view fusion pipelines, that progressively refines the 3D reconstruction, incorporating further information as better views of the scene become available. Representative instances of this progression are comparatively illustrated across different fusion approaches, in Figure 7.12, where observed shapes are represented by green, and predicted by orange.

First, when vanilla depth fusion is adopted, considering solely observed information [457], the resulting reconstruction remains sparse and leaves several occlusions unresolved, even after the fusion a large number of frames (Fig. 7.12a). Such cases require a thorough scan of the scene, which may be excessive time-consuming, disruptive to the application or even impossible on

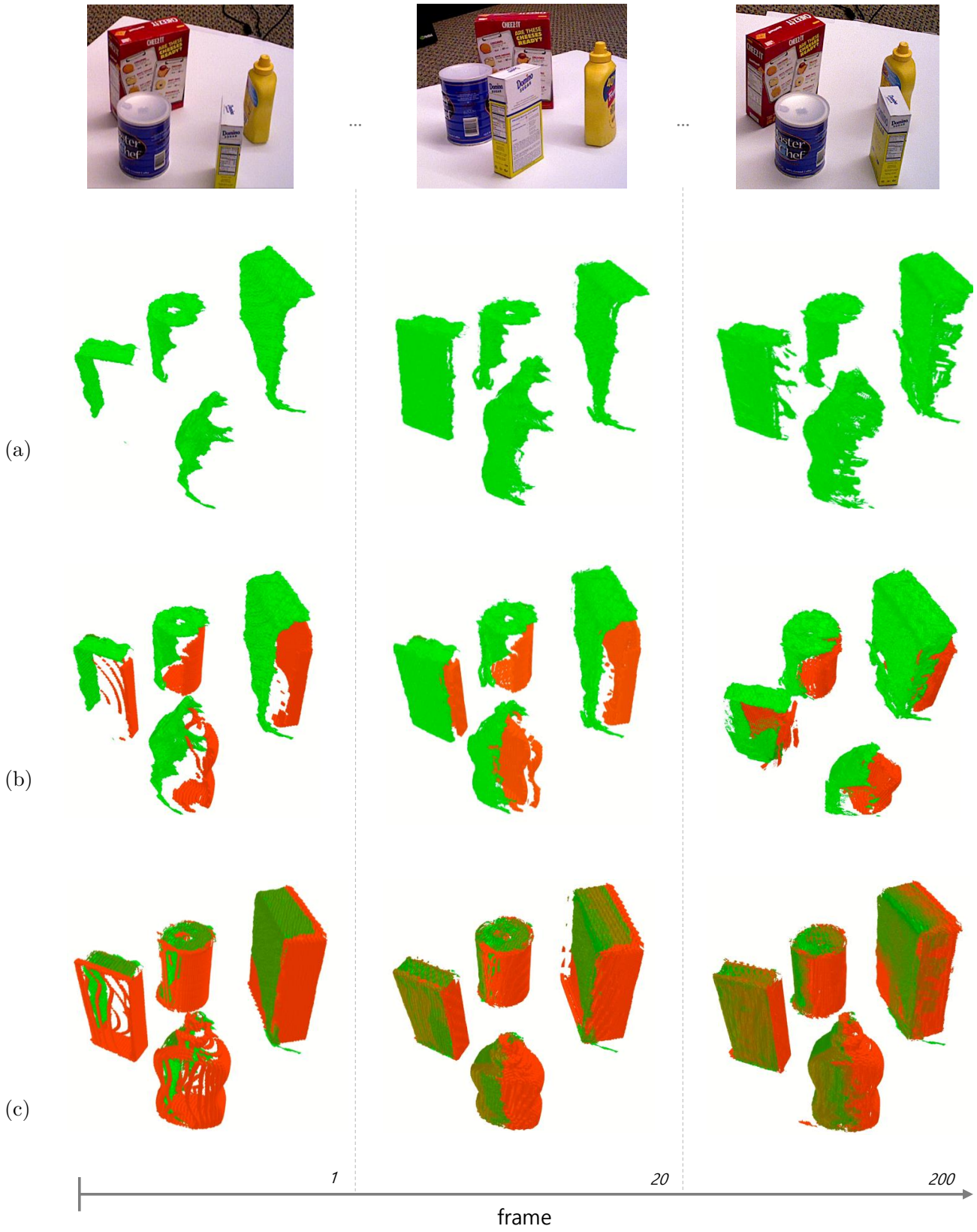


Figure 7.12: Multi-view 3D reconstruction of a scene, using: (a) Vanilla Depth Fusion [457], (b) X-Section [482] (c) the proposed approach; at different time instances of the scan.

several real-world deployment scenarios. X-Section's thickness prediction model demonstrates a significant enhancement in occlusion-handling. The predicted thickness information for visible parts of the object is progressively incorporated, leading to a denser reconstruction (Fig. 7.12b). However, this approach does not tackle inter-object occlusions, leaving the reconstruction largely incomplete; while during prolonged operation of the adopted fusion strategy persistent object artifacts are generated that eventually corrupt the shape representation. This is due to the equal treatment that X-Section's fusion scheme provides to both observed and predicted depth cues; leading to an accumulation of shape information on the map over time, rather than it's progressive refinement. Instead, the proposed approach is able to provide a considerably denser and coherent reconstruction at each step (Fig. 7.12c), uniformly handling self- and inter-object occlusions, while demonstrating more robustness in long-term operation, due to the informed fusion scheme that facilitates consistency.

7.5.5 Case study on Robot Grasping

Finally, this section showcases the effectiveness of the proposed 3D reconstruction approach, when employed within more complex real-world application scenarios, such as robot grasping.

Grasping comprises one of the fundamental task for robot manipulators [571], with grasp prediction being one of its main subtasks frequently undertaken by deep learning models, driven by the emergence of relevant large-scale datasets and effective models [572]. In its most common formulation, grasp prediction aims to generate a large number of 6-DoF grasp proposals [573] (i.e. poses of the robot's end effector that can result to robust object grasping). These predictions are subsequently fed to a motion planner [574], that evaluates the feasibility of each candidate grasp (i.e. the ability of the robot to kinematically attend to this pose), without any of its body parts colliding with its environment (including any other objects within the workspace), and selects the best grasp to be path-planned and get executed.

Intuitively, a detailed 3D representation is essential for such models in order to understand the geometry of the scene and the objects that would be manipulated by the robot. However, clutter in real-world environments and resulting occlusion, as well as partial observability (e.g. due to limited kinematic ability of the robot arm carrying the camera or other constraints in the scene), hinder the quality of traditional 3D reconstruction approaches. In this section, the proposed

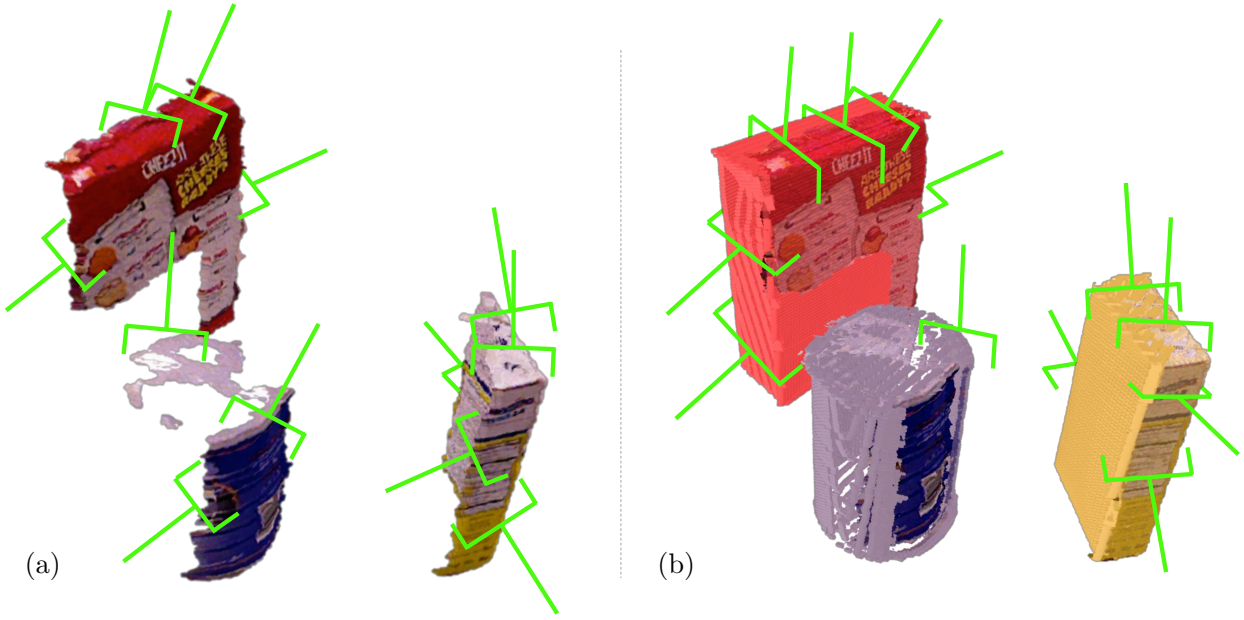


Figure 7.13: Grasp proposals generated by Contact-Graspnet using a 3D input representation from: a) A vanilla depth back-projection baseline; b) The proposed occlusion-handling approach.

occlusion-handling methodology is integrated with a state-of-the-art robot grasping pipeline, to further showcase its effectiveness directly in the application domain.

More specifically, the Contact-Graspnet [575] framework is adopted, following a learnable approach for predicting 6-DoF grasp poses, directly from input pointcloud data. Thereafter, a single-frame scenario is considered, where the pre-trained Contact-GraspNet CNN model is presented with a 3D representation generated by back-projecting solely a real depth observation, as well as by back-projecting both the observed and predicted views following the proposed approach. Fig. 7.13 illustrates a qualitative comparison between the two experiments[8].

The results indicate that incomplete 3D reconstruction is catastrophically hindering the effectiveness of grasp prediction, due the mistaken perception of the scene geometry inferred by the model. Instead, even coarse shape information generated by the proposed approach considering solely a single-view input is adequate to drastically improve grasp performance (qualitatively), shaping a better comprehension of 3D structures in the scene. Nonetheless, the lack of a watertight 3D representation still allows for false positive grasp proposals, that can be filtered out (as collisions between the end-effector and the object) by the motion-planner at a later stage of the process.

[8]Due to the extreme visual clutter introduced when illustrating all 200 grasp-poses provided the model, grasp proposals are selectively visualised, aiming to better showcase the examined aspects.

7.6 Discussion

This chapter focused on approximations applied directly on the highest, task layer, of the deep learning deployment stack. At this level, deep learning models can be employed to estimate the outcome of real-world tasks that are otherwise time-consuming or infeasible to perform (e.g. thorough camera scanning of a scene). In this context, focusing on the task of 3D reconstruction, a learnable approach was introduced to estimate the geometry of unseen parts of each detected object-of-interest. The proposed methodology was able to provide a rapid but complete approximation of each object's shape, at an exceedingly reduced time-scale compared to the time it would take for traditional mapping approaches to perform a complete scan of the scene.

The proposed approach comprised an object-centric pipeline, with the ability to handle both self- and inter-object occlusions uniformly, through a novel vis-á-vis virtual camera representation, that allows a CNN model to inpaint unseen areas and hallucinate different views of partially observable objects. As this task is naturally ill-posed, numerous design choices in the model architecture and training process, as well as the pre- and post-processing stages, were made aiming to constrain the predictive task, to aid convergence and generalisation.

Furthermore, the proposed occlusion-handling pipeline adopts a memory-efficient (image-space) depth and silhouette representation, that allows the reuse of segmentation CNN variants from the literature for the predictive task at hand (in place of computationally-explosive 3D CNNs). Additionally, the proposed representation maintained compatibility with SLAM systems for multi-view 3D reconstruction. As such, a tailored informed fusion approach was developed, with the ability to distinguish between observed and predicted shape information, to preserve consistency and facilitate the long-term operation of the system. This way, rapid approximations obtained from the proposed model during early-stages of the scanning process, can be replaced by real depth measurements as information from more camera views become available, progressively refining the 3D map.

The proposed approach demonstrated enhanced prediction accuracy compared to state-of-the-art CNN-based approaches from the literature, while dominating the Time-to-Accuracy curve at an application level. However, a considerable gap to ground-truth labels remains, calling for further research in this direction. Nonetheless, when incorporated within a robot grasping pipeline, the proposed approach was able to provide high-quality actionable 3D reconstruction, leading to more informed and effective grasp proposals, even under severely limited observability of the scene.

Alongside, a new dataset termed UnScene, comprising an enhanced set of annotations for YCB-Video sequences, offering object-centric occlusion-free depth and silhouette renderings (adopting the proposed representation) is created. Although synthetically generated, all annotations are view-dependent and match directly real-world inputs for YCB-Video, forming a hybrid (real-input, synthetic-output) that remains unaffected by the domain gap in related efforts.

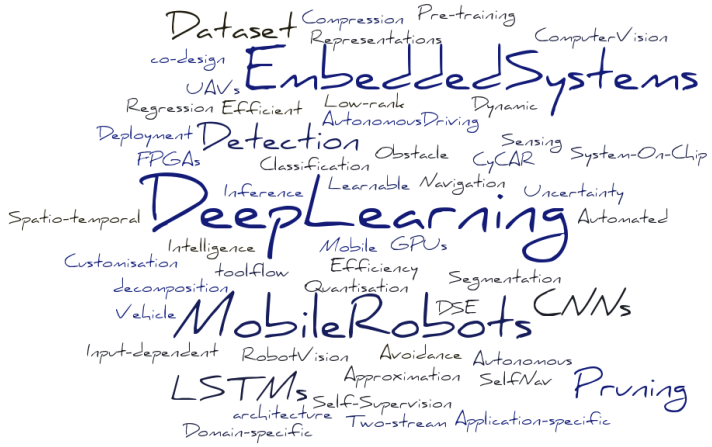


Image generated using: https://worditout.com/.

8

Conclusions and Future Work

Contents

8.1 Does *general purpose* mean "*good for nothing*"? 248
8.1.1 Computation-level Customisation 248
8.1.2 Model-level Customisation 250
8.1.3 Data-level Customisation 251
8.1.4 Task-level Customisation 252
8.2 Discussion 254
8.2.1 Dynamic Inference 254
8.2.2 Optimisation Objectives 255
8.2.3 Customisation Interplay 256
8.2.4 Design Automation 257
8.2.5 Target Tasks/Models 258
8.3 Closing Remarks 259

8.1 Does *general purpose* mean "*good for nothing*"?

This thesis holistically explored this question, visiting every layer of the deep learning deployment stack (Ch.1). The main take-away message arising from the findings of this thesis is that:

Efficiency comes from customisation.

So, although general-purpose systems are particularly effective when *development* aspects such flexibility, portability and re-usability are of major importance, in terms of *efficiency at deployment time* special-purpose solutions are the key. More specifically, this thesis showed that customisation can successfully be applied across all layers of the deep learning deployment task, yielding significant gains. The main employed vehicle for customisation has been *approximation*, that was exploited in view of *application-specific* requirements and constraints, or opportunities for optimisation. Throughout the chapters of this thesis, the target applications revolved around *robot vision*, which poses unconventional deployment challenges, but also unique customisation potential.

The remainder of this chapter summarises the contributions and key findings of this thesis, and jointly discusses open challenges and potential directions for future research.

8.1.1 Computation-level Customisation

The chapters of this thesis followed a bottom-up path along the deep learning deployment stack, starting from the lowest *computation layer*. In this context, Ch.3 employed cascaded low-precision arithmetic units to boost inference efficiency on CNN-based classification. Accordingly, Ch.4 applied an iterative refinement scheme on inference computations, through low-rank approximation, to optimise the time-to-accuracy trade-off on LSTM-based regression.

By focusing on computation-level approximations, both approaches were able to yield performance gains without the need of re-training the original model or requiring access to the training data. In contrast to methods applied closer to the application level, this is a distinctive customisation aptitude of the computation layer, facilitating *data privacy*, which becomes increasingly important across the consumer, commercial and healthcare sectors, both due to the asset-value and sensitivity of data, as well as imposed regulations [576].

Aiming to push the limits of performance, both chapters employed a *dynamic inference* scheme, in order to introduce and exploit a speed-accuracy trade-off. More specifically, Ch.3 focused

on the optimisation potential arising from the commonplace static workload allocation across inputs. By organising a low- and high-precision unit in a cascade, with a confidence evaluation unit in between, each sample could employ varying precision during inference according to its perceived difficulty. Although harder samples had to pay a notable computational overhead by going through a complete low-precision inference before being re-directed to the higher-precision variant, the overall system was still able to alleviate this wasted computation cost and benefit from the remarkable computational savings on easier samples. Inspired by works on early-exit models [127][128], future work may explore ways for re-using the computations of the low-precision model variant during later cascade stages, if invoked [577]. This could be achieved by employing specialised hardware architectures for online *bit-serial* arithmetic [578], minimising the computational overhead on difficult samples, while also enabling for a finer-granularity of precisions to be employed. Furthermore, it can be concluded that the "exit-policy" of dynamic inference systems should ideally demonstrate distinct properties from generic uncertainty estimation methods, in order to utilise the full efficiency potential of this computation scheme; with the development of such metrics remaining an open-challenge for future investigation.

Accordingly, the iterative approximation scheme of Ch.4 allowed for *"anytime" inference* through progressive refinement, where computation can be interrupted at any time (either *statically* in order to meet application-specific latency constraints [108], or *dynamically* to accommodate variable workload in resource-sharing environments [270]), while still providing the most accurate approximation of the result possible for the latency budget at hand. Future work can incorporate a mechanism for uncertainty estimation [579] on the examined regression task, to allow for difficulty-aware performance scaling as well, by adjusting the number of iterations spent on each input sample. For the case of recurrent models, this would also require training-time modifications for the model to be able to encapsulate temporal information in its internal state, under a variable refinement granularity across the samples of each sequence.

In order to realise latency speed-ups from the computation-level approximations introduced, both approaches heavily relied on *custom hardware* accelerators. This is because the lack of standardisation of custom low-precision data-types (Ch.3) and irregular data-path of the proposed iterative refinement scheme (Ch.4), lead to significant underutilisation of the computational resources of general-purpose platforms. In contrast, custom-hardware solutions tailor the hardware to the structure of the underlying workload, optimising the utilisation of the available resources

and leading to the reported efficiency gains. Particularly for quantisation, future work may focus on exploiting the increasing support for (a limited set of) custom precisions on deep-learning centric many-core platforms [25], that may soon loosen this constraint allowing for proportional performance gains by the proposed multi-precision CNN cascade on other platforms.

Additionally, both chapters exploited analytical *performance modelling* and *design space exploration* to provide highly-accurate performance estimation for different model and hardware architectural variants. The effectiveness of this approach constitutes an important enabler for model-hardware co-design methodologies, without the prohibitive overheads of traditional hardware design methodologies. The respective chapters showed that by following this design approach and exploiting the flexibility of FPGAs, the device utilisation and thus inference efficiency can be maximised, by providing highly-customised solutions in view of the model and platform at hand as well application-specific requirements. In this context, optimising (shared) hardware engines for multiple DNN models [580] using analytical modelling remains a promising direction for future work.

Finally, the work of Ch.3 further showed how exploiting the runtime re-configurability of FPGAs, by exposing architectural adaptation as a design dimension, can provide considerable gains in high-throughout scenarios where large batching is allowed. Accordingly, for latency-driven applications, resource sharing led to comparable gains at the expense of more complex (and less deterministic) modelling. Future work may try to come up with a unified solution across both scenarios, employing custom-hardware accelerators with multi-precision support, reducing the reconfiguration overhead through runtime hardware adaptability [581].

8.1.2 Model-level Customisation

Vistiting a higher layer of the deployment stack, this thesis showed that approximations on the *model layer* either rely on the inherent redundancy of trained DNNs [582] aiming to eliminate unnecessary computation through sparsification (Ch.4), or exploit a controlled compromise in generalisation capability (when allowed by the specificity of the target application) to provide customised efficient model variants [114] (Ch.5). In that sense, Ch.4 explored structured parameter pruning as an orthogonal direction combined with the proposed progressive LSTM inference scheme; while Ch.5 pushed the speed-accuracy frontier on aerial object detection by incorporating application-specific (e.g. altitude-aware) criteria to dynamically prune workload from a pre-trained model.

Such approximations can equivalently be applied post-training in a privacy-preserving manner. Additionally, in that layer, the attainment of corresponding speed-ups on the target deployment platform strongly depends on the *structure* of the applied modifications on the DNN architecture. Ch.4 imposed a heavily restricted structure on the employed sparsity pattern to preserve the regularity and uniformity of the underlying computation. Although unstructured pruning can usually offer notably increasing compression efficiency and theoretical workload savings, its implementation introduces significant overheads that may counteract its benefits due to scheduling challenges, stochastic memory-access patterns, additional sparsity index information storage and processing requirements and transient device underutilisation due to the variability of the workload [583].

Accordingly, Ch.5 applied a highly-structured computation-skipping approximation, in the form of informed region pruning, where a dynamically determined set of object proposals completely omitted a large part of the model. The employed selection criteria are application-specific and can be evaluated at runtime with negligible computational overhead, by being highly-parallelisable by design. As a result of this coarse-grained path selection, performance gains can be obtained without requiring custom-hardware support. Even so, the speed-accuracy trade-off was still not linear owing to other affected efficiency factors, such as reduced region batching and weight re-use opportunities, as a result of the proposed computation skipping mechanism.

Ch.5 also demonstrated the enhanced potential of exploiting the *multi-modal* sensor data that are widely available on mobile robot usecases, along with prior-domain knowledge, not just for increasing accuracy and robustness as usually studied [584], but also towards improving efficiency. Future work can focus on exposing more approximation dimensions (such as input resolution [105]) to the proposed altitude-aware dynamic inference scheme; as well as integrate similar application-specific information on more complex mobile robot and UAV based vision tasks [585], and potentially incorporate them on the training process of the model, to enhance their performance and efficiency through informed dynamic inference [586].

8.1.3 Data-level Customisation

Subsequently, Ch.6 explored approximations on the *data layer*, by means of employing implicit end-to-end learnable representations extracted from data, to replace explicitly calculated feature cues on the model's *input*.

More specifically, focusing on the task of vision-based robot/UAV navigation, a two-stream CNN architecture processing the current and previous frame was able to replace the need for explicitly calculating optical flow, leading to significant performance gains while remaining able to enjoy the accuracy benefits of exploiting *spatio-temporal information* [587]. This was enabled by a novel self-supervised pre-training mechanism, guiding the representation learning process towards identifying correlations between frames, without handcrafting complex loss terms or explicitly annotating data for the proxy task and/or defining the format of the intermediate representation. Employing this representation, a deep learning model was effectively used to approximate distance sensor readings relying solely on visual data, offering a tunable and transferable autonomous navigation solution that is applicable on every commercially-available camera-equipped drone.

Additionally, self-supervision has proven to be effective in reducing the human-involvement in the otherwise tedious data collection and annotation process. With the advancement of deep learning being heavily driven by the emergence of large-scale datasets, while robotics being challenged by inflated costs and safety concerns related to data collection in real-world environments, this paradigm can pave the way for the creation of large-scale *visual navigation datasets*, with a wide diversity between environments.

In this direction, future work may focus on employing the proposed self-supervised training approach in an *online learning* setting [588]. This would allow the robot/UAV to "learn from its mistakes", obtaining annotated data through IMU/odometry and adapting the model to generalise better on previously unseen domains [589], or even personalise on the target environment [590], to potentially achieve further computation savings through specialisation [591].

8.1.4 Task-level Customisation

Finally, Ch.7 visits the highest *task layer* of the deployment stack, where application-specific deep learning models are used to approximate the *output* of the target task as a whole. Focusing on semantic 3D reconstruction, the proposed methodology is able to "render" occlusion-free object-centric views of the target scene, approximating geometric information that was missing from the original observations due to occlusions, noise and other sensing limitations. As a result, task-level time-to-accuracy is notably improved on the underlying 3D reconstruction problem.

To achieve this, a novel memory-efficient image-space representation was introduced to initially migrate object-centric observations and subsequently predict their missing parts within this

more constrained space through the proposed CNN. The proposed representation allowed for robust *temporal* and *spatial embodyment* [70], in the sense of multi-frame and camera-view aware fusion. This enabled the integration and progressive refinement of information on a persistent representation, that can facilitate higher-level downstream tasks such as robot grasping. Additionally, the proposed fusion approach, distinguishing between measured and predicted information, played a key role in preserving the consistency of the resulting representation.

The next step of this embodyment envisions incorporating *active vision* to the pipeline [592], enabling the robot agent to select the next best-view [593] and control the camera pose in an informed way for scanning the scene [593]. This would dynamically resolve ambiguities while minimise the number of views required to obtain an accurate 3D reconstruction of the scene [594], further boosting task efficiency. Additionally, scaling the proposed approach beyond table-top scenes to whole room or even building scale [101], where a larger number and wider distribution of objects may appear, remains a promising direction for future work. Apart from the considerable task-efficiency gains this could bring on that level, it would also unlock further potential of the proposed methodology to mobile robot applications such as map-based autonomous navigation [595], and target-driven navigation [411].

Furthermore, the proposed methodology demonstrated that when diverting from monocular RGB-based cues, a careful blend of *real data* with *synthetic annotations* can effectively bridge the domain gap. This approach could enable the creation of large-scale hybrid datasets at low cost, targeting 3D reconstruction [462] as well as more complex downstream tasks, such as robot grasping [572].

Finally, in the context of this thesis, Ch.7 approached 3D reconstruction from a task-efficiency view-point, aiming to maximise the underlying time-to-accuracy trade-off (i.e. providing an early approximation of unseen geometry information even from a single camera view, and progressively refining the resulting 3D representation as more views become available). This is just the tip of the iceberg in an rapidly evolving research field, with plenty of future-work opportunities. These include enforcing multi-view consistency [596], scene-level context [597] and physical plausibility (object hull intersection) constraints [598] to the proposed model *at training time*, to further stabilise the proposed representation and incorporating special handling for dynamic and deformable objects in the scene [599].

8.2 Discussion

8.2.1 Dynamic Inference

As deep learning models are becoming progressively *less redundant* through the use of efficient design methodologies, while their execution environments are characterised by *increased dynamicity* due to resource-sharing across multiple models/tasks as well as the impact of thermal and energy management, *dynamic inference* is becoming an important enabler of deployability and efficiency for the future. Under this paradigm, AI models are not statically deployed, but maintain a level of flexibility in their computation graph, offering an adaptable speed-accuracy trade-off during runtime.

As a result the computation and memory footprint of the inference process can be dynamically adjusted upon deployment, to reflect on the availability of the platform's computational resources [270], the needs of the target application and the difficulty of the input sample at hand [269]. Various flavors of this setting, referred to as *input-dependent inference*, have been explored in the context of this thesis, including the multi-precision CNN cascade of Ch.3, the iterative LSTM approximation of Ch.4 and the dynamic region selection of Ch.5. Additionally, recent literature has explored several orthogonal dimensions of dynamic inference, including early-exit models [174], dynamic layer skipping [600], resolution adaptation [105] and dynamic routing [601].

A key component on all the above work is the need for a sample "routing" mechanism, that determines the computation path of each input at runtime, with minimum overhead. In the input-dependent inference scheme, this mechanism is typically associated with the notion of *uncertainty*, as a means of capturing the predictive *difficulty* of each sample. In its current form, uncertainty estimation faces several challenges, such as (i) the excessive computational overhead of ensemble approaches [602][603][604] and (ii) the sensitivity of proxy metrics such as softmax and entropy on the over-confidence of deep learning models [605].

Consequently, the domain of dynamic (input-dependent) inference can significantly benefit from new methodologies for estimating the predictive difficulty of samples, related or not to uncertainty, that would exhibit: (i) well-calibrated behaviour; (ii) minimum computational overhead; as well as (iii) deployment-time tunability and (iv) interpretability. Promising solutions towards this goal shall aim to minimise the workload spent on each sample while preserving predictive

accuracy. Future work on this direction may find inspiration from relevant research fields such as out-of-distribution (OOD) detection [606][607] and evidential deep learning [608][609].

Nonetheless, higher-level robot tasks such as the vehicle detection (Ch.5), autonomous navigation (Ch.6) and multi-view 3D reconstruction (Ch.7), would also greatly benefit from efficient and effective techniques for quantifying DNN prediction uncertainty. Notably, Bayesian techniques for temporal, spatial and multi-modal information fusion have been well-established in "traditional" robotics [610], and incorporating deep learning in such settings (i.e. treating its predictions as additional sensor readings) is still an open question, requiring a well-calibrated estimation of uncertainty, especially when operating in real-world "open-set" conditions [611][612].

8.2.2 Optimisation Objectives

An important enabler for the approximation approaches explored in this thesis has been the projection of their impact upwards, to the application-level. When studied in isolation, the impact of approximations is measured by task-agnostic proxy metrics, such as RMSE between approximate and faithful outputs (for accuracy) and single-frame workload/latency (for speed).

However, approximate inference relies on exploiting the underlying speed-accuracy trade-offs by departing from faithful model implementations. As a result, it is important to translate and study the impact of each approximation directly on the target application-domain (where certain effects on lower-level module performance or accuracy may be hidden or inflated), in order to understand the actual implications of different design choices, and find the right balance for the examined use-case.

Not less important, proxy metrics for performance, such as FLOPS and number of parameters, have proven to be severely ineffective estimators of the actual inference latency, as they fail to capture the mapping inefficiencies of the underlying workload to the target platform. Again, projecting these metrics directly on the application domain, through hardware-in-the-loop simulation or benchmarking [613] is required to bridge this gap and allow for realistic evaluation of the impact of different approximation methods and design choices.

Finally, task-agnostic metrics typically study model-level speed and accuracy in isolation. However, in continuous vision applications, such a visual perception on mobile robots, speed and accuracy can be highly correlated. For example, in the UAV use-cases of Ch.5 and Ch.6 a highly-accurate but computationally-expensive model may not be able to able to cope with the incoming frame rate, causing the system to "drop" frames. This could introduce false negatives (e.g. in

terms of vehicle detections or obstacle recognitions) harming the task-level accuracy of the overall system. Employing application-specific metrics instead, allows capturing all sources of system error/failures in a unified manner. Thus, when directly used as primary optimisation objectives, such metrics expose both the unique challenges and opportunities of different applications. This enables the employment of the most suitable granularity of approximation techniques, pushing the limits of efficiency by fully exploiting the underlying speed-accuracy trade-offs.

8.2.3 Customisation Interplay

This thesis explored several customisation/approximation methodologies across the deep learning deployment stack, with each chapter focusing on a different scenario and abstraction level. Ch.3 and Ch.4 showed that approximations focusing on the lower layers of the stack frequently affect the regularity and structure of the underlying workload, introducing inefficiencies on the mapping to general-purpose compute platforms, resulting in reduced device utilisation and inference efficiency. To remedy this challenge, low-level approximations can largely benefit from accompanying custom-hardware solutions, offering enhanced flexibility on the mapping process and thus allowing for better utilisation of the available resources, leading to proportional performance gains. In contrast, Ch.5, Ch.6 and Ch.7 showed that customisation on higher layers of the deployment stack can easier yield out-of-the-box latency speed-ups, by avoiding to tackle with low-level parts of the pipeline. However, such customisations have a strong reliance on accessing specialised training data at the target domain (that are not always available), and often sacrifice generalisability of the adopted solution in favour of performance. Nonetheless, such methodologies leave unexplored customisation potential at the lower layers of the stack, that could lead to *complimentary* performance gains.

Evidently, the design of targeted approximation methodologies for certain layers of the deployment stack does not happen in a silo. Instead, approximation methodologies typically rely on control signals coming from higher layers of the stack itself. Quantisation for example, studied in Ch.3, is conducted on the computation layer by making strong assumptions about the input data distribution relying on the higher data layer. Accordingly, application-specific approximations, such as the model-layer proposal pruning of Ch.5, rely on information about the target use-case present in the task layer.

However, the interplay between different approximation schemes simultaneously applied across these layers remains largely unexplored in the current literature. Insights provided from recent

works indicate that the most efficient way to scale up/down DNN architectures is by proportionally increasing/decreasing the depth and width dimensions [52], leading to the EfficientNet model family. Additionally, aiming to recover the accuracy loss caused by quantisation, Wide Reduced Precision Networks [614] successfully increased the learning capacity by adding extra filter channels.

Beside from such empirical result, a formal study of the *learning capacity* landscape and how different approximation methodology affect its "dimensionality" is of greater interest. The aim of such a study would be to provide necessary insights about the interplay of approximations methods, to answer question such as: Is a quantised model's accuracy more susceptible to pruning or early-exiting than its full-precision counterpart? And if such a trade-off exists, what is the most appropriate methodology to find the right balance between different approximation strategies? From a systems perspective, an insight arising from this thesis is that this process should be aware of the target hardware platform and the impact of each approximation strategy on both accuracy and inference speed. However, a more principled and formal exploration is required in order to draw robust conclusions answering these questions.

8.2.4 Design Automation

The approximation taxonomy introduced in this thesis showed that customisation approaches are becoming increasingly intrusive as they traverse the lower layers of the deployment stack. This progressively imposes joint requirements for deep learning, systems and hardware design expertise. The fast pace of progress in AI continuously introduces new diverse model architectures. The adoption of these models on emerging applications such as robotics is struggling to keep up with such latest advancements, partly due to the requirement for cross-domain expertise. Additionally, the excessive computational burden of new accuracy-optimised models introduces the requirements for an intermediate customisation/approximation step in order to optimise the models for deployment in the embedded landscape, and enable their applicability on the target task. This fact further hinders the fast adoption of latest AI developments in downstream real-world applications.

Thus, new methodologies enabling the (partial or complete) *automation* of the customisation task are required, to bridge this gap. Similarly to automated model design approaches, towards this direction automated DNN-to-HW toolflows, such as those introduced in Ch.3 and Ch.4, can facilitate the rapid adoption of highly efficient model variants mapped on tailored hardware accelerators,

while meeting user-specified application-specific requirements/constraints and hiding the need for hardware design expertise and tedious development process of traditional design approaches.

Accordingly, approximation approaches targeting the higher layers of the deployment stack rely on *custom datasets* with rich domain-specific annotations. To mitigate this overhead, transfer-learning approaches (as the one used in Ch.5), automation at the data collection and annotation process (e.g. through self-supervised learning, as in Ch.6) or even the use of simulators and semi-synthetic data (as in Ch.7) can be used. However, in the general case, robust domain adaptation between synthetic and real environments is still an open challenge, actively explored though data augmentation [615], domain randomisation [550] and meta-learning [616] -based methodologies.

Furthermore, *re-usability* of proposed solutions is important, to accelerate the development of new technologies, as well as their pace of adoption in production. In that sense, it is desirable to avoid incorporating platform-specific biases on the annotation of the collected datasets, and in turn through the training process of end-to-end models; especially in cases where well-established analytical solutions exists (as in the case of the tunable motion planning of Ch.5 and the 3D-geometry based pre-/post-processing of Ch.7). From this perspective, modular pipelines, comprising a mix of deep learning and analytical components that remain *post-training adaptable* to different application needs and device requirements, are preferable to end-to-end counterparts where possible.

8.2.5 Target Tasks/Models

Finally, over the past decade *image classification* has been the main vehicle of progress in the deep learning field, partly due to the accuracy-centric standardisation introduced by the ILSVRC benchmark. Although other vision tasks, such as detection, segmentation and depth/pose estimation commonly re-use backbones originally developed and trained for classification, such tasks pose their own unique challenges and opportunities for efficiency. However, most of the approximation methodologies proposed in the literature focus on image classification, leaving the challenges of more complex architectures, such as their irregular computation pattern, workload distribution, custom operators and layers and quantification of uncertainty, largely unexplored [127].

Additionally, current approximation methodologies focus on monocular vision modules. Tasks such as robot vision usually rely on multi-modal data, and 3D representations such as RGB-D frames, point-clouds and voxel-grids. This introduces further diversity on the architectures of deep learning approaches, calling for support from approximation schemes. At the same

time, these modalities also offer unique customisation opportunities that can be exploited by application-specific approximation methodologies, such as the one introduced in Ch.5.

Furthermore, in the ever-changing field of deep learning, new model families are flourishing yearly. However, the adoption pace of these architectures from other fields is naturally slow, and actually only few of them finally become applicable across downstream tasks in the real-world. At the time of writing this chapter, Transformers [617][618], Diffusion Models [619] and Neural Radiance Fields (NeRF) [620] constitute the most prominent paradigms, progressively establishing their foundations in robotics (among other fields). These models pose unique deployment challenges, and efforts for their efficient deployment have started to appear in the literature [621][622][623][624][625], while plenty of customisation/approximation opportunities remain unexplored to date.

8.3 Closing Remarks

The field of deep learning is progressing at an unprecedented pace. One force component driving these developments is (and will remain) accuracy-centric, exploiting the constantly increasing computational power for training at the datacentre, resulting to hyper-scale models offering a generic latent space that can act as shared foundation across several tasks [41].

On the other hand, the advanced visual understanding capabilities of deep learning models, continuously attract the attention of new use-cases and applications to adopt the latest advancements, setting new standards on intelligence and autonomy in the real world. Robotics has been one of the most drastically affected applications, with deep learning models equipping autonomous agents with unparalleled visual perception and understanding capabilities. This acted as an enabler for their real-world deployment in the consumer and commercial landscape, rendering mobile robots and UAVs as one of the most prominent and disruptive technologies for the years to come [71].

With the evolution of hardware performance being orders of magnitude slower than the workload requirements of the latest deep learning revelations, domain/application-specific customisation methodologies and dynamic inference schemes such as approximate and input-dependent inference are expected to play a fundamental role in the real-world deployment of deep learning systems in the years to come, towards robot visual intelligence...

Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning". In: *Nature* 521.7553 (2015) (pp. 2).

[2] J. Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural networks* 61 (2015) (pp. 2).

[3] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (pp. 2, 35, 56, 153, 217, 231).

[4] S. Ren, K. He, R. Girshick, and J. Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in Neural Information Processing Systems (NeurIPS)* 28 (2015) (pp. 2, 39, 131, 133, 135, 136, 141, 142, 153).

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. "You only look once: Unified, real-time object detection". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (pp. 2, 39, 132).

[6] V. Badrinarayanan, A. Kendall, and R. Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 39.12 (2017) (pp. 2, 40, 208).

[7] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *IEEE transactions on Pattern Analysis and Machine Intelligence (PAMI)* 40.4 (2017) (pp. 2, 40).

[8] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009 (pp. 2, 24).

[9] S. Madden. "From databases to big data". In: *IEEE Internet Computing* 16.3 (2012) (pp. 2).

[10] D. G. Lowe. "Object recognition from local scale-invariant features". In: *IEEE International Conference on Computer Vision (ICCV)*. Vol. 2. IEEE. 1999 (pp. 2).

[11] A. L. Samuel. "Some studies in machine learning using the game of checkers". In: *IBM Journal of Research and Development* 3.3 (1959) (pp. 2).

[12] A. Karpathy. *Software 2.0*. 2017. URL: https://karpathy.medium.com/software-2-0-a64152b37c35 (visited on 09/30/2020) (pp. 2).

[13] P. Warden. *Deep Learning is eating software*. 2017. URL: https://petewarden.com/2017/11/13/deep-learning-is-eating-software/ (visited on 09/30/2020) (pp. 2).

[14] N. P. Jouppi, D. H. Yoon, G. Kurian, S. Li, N. Patil, J. Laudon, C. Young, and D. Patterson. "A domain-specific supercomputer for training deep neural networks". In: *Communications of the ACM* 63.7 (2020) (pp. 2).

[15] J. Chen and X. Ran. "Deep learning with edge computing: A review". In: *Proceedings of the IEEE* 107.8 (2019) (pp. 2, 3).

[16] K. Fukushima and S. Miyake. “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition”. In: *US-Japan Joint Seminar On Competition and Cooperation in Neural Nets*. Springer. 1982 (pp. 2).

[17] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. “Handwritten digit recognition with a back-propagation network”. In: *Advances in Neural Information Processing Systems (NeurIPS)* 2 (1989) (pp. 2).

[18] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989) (pp. 2, 24, 26).

[19] L. Deng et al. “Recent advances in deep learning for speech research at Microsoft”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2013 (pp. 2).

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. 2012 (pp. 2, 33, 82, 87, 88, 177, 180, 186).

[21] J. Deng, W. Dong, R. Socher, L. Li, and and. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009 (pp. 2, 9, 34, 68, 82, 152, 180, 231).

[22] A. Geiger, P. Lenz, and R. Urtasun. “Are we ready for autonomous driving? The KITTI vision benchmark suite”. In: *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012 (pp. 2, 116).

[23] J. Nickolls, I. Buck, M. Garland, and K. Skadron. “Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for?” In: *Queue* 6.2 (2008) (pp. 2, 3).

[24] J. L. Hennessy and D. A. Patterson. “A new golden age for computer architecture”. In: *Communications of the ACM* 62.2 (2019) (pp. 2).

[25] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter. “Nvidia tensor core programmability, performance & precision”. In: *IEEE International Parallel and Distributed Processing Symposium Workshops*. IEEE. 2018 (pp. 2, 250).

[26] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. “DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015 (pp. 3, 6, 117).

[27] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. “Dermatologist-level classification of skin cancer with deep neural networks”. In: *Nature* 542.7639 (2017) (pp. 3).

[28] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik. “Deep neural nets as a method for quantitative structure–activity relationships”. In: *Journal of Chemical Information and Modeling* 55.2 (2015) (pp. 3).

[29] D. Bahdanau, K. Cho, and Y. Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014) (pp. 3).

[30] A. Hannun et al. “Deep speech: Scaling up end-to-end speech recognition”. In: *arXiv preprint arXiv:1412.5567* (2014) (pp. 3).

[31] D. Silver et al. “Mastering the game of go without human knowledge”. In: *Nature* 550.7676 (2017) (pp. 3).

[32] S. Levine, C. Finn, T. Darrell, and P. Abbeel. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research (JMLR)* 17.1 (2016) (pp. 3).

[33] A. Canziani, E. Culurciello, and A. Paszke. "Evaluation of neural network architectures for embedded systems". In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2017 (pp. 3).

[34] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein. "On the expressive power of deep neural networks". In: *International Conference on Machine Learning (ICML)*. PMLR. 2017 (pp. 3).

[35] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le. "Self-training with noisy student improves imagenet classification". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (pp. 3).

[36] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations (ICLR)* (2015) (pp. 3, 34, 56, 82, 88).

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going Deeper with Convolutions". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (pp. 3, 35).

[38] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. "Densely connected convolutional networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 3).

[39] C. A. Mack. "Fifty years of Moore's law". In: *IEEE Transactions on Semiconductor Manufacturing* 24.2 (2011) (pp. 3).

[40] L. Eeckhout. "Is Moore's law slowing down? What's next?" In: *IEEE/ACM International Symposium on Microarchitecture (MICRO)* 37.04 (2017) (pp. 3).

[41] R. Bommasani et al. "On the opportunities and risks of foundation models". In: *arXiv preprint arXiv:2108.07258* (2021) (pp. 3, 259).

[42] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso. "The computational limits of deep learning". In: *arXiv preprint arXiv:2007.05558* (2020) (pp. 3, 7, 45).

[43] J.-W. Jang et al. "Sparsity-aware and re-configurable NPU architecture for Samsung flagship mobile SoC". In: *International Symposium on Computer Architecture (ISCA)*. IEEE. 2021 (pp. 3).

[44] N. P. Jouppi et al. "In-datacenter performance analysis of a tensor processing unit". In: *International Symposium on Computer Architecture (ISCA)*. 2017 (pp. 3, 120).

[45] S. Laskaridis, S. I. Venieris, A. Kouris, R. Li, and N. D. Lane. "The Future of Consumer Edge-AI Computing". In: *arXiv preprint arXiv:2210.10514* (2022) (pp. 3).

[46] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc. "Design of ion-implanted MOSFET's with very small physical dimensions". In: *IEEE Journal of Solid-State Circuits* 9.5 (1974) (pp. 3).

[47] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. "Dark silicon and the end of multicore scaling". In: *International Symposium on Computer Architecture (ISCA)*. 2011 (pp. 3).

[48] N. D. Lane and P. Warden. "The deep (learning) transformation of mobile and embedded computing". In: *Computer* 51.5 (2018) (pp. 3).

[49] M. Ham et al. "NNStreamer: Efficient and Agile Development of On-Device AI Systems". In: *IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice*. IEEE. 2021 (pp. 3).

[50] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017) (pp. 3, 4, 36).

[51] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size". In: *arXiv preprint arXiv:1602.07360* (2016) (pp. 3, 47).

[52] M. Tan and Q. Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *International Conference on Machine Learning (ICML)*. PMLR. 2019 (pp. 4, 257).

[53] S. Sadiq, P. Maji, J. Hare, and G. Merrett. *Deff-arts: Differentiable efficient architecture search*. 2020 (pp. 4).

[54] M. Soltanolkotabi, A. Javanmard, and J. D. Lee. "Theoretical insights into the optimization landscape of over-parameterized shallow neural networks". In: *IEEE Transactions on Information Theory* 65.2 (2018) (pp. 4).

[55] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. "Incremental network quantization: Towards lossless cnns with low-precision weights". In: *International Conference on Learning Representations (ICLR)* (2017) (pp. 4, 57, 60).

[56] S. Han, J. Pool, J. Tran, and W. Dally. "Learning Both Weights and Connections for Efficient Neural Network". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2015 (pp. 4, 51, 57).

[57] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li. "Coordinating filters for faster deep neural networks". In: *IEEE International Conference on Computer Vision (ICCV)*. 2017 (pp. 4).

[58] G. Hinton, O. Vinyals, and J. Dean. "Distilling the knowledge in a neural network". In: *NeurIPS Deep Learning and Representation Learning Workshop* (2015) (pp. 4, 48).

[59] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo. "Compute and energy consumption trends in deep learning inference". In: *arXiv preprint arXiv:2109.05472* (2021) (pp. 4).

[60] Y. Guan, G. Sun, Z. Yuan, X. Li, N. Xu, S. Chen, J. Cong, and Y. Xie. "Crane: mitigating accelerator under-utilization caused by sparsity irregularities in cnns". In: *IEEE Transactions on Computers* 69.7 (2020) (pp. 5).

[61] H. Cai, L. Zhu, and S. Han. "Proxylessnas: Direct neural architecture search on target task and hardware". In: *International Conference on Learning Representations (ICLR)* (2019) (pp. 5, 48).

[62] L. Dudziak, T. Chau, M. Abdelfattah, R. Lee, H. Kim, and N. Lane. "Brp-nas: Prediction-based nas using gcns". In: *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020) (pp. 5).

[63] K. Hazelwood et al. "Applied machine learning at facebook: A datacenter infrastructure perspective". In: *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2018 (pp. 5, 32).

[64] J. Hennessy and D. Patterson. "A new golden age for computer architecture: domain-specific hardware/software co-design, enhanced". In: *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. 2018 (pp. 5).

[65] S. I. Venieris and C.-S. Bouganis. "fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs". In: *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2016 (pp. 5, 9, 44, 58, 76, 77).

[66] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. "EIE: Efficient inference engine on compressed deep neural network". In: *ACM SIGARCH Computer Architecture News* 44.3 (2016) (pp. 5, 41).

[67] J. Fowers et al. “A configurable cloud-scale DNN processor for real-time AI”. In: *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. IEEE. 2018 (pp. 5).

[68] C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl. “There’s plenty of room at the Top: What will drive computer performance after Moore’s law?” In: *Science* 368.6495 (2020) (pp. 5).

[69] M. S. Abdelfattah, Ł. Dudziak, T. Chau, R. Lee, H. Kim, and N. D. Lane. “Best of both worlds: Automl codesign of a cnn and its hardware accelerator”. In: *ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2020 (pp. 5, 48).

[70] N. Sünderhauf et al. “The limits and potentials of deep learning for robotics”. In: *The International Journal of Robotics Research (IJRR)* 37.4-5 (2018) (pp. 6, 53, 168, 253).

[71] G.-Z. Yang et al. “The grand challenges of science robotics”. In: *Science Robotics* 3.14 (2018) (pp. 6, 259).

[72] A. Loquercio, A. I. Maqueda, C. R. del-Blanco, and D. Scaramuzza. “DroNet: Learning to Fly by Driving”. In: *IEEE Robotics and Automation Letters (RA-L)* 3.2 (2018) (pp. 6, 131, 165, 168, 170, 171, 175, 184).

[73] I. Ahmed, S. Din, G. Jeon, F. Piccialli, and G. Fortino. “Towards collaborative robotics in top view surveillance: A framework for multiple object tracking by detection using deep learning”. In: *IEEE/CAA Journal of Automatica Sinica* 8.7 (2021) (pp. 6).

[74] P. V. Amadori, T. Fischer, R. Wang, and Y. Demiris. “Decision anticipation for driving assistance systems”. In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2020 (pp. 6).

[75] E. Johns, S. Leutenegger, and A. J. Davison. “Deep learning a grasp function for grasping under gripper pose uncertainty”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016 (pp. 6, 171).

[76] N. Sünderhauf, F. Dayoub, S. McMahon, B. Talbot, R. Schulz, P. Corke, G. Wyeth, B. Upcroft, and M. Milford. “Place categorization and semantic mapping on a mobile robot”. In: *IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016 (pp. 6, 90).

[77] J. McCormac, A. Handa, A. Davison, and S. Leutenegger. “Semanticfusion: Dense 3d Semantic Mapping with Convolutional Neural Networks”. In: *International Conference on Robotics and Automation (ICRA)*. 2017 (pp. 6, 10, 79, 203).

[78] Y. Alghamdi, A. Munir, and H. M. La. “Architecture, classification, and applications of contemporary unmanned aerial vehicles”. In: *IEEE Consumer Electronics Magazine (CEM)* 10.6 (2021) (pp. 6).

[79] M. B. Bejiga, A. Zeggada, A. Nouffidj, and F. Melgani. “A convolutional neural network approach for assisting avalanche search and rescue operations with uav imagery”. In: *Remote Sensing* 9.2 (2017) (pp. 6).

[80] H. X. Pham, H. M. La, D. Feil-Seifer, and M. Deans. “A distributed control framework for a team of unmanned aerial vehicles for dynamic wildfire tracking”. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017 (pp. 7).

[81] S. W. Chen, S. S. Shivakumar, S. Dcunha, J. Das, E. Okon, C. Qu, C. J. Taylor, and V. Kumar. “Counting apples and oranges with deep learning: a data-driven approach”. In: *IEEE Robotics and Automation Letters (RA-L)* 2.2 (2017) (pp. 7).

[82] T. Ikeda, S. Yasui, M. Fujihara, K. Ohara, S. Ashizawa, A. Ichikawa, A. Okino, T. Oomichi, and T. Fukuda. “Wall contact by octo-rotor UAV with one DoF manipulator for bridge inspection”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 (pp. 7).

[83] F. Mohammed, A. Idries, N. Mohamed, J. Al-Jaroodi, and I. Jawhar. "UAVs for smart cities: Opportunities and challenges". In: *International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2014 (pp. 7).

[84] O. Cetinkaya and G. V. Merrett. "Efficient deployment of UAV-powered sensors for optimal coverage and connectivity". In: *IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2020 (pp. 7).

[85] D. Jakubovitz, R. Giryes, and M. R. Rodrigues. "Generalization error in deep learning". In: *International MATHEON Conference on Compressed Sensing and Its Applications*. Springer. 2019 (pp. 7).

[86] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza. "Are we ready for autonomous drone racing? the UZH-FPV drone racing dataset". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2019 (pp. 7).

[87] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. "Scene parsing through ade20k dataset". In: *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 7).

[88] H. Pham, Z. Dai, Q. Xie, and Q. V. Le. "Meta pseudo labels". In: *IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR)*. 2021 (pp. 7).

[89] E. Faniadis and A. Amanatiadis. "Deep learning inference at the edge for mobile and aerial robotics". In: *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE. 2020 (pp. 7).

[90] M. Satyanarayanan. "The emergence of edge computing". In: *Computer* 50.1 (2017) (pp. 7, 47).

[91] A. Carrio, C. Sampedro, A. Rodriguez-Ramos, and P. Campoy. "A Review of Deep Learning Methods and Applications for Unmanned Aerial Vehicles". In: *Journal of Sensors* 2017 (2017) (pp. 7).

[92] B. Boroujerdian, H. Genc, S. Krishnan, W. Cui, A. Faust, and V. Reddi. "Mavbench: Micro aerial vehicle benchmarking". In: *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. 2018 (pp. 8).

[93] M. Bouhali, F. Shamani, Z. E. Dahmane, A. Belaidi, and J. Nurmi. "FPGA applications in unmanned aerial vehicles-a review". In: *International Symposium on Applied Reconfigurable Computing (ARC)*. Springer. 2017 (pp. 8).

[94] P. De Petris, H. Nguyen, M. Dharmadhikari, M. Kulkarni, N. Khedekar, F. Mascarich, and K. Alexis. "Rmf-owl: A collision-tolerant flying robot for autonomous subterranean exploration". In: *International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2022 (pp. 8).

[95] M. Blott, L. Halder, M. Leeser, and L. Doyle. "Qutibench: Benchmarking neural networks on heterogeneous hardware". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 15.4 (2019) (pp. 8).

[96] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield. "Toward Low-Flying Autonomous MAV Trail Navigation Using Deep Neural Networks for Environmental Awareness". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 (pp. 8, 131, 170).

[97] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu. "A survey of deep learning techniques for autonomous driving". In: *Journal of Field Robotics (JFR)* 37.3 (2020) (pp. 8).

[98] A. K. Singh, S. Dey, K. McDonald-Maier, K. R. Basireddy, G. V. Merrett, and B. M. Al-Hashimi. "Dynamic energy and thermal management of multi-core mobile platforms: A survey". In: *IEEE Design & Test* 37.5 (2020) (pp. 8, 52).

[99] C. Wang, H.-S. Fang, M. Gou, H. Fang, J. Gao, and C. Lu. "Graspness discovery in clutters for fast and accurate grasp detection". In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021 (pp. 9, 197).

[100] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza. "Deep drone racing: Learning agile flight in dynamic environments". In: *Conference on Robot Learning (CoRL)*. PMLR. 2018 (pp. 9).

[101] A. Rosinol, A. Violette, M. Abate, N. Hughes, Y. Chang, J. Shi, A. Gupta, and L. Carlone. "Kimera: From SLAM to spatial perception with 3D dynamic scene graphs". In: *International Journal of Robotics Research (IJRR)* 40.12-14 (2021) (pp. 9, 203, 253).

[102] L. Shao, Z. Cai, L. Liu, and K. Lu. "Performance evaluation of deep feature learning for RGB-D image/video classification". In: *Information Sciences* 385 (2017) (pp. 9).

[103] H. Ye, Z. Wu, R.-W. Zhao, X. Wang, Y.-G. Jiang, and X. Xue. "Evaluating two-stream CNN for video classification". In: *ACM International Conference on Multimedia Retrieval*. 2015 (pp. 9).

[104] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. "Learning spatiotemporal features with 3d convolutional networks". In: *IEEE International Conference on Computer Vision (ICCV)*. 2015 (pp. 9).

[105] L. Yang, Y. Han, X. Chen, S. Song, J. Dai, and G. Huang. "Resolution adaptive networks for efficient inference". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (pp. 9, 52, 251, 254).

[106] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. "Designing Efficient DNN Models". In: *Efficient Processing of Deep Neural Networks*. Cham: Springer International Publishing, 2020. URL: https://doi.org/10.1007/978-3-031-01766-7_9 (pp. 9, 46, 47, 50).

[107] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. "Mnasnet: Platform-aware neural architecture search for mobile". In: *IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (pp. 9).

[108] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han. "Once-for-all: Train one network and specialize it for efficient deployment". In: *International Conference on Learning Representations (ICLR)* (2020) (pp. 9, 249).

[109] V. J. Reddi et al. "Mlperf inference benchmark". In: *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. IEEE. 2020 (pp. 9).

[110] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. "Nas-bench-101: Towards reproducible neural architecture search". In: *International Conference on Machine Learning (ICML)*. PMLR. 2019 (pp. 9).

[111] P. Micikevicius et al. "FP8 formats for deep learning". In: *arXiv preprint arXiv:2209.05433* (2022) (pp. 9).

[112] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. "XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks". In: *European Conference on Computer Vision (ECCV)*. Springer. 2016 (pp. 9, 50).

[113] R. Lee, S. Venieris, Ł. Dudziak, S. Bhattacharya, and N. Lane. "MobiSR: Efficient On-Device Super-Resolution through Heterogeneous Mobile Processors". In: *Mobile Computing and Networking (MobiCom)*. 2019 (pp. 9, 61).

[114] C. Kyrkou, G. Plastiras, T. Theocharides, S. I. Venieris, and C. Bouganis. "DroNet: Efficient convolutional neural network detector for real-time UAV applications". In: *Design, Automation and Test in Europe Conference Exhibition (DATE)*. 2018 (pp. 10, 136, 142, 143, 250).

[115] Udacity. *Introduction to Self-Driving Cars*. 2017. URL: https://www.udacity.com/course/intro-to-self-driving-cars--nd113 (visited on 09/30/2020) (pp. 10).

[116] A. Kouris, S. I. Venieris, M. Rizakis, and C.-S. Bouganis. "Approximate LSTMs for time-constrained inference: Enabling fast reaction in self-driving cars". In: *IEEE Consumer Electronics Magazine (CEM)* 9.4 (2020) (pp. 14).

[117] S. I. Venieris, A. Kouris, and C.-S. Bouganis. "Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions". In: *ACM Computing Surveys (CSUR)* 51.3 (2018) (pp. 14, 44, 58, 89).

[118] A. Kouris, S. I. Venieris, and C.-S. Bouganis. "A throughput-latency co-optimised cascade of convolutional neural network classifiers". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2020 (pp. 14).

[119] A. Kouris, C. Kyrkou, and C.-S. Bouganis. "Informed region selection for efficient uav-based object detectors: Altitude-aware vehicle detection with cycar dataset". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019 (pp. 14).

[120] A. Kouris, S. I. Venieris, and C.-S. Bouganis. "Towards efficient on-board deployment of DNNs on intelligent autonomous systems". In: *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE. 2019 (pp. 14).

[121] A. Kouris, S. I. Venieris, and C. Bouganis. "CascadeCNN: Pushing the Performance Limits of Quantisation in Convolutional Neural Networks". In: *Internaltional Conference on Field Programmable Logic and Applications (FPL)*. 2018 (pp. 14).

[122] A. Kouris and C. Bouganis. "Learning to Fly by MySelf: A Self-Supervised CNN-Based Approach for Autonomous Navigation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018 (pp. 14).

[123] A. Kouris, S. I. Venieris, and C. Bouganis. "CascadeCNN: Pushing the Performance Limits of Quantisation". In: *Systems and Machine Learning (SysML)*. 2018 (pp. 15).

[124] M. Rizakis, S. I. Venieris, A. Kouris, and C.-S. Bouganis. "Approximate FPGA-Based LSTMs Under Computation Time Constraints". In: *International Symposium on Applied Reconfigurable Computing (ARC)*. Springer. 2018 (pp. 15).

[125] S. I. Venieris, A. Kouris, and C.-S. Bouganis. "Deploying deep neural networks in the embedded space". In: EMDL '18 (2018) (pp. 15).

[126] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes". In: *arXiv preprint arXiv:1711.00199* (2017) (pp. 15, 209, 211, 222, 224, 231).

[127] A. Kouris, S. I. Venieris, S. Laskaridis, and N. Lane. "Multi-exit semantic segmentation networks". In: *European Conference on Computer Vision (ECCV)*. Springer. 2022 (pp. 15, 249, 258).

[128] S. Laskaridis, A. Kouris, and N. D. Lane. "Adaptive inference through early-exit networks: Design, challenges and directions". In: *International Workshop on Embedded and Mobile Deep Learning (EMDL)*. 2021 (pp. 15, 52, 249).

[129] M. Elgendy. *Deep learning for vision systems*. Simon and Schuster, 2020 (pp. 19, 26).

[130] Z. Zhang. "Flexible camera calibration by viewing a plane from unknown orientations". In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. Vol. 1. Ieee. 1999 (pp. 20).

[131] R. Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022 (pp. 20, 202).

[132] P. Beeson, J. Modayil, and B. Kuipers. "Factoring the mapping problem: Mobile robot map-building in the hybrid spatial semantic hierarchy". In: *International Journal of Robotics Research (IJRR)* 29.4 (2010) (pp. 21).

[133] M. Naseer, S. Khan, and F. Porikli. "Indoor scene understanding in 2.5/3d for autonomous agents: A survey". In: *IEEE access* 7 (2018) (pp. 21, 197, 202).

[134] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003 (pp. 22).

[135] D. H. Wolpert and W. G. Macready. "No free lunch theorems for optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997) (pp. 23).

[136] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1. Ieee. 2005 (pp. 23).

[137] T. Lindeberg. "Scale invariant feature transform". In: (2012) (pp. 23).

[138] H. Bay, T. Tuytelaars, and L. Van Gool. "Surf: Speeded up robust features". In: *European Conference on Computer Vision (ECCV)* (2006) (pp. 23).

[139] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. "Support vector machines". In: *IEEE Intelligent Systems and their Applications* 13.4 (1998) (pp. 24).

[140] J. R. Quinlan. "Induction of decision trees". In: *Machine learning* 1 (1986) (pp. 24).

[141] Y. Freund, R. Schapire, and N. Abe. "A short introduction to boosting". In: *Journal-Japanese Society for Artificial Intelligence* 14.771-780 (1999) (pp. 24).

[142] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1. Ieee. 2001 (pp. 24).

[143] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016 (pp. 24, 26, 180).

[144] Y. Bengio, A. Courville, and P. Vincent. "Representation learning: A review and new perspectives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 35.8 (2013) (pp. 24).

[145] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. "What is the best multi-stage architecture for object recognition?" In: *IEEE International Conference on Computer Vision (ICCV)*. IEEE. 2009 (pp. 24).

[146] S. Haykin. *Neural networks and learning machines, 3/E*. Pearson Education India, 2009 (pp. 25).

[147] D. H. Hubel and T. N. Wiesel. "Receptive fields and functional architecture of monkey striate cortex". In: *The Journal of physiology* 195.1 (1968) (pp. 25).

[148] S. Ghosh-Dastidar and H. Adeli. "Spiking neural networks". In: *International Journal of Neural Systems (IJNS)* 19.04 (2009) (pp. 25).

[149] F. Akopyan et al. "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 34.10 (2015) (pp. 25).

[150] Y. Bengio et al. "Learning deep architectures for AI". In: *Foundations and trends® in Machine Learning* 2.1 (2009) (pp. 25).

[151] G. Strang et al. *Linear algebra and learning from data*. Vol. 4. Wellesley-Cambridge Press Cambridge, 2019 (pp. 25).

[152] G. E. Hinton and R. R. Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006) (pp. 26).

[153] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. "Phoneme recognition using time-delay neural networks". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing (ICASSP)* 37.3 (1989) (pp. 26).

[154] V. Nair and G. E. Hinton. "Rectified linear units improve restricted boltzmann machines". In: *International Conference on Machine Learning (ICML)*. 2010 (pp. 28).

[155] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. "Striving for simplicity: The all convolutional net". In: *arXiv preprint arXiv:1412.6806* (2014) (pp. 28).

[156] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997) (pp. 29, 97).

[157] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". In: *Advances in Neural Information Processing Systems (NeurIPS)* 28 (2015) (pp. 30).

[158] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak. "Convolutional, long short-term memory, fully connected deep neural networks". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Ieee. 2015 (pp. 30).

[159] C.-J. Wu et al. "Machine learning at facebook: Understanding inference at the edge". In: *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2019 (pp. 31, 47).

[160] L. Bottou et al. "Stochastic gradient learning in neural networks". In: *Proceedings of Neuro-Nımes* 91.8 (1991) (pp. 31).

[161] N. Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1 (1999) (pp. 32).

[162] Y. Nesterov. "A method for unconstrained convex minimization problem with the rate of convergence O (1/kˆ 2)". In: *Doklady an ussr*. Vol. 269. 1983 (pp. 32).

[163] J. Duchi, E. Hazan, and Y. Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of Machine Learning Research (JMLR)* 12.7 (2011) (pp. 32).

[164] S. Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016) (pp. 32).

[165] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014) (pp. 32).

[166] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986) (pp. 32).

[167] R. Hecht-Nielsen. "Theory of the backpropagation neural network". In: *Neural networks for perception*. Elsevier, 1992 (pp. 32).

[168] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, 2002 (pp. 32).

[169] A. Ng. "Machine learning yearning". In: *URL: http://www. mlyearning. org/(96)* 139 (2017) (pp. 32).

[170] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *Journal of Machine Learning Research (JMLR)* 15.1 (2014) (pp. 32).

[171] S. Ioffe and C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International Conference on Machine Learning (ICML)*. pmlr. 2015 (pp. 32).

[172] A. Krogh and J. Hertz. "A simple weight decay can improve generalization". In: *Advances in Neural Information Processing Systems (NeurIPS)* 4 (1991) (pp. 32).

[173] J. Gu et al. "Recent advances in convolutional neural networks". In: *Pattern Recognition* 77 (2018) (pp. 32).

[174] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger. "Multi-Scale Dense Networks for Resource Efficient Image Classification". In: *International Conference on Learning Representations (ICLR)*. 2018 (pp. 33, 90, 103, 254).

[175] S. Teerapittayanon, B. McDanel, and H.-T. Kung. "BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks". In: *Intenational Conference on Pattern Recognition (ICPR)*. 2016 (pp. 33, 52, 61, 103).

[176] A. J. Joshi, F. Porikli, and N. Papanikolopoulos. "Multi-Class Active Learning for Image Classification". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009 (pp. 33, 67).

[177] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE*. 1998 (pp. 33).

[178] L. Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012) (pp. 33).

[179] G. A. Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (1995) (pp. 34).

[180] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *AAAI Conference on Artificial Intelligence*. 2017 (pp. 35, 36, 153).

[181] M. Lin, Q. Chen, and S. Yan. "Network in network". In: *arXiv preprint arXiv:1312.4400* (2013) (pp. 36).

[182] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 36, 153).

[183] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. "Learning transferable architectures for scalable image recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 36).

[184] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. "Revisiting unreasonable effectiveness of data in deep learning era". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 36).

[185] A. Brock, S. De, S. L. Smith, and K. Simonyan. "High-performance large-scale image recognition without normalization". In: *International Conference on Machine Learning (ICML)*. PMLR. 2021 (pp. 36).

[186] R. Salman and V. Kecman. "Regression as classification". In: *2012 Proceedings of IEEE Southeastcon*. IEEE. 2012 (pp. 37).

[187] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. "How transferable are features in deep neural networks?" In: *Advances in Neural Information Processing Systems (NeurIPS)* 27 (2014) (pp. 37).

[188] H. W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval research logistics quarterly* 2.1-2 (1955) (pp. 38).

[189] A. Neubeck and L. Van Gool. "Efficient non-maximum suppression". In: *International Conference on Pattern Recognition (ICPR)*. Vol. 3. IEEE. 2006 (pp. 38).

[190] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. "Microsoft COCO: Common Objects in Context". In: *European Conference on Computer Vision (ECCV)*. 2014 (pp. 38, 136, 152, 231).

[191] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. "Ssd: Single Shot Multibox Detector". In: *European Conference on Computer Vision (ECCV)*. 2016 (pp. 39, 132, 134, 153).

[192] J. Redmon and A. Farhadi. "YOLO9000: Better, Faster, Stronger". In: *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 39, 134).

[193] J. Redmon and A. Farhadi. "Yolov3: An incremental improvement". In: *arXiv preprint arXiv:1804.02767* (2018) (pp. 39).

[194] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. "Focal loss for dense object detection". In: *IEEE International Conference on Computer Vision*. 2017 (pp. 39).

[195] J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (pp. 40).

[196] H. Noh, S. Hong, and B. Han. "Learning deconvolution network for semantic segmentation". In: *IEEE International Conference on Computer Vision (ICCV)*. 2015 (pp. 40).

[197] O. Ronneberger, P. Fischer, and T. Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer. 2015 (pp. 40, 211, 217).

[198] F. Yu, V. Koltun, and T. Funkhouser. "Dilated residual networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 40).

[199] C. Zhao, Q. Sun, C. Zhang, Y. Tang, and F. Qian. "Monocular depth estimation based on deep learning: An overview". In: *Science China Technological Sciences* 63.9 (2020) (pp. 40, 217).

[200] F. Milletari, N. Navab, and S.-A. Ahmadi. "V-net: Fully convolutional neural networks for volumetric medical image segmentation". In: *International Conference on 3D Vision (3DV)*. Ieee. 2016 (pp. 40).

[201] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M Jorge Cardoso. "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations". In: *International Workshop on Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Springer. 2017 (pp. 40).

[202] K. He, G. Gkioxari, P. Dollar, and R. Girshick. "Mask R-CNN". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2018) (pp. 40, 131, 210, 231).

[203] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. "Feature pyramid networks for object detection". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 40).

[204] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee. "Yolact: Real-time instance segmentation". In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019 (pp. 40).

[205] J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011 (pp. 41).

[206] V. Vanhoucke, A. Senior, and M. Z. Mao. "Improving the speed of neural networks on CPUs". In: (2011) (pp. 41).

[207] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma. "Always-on vision processing unit for mobile applications". In: *IEEE/ACM International Symposium on Microarchitecture (MICRO)* 35.2 (2015) (pp. 41).

[208] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li. "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks". In: *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2017 (pp. 41).

[209] J. P. Huber and M. W. Rosneck. *Successful ASIC design the first time through*. Springer, 1991 (pp. 41).

[210] O. Mencer et al. "The History, Status, and Future of FPGAs: Hitting a nerve with field-programmable gate arrays". In: *Queue* 18.3 (2020) (pp. 41, 44).

[211] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart. *The Zynq book: embedded processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 all programmable SoC*. Strathclyde Academic Media, 2014 (pp. 42, 44).

[212] J. Sanders and E. Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010 (pp. 41).

[213] D. B. Kirk and W. H. Wen-Mei. *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann, 2016 (pp. 43).

[214] W.-m. Hwu, K. Keutzer, and T. G. Mattson. "The concurrency challenge". In: *IEEE Design & Test of Computers* 25.4 (2008) (pp. 43).

[215] N. Otterness, M. Yang, S. Rust, E. Park, J. H. Anderson, F. D. Smith, A. Berg, and S. Wang. "An evaluation of the NVIDIA TX1 for supporting real-time computer-vision workloads". In: *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE. 2017 (pp. 43).

[216] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093* (2014) (pp. 43).

[217] M. Abadi et al. "Tensorflow: a system for large-scale machine learning." In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Savannah, GA, USA. 2016 (pp. 43).

[218] A. Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in Neural Information Processing Systems (NeurIPS)* 32 (2019) (pp. 43).

[219] T. J. Todman, G. A. Constantinides, S. J. Wilton, O. Mencer, W. Luk, and P. Y. Cheung. "Reconfigurable computing: architectures and design methods". In: *IEE Proceedings- Computers and Digital Techniques* 152.2 (2005) (pp. 43).

[220] S. M. S. Trimberger. "Three ages of fpgas: a retrospective on the first thirty years of fpga technology: this paper reflects on how moore's law has driven the design of fpgas through three epochs: the age of invention, the age of expansion, and the age of accumulation". In: *IEEE Solid-State Circuits Magazine* 10.2 (2018) (pp. 44).

[221] E. Chung et al. "Serving dnns in real time at datacenter scale with project brainwave". In: *IEEE/ACM International Symposium on Microarchitecture (MICRO)* 38.2 (2018) (pp. 44).

[222] K. P. Seng, P. J. Lee, and L. M. Ang. "Embedded intelligence on FPGA: Survey, applications and challenges". In: *Electronics* 10.8 (2021) (pp. 44).

[223] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer. "Xilinx adaptive compute acceleration platform: VersalTM architecture". In: *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 2019 (pp. 44).

[224] P. Coussy and A. Morawiec. *High-level synthesis*. Vol. 1. Springer, 2010 (pp. 44).

[225] G. Inggs, S. Fleming, D. Thomas, and W. Luk. "Is high level synthesis ready for business? A computational finance case study". In: *International Conference on Field-Programmable Technology (FPT)*. IEEE. 2014 (pp. 44).

[226] M. Fingeroff. *High-level synthesis: blue book*. Xlibris Corporation, 2010 (pp. 44).

[227] G. Menghani. "Efficient deep learning: A survey on making deep learning models smaller, faster, and better". In: *ACM Computing Surveys (CSUR)* (2021) (pp. 45).

[228] E. Wang, J. J. Davis, R. Zhao, H.-C. Ng, X. Niu, W. Luk, P. Y. Cheung, and G. A. Constantinides. "Deep Neural Network Approximation for Custom Hardware: Where We've Been, Where We're Going". In: *ACM Computing Surveys (CSUR)* 52.2 (2019) (pp. 45, 57).

[229] A. Marchisio, M. A. Hanif, F. Khalid, G. Plastiras, C. Kyrkou, T. Theocharides, and M. Shafique. "Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges". In: *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE. 2019 (pp. 45).

[230] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang. "Edge intelligence: Paving the last mile of artificial intelligence with edge computing". In: *Proceedings of the IEEE* 107.8 (2019) (pp. 46).

[231] J. D. Little. "A proof for the queuing formula: L= λ W". In: *Operations research* 9.3 (1961) (pp. 46).

[232] R. Shokri and V. Shmatikov. "Privacy-preserving deep learning". In: *ACM SIGSAC Conference on Computer and Communications Security*. 2015 (pp. 47, 58, 98).

[233] J. R. Ruiz-Sarmiento, C. Galindo, and J. González-Jiménez. "Robot@ home, a robotic dataset for semantic mapping of home environments". In: *International Journal of Robotics Research (IJRR)* 36.2 (2017) (pp. 47).

[234] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. "Efficient processing of deep neural networks: A tutorial and survey". In: *Proceedings of the IEEE* 105.12 (2017) (pp. 47).

[235] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the inception architecture for computer vision". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (pp. 47).

[236] F. Yu, D. Wang, E. Shelhamer, and T. Darrell. "Deep layer aggregation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 47).

[237] X. Zhang, X. Zhou, M. Lin, and J. Sun. "Shufflenet: An extremely efficient convolutional neural network for mobile devices". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 47).

[238] J. Hu, L. Shen, and G. Sun. "Squeeze-and-excitation networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 47).

[239] L.-C. Chen, M. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens. "Searching for efficient multi-scale architectures for dense image prediction". In: *Advances in Neural Information Processing Systems (NeurIPS)* 31 (2018) (pp. 48).

[240] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. "Large-scale evolution of image classifiers". In: *International Conference on Machine Learning (ICML)*. PMLR. 2017 (pp. 48).

[241] B. Zoph and Q. V. Le. "Neural architecture search with reinforcement learning". In: *arXiv preprint arXiv:1611.01578* (2016) (pp. 48).

[242] A. Zela, A. Klein, S. Falkner, and F. Hutter. "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search". In: *arXiv preprint arXiv:1807.06906* (2018) (pp. 48).

[243] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi. "Morphnet: Fast & simple resource-constrained structure learning of deep networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 48).

[244] H. Liu, K. Simonyan, and Y. Yang. "Darts: Differentiable architecture search". In: *International Conference on Learning Representations (ICLR)* (2019) (pp. 48).

[245] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam. "Netadapt: Platform-aware neural network adaptation for mobile applications". In: *European Conference on Computer Vision (ECCV)*. 2018 (pp. 48).

[246] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun. "Dpp-net: Device-aware progressive search for pareto-optimal neural architectures". In: *European Conference on Computer Vision (ECCV)*. 2018 (pp. 48).

[247] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. "Fitnets: Hints for thin deep nets". In: *arXiv preprint arXiv:1412.6550* (2014) (pp. 48).

[248] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic. *Digital integrated circuits*. Vol. 2. Prentice hall Englewood Cliffs, 2002 (pp. 49).

[249] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. "A survey of quantization methods for efficient neural network inference". In: *arXiv preprint arXiv:2103.13630* (2021) (pp. 49).

[250] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. "Quantization and training of neural networks for efficient integer-arithmetic-only inference". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 50).

[251] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji. "High-performance FPGA-based CNN accelerator with block-floating-point arithmetic". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.8 (2019) (pp. 50).

[252] D. Williamson. "Dynamically scaled fixed point arithmetic". In: *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing Conference Proceedings*. IEEE. 1991 (pp. 50).

[253] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. "Binarized Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016 (pp. 50).

[254] Y. Zhang, J. Pan, X. Liu, H. Chen, D. Chen, and Z. Zhang. "FracBNN: Accurate and FPGA-efficient binary neural networks with fractional activations". In: *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 2021 (pp. 50).

[255] A. Elthakeb, P. Pilligundla, F. Mireshghallah, A. Yazdanbakhsh, S. Gao, and H. Esmaeilzadeh. "Releq: an automatic reinforcement learning approach for deep quantization of neural networks". In: *NeurIPS ML for Systems workshop*. 2019 (pp. 50).

[256] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han. "Haq: Hardware-aware automated quantization with mixed precision". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019 (pp. 50).

[257] A. Bulat and G. Tzimiropoulos. "Bit-Mixer: Mixed-precision networks with runtime bit-width selection". In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021 (pp. 50).

[258] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag. "What is the state of neural network pruning?" In: *Machine Learning and Systems Conference (MLSys)* 2 (2020) (pp. 50).

[259] S. A. Janowsky. "Pruning versus clipping in neural networks". In: *Physical Review A* 39.12 (1989) (pp. 51).

[260] Y. LeCun, J. Denker, and S. Solla. "Optimal brain damage". In: *Advances in Neural Information Processing Systems (NeurIPS)* 2 (1989) (pp. 51).

[261] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. "Learning structured sparsity in deep neural networks". In: *Advances in Neural Information Processing Systems (NeurIPS)* 29 (2016) (pp. 51).

[262] Y. He, X. Zhang, and J. Sun. "Channel pruning for accelerating very deep neural networks". In: *International Conference on Computer Vision*. 2017 (pp. 51).

[263] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke. "Scalpel: Customizing dnn pruning to the underlying hardware parallelism". In: *ACM SIGARCH Computer Architecture News* 45.2 (2017) (pp. 51).

[264] T.-J. Yang, Y.-H. Chen, and V. Sze. "Designing energy-efficient convolutional neural networks using energy-aware pruning". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 51).

[265] A.-H. Phan, K. Sobolev, K. Sozykin, D. Ermilov, J. Gusak, P. Tichavskỳ, V. Glukhov, I. Oseledets, and A. Cichocki. "Stable low-rank tensor decomposition for compression of convolutional neural network". In: *European Conference on Computer Vision (ECCV)*. Springer. 2020 (pp. 51).

[266] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. "Compression of deep convolutional neural networks for fast and low power mobile applications". In: *arXiv preprint arXiv:1511.06530* (2015) (pp. 51).

[267] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. "Speeding-up convolutional neural networks using fine-tuned cp-decomposition". In: *arXiv preprint arXiv:1412.6553* (2014) (pp. 51).

[268] L. Xun, L. Tran-Thanh, B. M. Al-Hashimi, and G. V. Merrett. "Optimising resource management for embedded machine learning". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2020 (pp. 52).

[269] L. Liu and J. Deng. "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution". In: *AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018 (pp. 52, 254).

[270] W. Lou, L. Xun, A. Sabet, J. Bi, J. Hare, and G. V. Merrett. "Dynamic-OFA: Runtime DNN architecture switching for performance scaling on heterogeneous embedded platforms". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021 (pp. 52, 249, 254).

[271] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang. "Adaptive deep learning model selection on embedded systems". In: *ACM SIGPLAN Notices* 53.6 (2018) (pp. 52).

[272] E. Park, D. Kim, S. Kim, Y.-D. Kim, G. Kim, S. Yoon, and S. Yoo. "Big/little deep neural network for ultra low power inference". In: *International Conference on Hardware/Software Codesign and System Synthesis*. IEEE. 2015 (pp. 52).

[273] X. Wang, Y. Luo, D. Crankshaw, A. Tumanov, F. Yu, and J. E. Gonzalez. "Idk cascades: Fast deep learning by learning not to overthink". In: *arXiv preprint arXiv:1706.00885* (2017) (pp. 52).

[274] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama. "Adaptive neural networks for efficient inference". In: *International Conference on Machine Learning (ICML)*. PMLR. 2017 (pp. 52, 63).

[275] G. Shu, W. Liu, X. Zheng, and J. Li. "IF-CNN: Image-aware inference framework for CNN with the collaboration of mobile devices and cloud". In: *IEEE Access* 6 (2018) (pp. 52).

[276] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang. "Dynamic neural networks: A survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 44.11 (2021) (pp. 52).

[277] X. Gao, Y. Zhao, Ł. Dudziak, R. Mullins, and C.-z. Xu. "Dynamic channel pruning: Feature boosting and suppression". In: *International Conference on Learning Representations (ICLR)* (2019) (pp. 52).

[278] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris. "Blockdrop: Dynamic inference paths in residual networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 52).

[279] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez. "Skipnet: Learning dynamic routing in convolutional networks". In: *European Conference on Computer Vision (ECCV)*. 2018 (pp. 52).

[280] A. M. López, A. Imiya, T. Pajdla, and J. M. Álvarez. *Computer vision in vehicle technology: Land, sea, and air*. John Wiley & Sons, 2017 (pp. 53).

[281] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. "Pixhawk: A system for autonomous flight using onboard computer vision". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2011 (pp. 53).

[282] D. Scaramuzza et al. "Vision-controlled micro flying robots: from system design to autonomous navigation and mapping in GPS-denied environments". In: *IEEE Robotics & Automation Magazine (RA-M)* 21.3 (2014) (pp. 53).

[283] M. D. Zeiler and R. Fergus. "Visualizing and understanding convolutional networks". In: *European Conference on Computer Vision (ECCV)*. Springer. 2014 (pp. 56).

[284] T. Abukhalil, H. Almahafzah, M. Alksasbeh, and B. A. Alqaralleh. "Power optimization in mobile robots using a real-time heuristic". In: *Journal of Robotics* 2020 (2020) (pp. 57).

[285] J. Qiu et al. "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network". In: *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 2016 (pp. 57, 77, 88).

[286] S. Han, H. Mao, and W. J. Dally. "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding". In: *International Conference on Learning Representations (ICLR)* (2016) (pp. 57).

[287] S. Han et al. "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA". In: *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 2017 (pp. 57, 98, 101, 102).

[288] M. Jaderberg, A. Vedaldi, and A. Zisserman. "Speeding up Convolutional Neural Networks with Low Rank Expansions". In: *British Machine Vision Conference (BMVC)*. 2014 (pp. 57).

[289] Y. Ma, Y. Cao, S. Vrudhula, and J. s. Seo. "An Automatic RTL Compiler for High-Throughput FPGA Implementation of Diverse Convolutional Neural Networks". In: *Internaltional Conference on Field Programmable Logic and Applications (FPL)*. 2017 (pp. 58, 60, 71, 83, 85, 86).

[290] Y. Umuroglu et al. "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference". In: *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 2017 (pp. 58, 60).

[291] P. Gysel, M. Motamedi, and S. Ghiasi. "Hardware-oriented Approximation of Convolutional Neural Networks". In: *International Conference on Learning Representations (ICLR)*. 2016 (pp. 58, 60).

[292] K. Guo et al. "Angel-Eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA". In: *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2018) (pp. 58, 60, 85, 86).

[293] M. J. Wainwright, M. Jordan, and J. C. Duchi. "Privacy aware learning". In: *Advances in Neural Information Processing Systems (NeurIPS)* 25 (2012) (pp. 58, 98).

[294] S. Wang, S.-C. S. Cheung, and H. Sajid. "Visual bubble: Protecting privacy in wearable cameras". In: *IEEE Consumer Electronics Magazine (CEM)* 7.1 (2017) (pp. 58).

[295] F. Firouzi, A. M. Rahmani, K. Mankodiya, M. Badaroglu, G. V. Merrett, P Wong, and B. Farahani. *Internet-of-Things and big data for smarter healthcare: From device to architecture, applications and analytics*. 2018 (pp. 58).

[296] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith. "Federated learning: Challenges, methods, and future directions". In: *IEEE Signal Processing Magazine* 37.3 (2020) (pp. 58).

[297] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda. "Understanding the impact of precision quantization on the accuracy and energy of neural networks". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2017 (pp. 60).

[298] S. I. Venieris and C.-S. Bouganis. "Latency-driven Design for FPGA-based Convolutional Neural Networks". In: *Internaltional Conference on Field Programmable Logic and Applications (FPL)*. 2017 (pp. 60, 80, 83, 85, 86, 88).

[299] N. Suda et al. "Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks". In: *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 2016 (pp. 60, 82).

[300] A. Prost-Boucle et al. "Scalable High-Performance Architecture for Convolutional Ternary Neural Networks on FPGA". In: *Internaltional Conference on Field Programmable Logic and Applications (FPL)*. 2017 (pp. 60).

[301] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. "Large-scale Video Classification with Convolutional Neural Networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014 (pp. 61, 171, 179).

[302] A. Almahairi et al. "Dynamic Capacity Networks". In: *International Conference on Machine Learning (ICML)*. 2016 (pp. 61).

[303] C. Gao, D. Neil, E. Ceolini, S.-C. Liu, and T. Delbruck. "DeltaRNN: A Power-efficient Recurrent Neural Network Accelerator". In: *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 2018 (pp. 61, 101, 102).

[304] Y. Kaya, S. Hong, and T. Dumitras. "Shallow-deep networks: Understanding and mitigating network overthinking". In: *International Conference on Machine Learning (ICML)*. PMLR. 2019 (pp. 61, 103).

[305] P. Viola and M. J. Jones. "Robust Real-Time Face Detection". In: *International Journal on Computer Vision (IJCV)* (2004) (pp. 62).

[306] Z. E. Xu, M. J. Kusner, K. Q. Weinberger, M. Chen, and O. Chapelle. "Classifier Cascades and Trees for Minimizing Feature Evaluation Cost". In: *Journal of Machine Learning Research (JMLR)* 15 (2014) (pp. 62).

[307] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua. "A Convolutional Neural Network Cascade for Face Detection". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (pp. 62).

[308] A. Angelova et al. "Real-Time Pedestrian Detection With Deep Network Cascades". In: *British Machine Vision Conference (BMVC)*. 2015 (pp. 62).

[309] A. Diba et al. "Weakly Supervised Cascaded Convolutional Networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 62).

[310] Z. Wei et al. "A Self-Adaptive Cascade ConvNets Model Based on Label Relation Mining". In: *Neurocomputing* 328 (2019) (pp. 62).

[311] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. "NoScope: Optimizing Neural Network Queries over Video at Scale". In: *Proceedings of the VLDB Endowment* 10.11 (2017) (pp. 62).

[312] S. Amiri, M. Hosseinabady, S. McIntosh-Smith, and J. Nunez-Yanez. "Multi-Precision Convolutional Neural Networks on Heterogeneous Hardware". In: *Design, Automation & Test in Europe Conf. (DATE)*. 2018 (pp. 63).

[313] W. Zhang, M. Jiang, and G. Luo. "Evaluating low-memory GEMMs for convolutional neural network inference on FPGAS". In: *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE. 2020 (pp. 71).

[314] D. Nguyen, D. Kim, and J. Lee. "Double MAC: Doubling the Performance of Convolutional Neural Networks on Modern FPGAs". In: *Design, Automation & Test in Europe Conference (DATE)*. 2017 (pp. 71, 80).

[315] S. Williams et al. “Roofline: An Insightful Visual Performance Model for Multicore Architectures”. In: *Communications of the ACM* 52.4 (2009) (pp. 72, 114, 119).

[316] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li. “DeepBurning: Automatic Generation of FPGA-based Learning Accelerators for the Neural Network Family”. In: *Design Automation Conference (DAC)*. 2016 (pp. 77, 88).

[317] V. Gokhale, A. Zaidy, A. X. M. Chang, and E. Culurciello. “Snowflake: An Efficient Hardware Accelerator for Convolutional Neural Networks”. In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. 2017 (pp. 77, 80, 88).

[318] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison. “Real-time Camera Tracking: When is High Frame-rate Best?” In: *European Conference on Computer Vision (ECCV)*. 2012 (pp. 79).

[319] K. Boikos and C. Bouganis. “A High-Performance System-on-Chip Architecture for Direct Tracking for SLAM”. In: *Internaltional Conference on Field Programmable Logic and Applications (FPL)*. 2017 (pp. 79).

[320] C. Zhang et al. “Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks”. In: *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 2015 (pp. 80).

[321] U. Aydonat, S. O’Connell, D. Capalija, A. C. Ling, and G. R. Chiu. “An OpenCL™Deep Learning Accelerator on Arria 10”. In: *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 2017 (pp. 85, 86).

[322] J. Zhang and J. Li. “Improving the Performance of OpenCL-based FPGA Accelerator for Convolutional Neural Network”. In: *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 2017 (pp. 85, 86).

[323] Y. Shen, M. Ferdman, and P. Milder. “Escher: A CNN Accelerator with Flexible Buffering to Minimize Off-Chip Transfer”. In: *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2017 (pp. 85, 86).

[324] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. “Learning deep features for scene recognition using places database”. In: *Advances in Neural Information Processing Systems (NeurIPS)* 27 (2014) (pp. 90).

[325] B. Zhou, A. Khosla, A. Lapedriza, A. Torralba, and A. Oliva. “Places: An image database for deep scene understanding”. In: *arXiv preprint arXiv:1610.02055* (2016) (pp. 90).

[326] I. Sutskever, O. Vinyals, and Q. V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2014 (pp. 97).

[327] S. Otte et al. “Recurrent Neural Networks for Fast and Robust Vibration- based Ground Classification on Mobile Robots”. In: *IEEE International Conference on Robottics and Automation (ICRA)*. 2016 (pp. 97).

[328] W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki. “Scene Labeling with LSTM Recurrent Neural Networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (pp. 97).

[329] A. Alahi et al. “Social LSTM: Human Trajectory Prediction in Crowded Spaces”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (pp. 97).

[330] Q. Wang, Y. Liu, J. Liu, Y. Gu, and S. Kamijo. “Critical Areas Detection and Vehicle Speed Estimation System Towards Intersection-Related Driving Behavior Analysis”. In: *2018 IEEE International Conference on Consumer Electronics (ICCE)*. 2018 (pp. 97).

[331] N. Kumar, D. Puthal, T. Theocharides, and S. P. Mohanty. “Unmanned Aerial Vehicles in Consumer Applications: New Applications in Current and Future Smart Environments”. In: *IEEE Consumer Electronics Magazine (CEM)* 8.3 (2019) (pp. 97).

[332] D. V. McGehee, E. N. Mazzae, and G. S. Baldwin. “Driver Reaction Time in Crash Avoidance Research: Validation of a Driving Simulator Study on a Test Track”. In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Vol. 44. 20. 2000 (pp. 97).

[333] Y. Guan, Z. Yuan, G. Sun, and J. Cong. “FPGA-based accelerator for long short-term memory recurrent neural networks”. In: *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2017 (pp. 97, 119).

[334] A. X. M. Chang and E. Culurciello. “Hardware Accelerators for Recurrent Neural Networks on FPGA”. In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. 2017 (pp. 97, 100).

[335] X. Zhang, X. Liu, A. Ramachandran, C. Zhuge, S. Tang, P. Ouyang, Z. Cheng, K. Rupnow, and D. Chen. “High-Performance Video Content Recognition with Long-Term Recurrent Convolutional Network for FPGA”. In: *International Conference on Field Programmable Logic and Applications (FPL)*. 2017 (pp. 97, 98, 100).

[336] S. Ray. “Safety, Security, and Reliability: The Automotive Robustness Problem and an Architectural Solution”. In: *2019 IEEE International Conference on Consumer Electronics (ICCE)*. 2019 (pp. 98, 116).

[337] Z. Wang, J. Lin, and Z. Wang. “Accelerating Recurrent Neural Networks: A Memory-Efficient Approach”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.10 (2017) (pp. 98, 100).

[338] V. Rybalkin et al. “FINN-L: Library Extensions and Design Trade-off Analysis for Variable Precision LSTM Networks on FPGAs”. In: *Internaltional Conference on Field Programmable Logic and Applications (FPL)*. 2018 (pp. 98).

[339] S. Wang, S. S. Cheung, and H. Sajid. “Visual Bubble: Protecting Privacy in Wearable Cameras”. In: *IEEE Consumer Electronics Magazine (CEM)* 7.1 (2018) (pp. 98).

[340] M. Zhang, S. Rajbhandari, W. Wang, and Y. He. “DeepCPU: Serving RNN-based Deep Learning Models 10x Faster”. In: *2018 USENIX Annual Technical Conference*. 2018 (pp. 99).

[341] G. Diamos et al. “Persistent RNNs: Stashing Recurrent Weights On-Chip”. In: *International Conference on Machine Learning (ICML)*. 2016 (pp. 99).

[342] F. Zhu, J. Pool, M. Andersch, J. Appleyard, and F. Xie. “Sparse Persistent RNNs: Squeezing Large Recurrent Networks On-Chip”. In: *International Conference on Learning Representations (ICLR)*. 2018 (pp. 99).

[343] X. Zhang, C. Xie, J. Wang, W. Zhang, and X. Fu. “Towards Memory Friendly Long-Short Term Memory Networks (LSTMs) on Mobile GPUs”. In: *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2018 (pp. 99, 100, 102).

[344] J. Fowers et al. “A Configurable Cloud-scale DNN Processor for Real-time AI”. In: *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*. 2018 (pp. 100).

[345] S. Li, C. Wu, H. Li, B. Li, Y. Wang, and Q. Qiu. “FPGA Acceleration of Recurrent Neural Network Based Language Model”. In: *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2015 (pp. 100).

[346] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr. “Accelerating Recurrent Neural Networks in Analytics Servers: Comparison of FPGA, CPU, GPU, and ASIC”. In: *Internaltional Conference on Field Programmable Logic and Applications (FPL)*. 2016 (pp. 100).

[347] J. Chung et al. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In: *NeurIPS Workshop on Deep Learning*. 2014 (pp. 100).

[348] Z. Li et al. “E-RNN: Design Optimization for Efficient Recurrent Neural Networks in FPGAs”. In: *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2019 (pp. 100, 102).

[349] J. Park, W. Yi, D. Ahn, J. Kung, and J. Kim. "Balancing Computation Loads and Optimizing Input Vector Loading in LSTM Accelerators". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2019) (pp. 101).

[350] D. Neil, J. H. Lee, T. Delbruck, and S.-C. Liu. "Delta Networks for Optimized Recurrent Network Computation". In: *International Conference on Machine Learning (ICML)*. 2017 (pp. 102).

[351] M. Denil, B. Shakibi, L. Dinh, and N. De Freitas. "Predicting Parameters in Deep Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2013 (pp. 106).

[352] C. Eckart and G. Young. "The approximation of one matrix by another of lower rank". In: *Psychometrika* 1.3 (1936) (pp. 107).

[353] S. Bouguezzi, H. Faiedh, and C. Souani. "Hardware implementation of tanh exponential activation function using FPGA". In: *International Multi-Conference on Systems, Signals & Devices (SSD)*. IEEE. 2021 (pp. 114).

[354] D. A. Pomerleau. "Alvinn: An Autonomous Land Vehicle in a Neural Network". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 1989 (pp. 116).

[355] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell. "BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2020) (pp. 116, 117).

[356] S. Liu, J. Tang, Z. Zhang, and J.-L. Gaudiot. "Computer Architectures for Autonomous Driving". In: *Computer* 50.8 (2017) (pp. 116).

[357] M. Bojarski et al. "End to End Learning for Self-Driving Cars". In: *arXiv preprint arXiv:1604.07316* (2016) (pp. 117).

[358] V. K. Kukkala, J. Tunnell, S. Pasricha, and T. Bradley. "Advanced Driver-Assistance Systems: A Path Toward Autonomous Vehicles". In: *IEEE Consumer Electronics Magazine (CEM)* 7.5 (2018) (pp. 117).

[359] L. Chi and Y. Mu. "Deep Steering: Learning End-to-End Driving Model from Spatial and Temporal Visual Cues". In: *arXiv:1708.03798* (2017) (pp. 117).

[360] S. Thrun. "Toward Robotic Cars". In: *Communications of the ACM* 53.4 (2010) (pp. 117).

[361] P. Corcoran and S. K. Datta. "Mobile-Edge Computing and the Internet of Things for Consumers: Extending Cloud Computing and Services to the Edge of the Network". In: *IEEE Consumer Electronics Magazine (CEM)* 5.4 (2016) (pp. 117).

[362] H. Xu, Y. Gao, F. Yu, and T. Darrell. "End-To-End Learning of Driving Models From Large-Scale Video Datasets". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 117, 177).

[363] Z. Que, Y. Zhu, H. Fan, J. Meng, X. Niu, and W. Luk. "Mapping large LSTMs to FPGAs with weight reuse". In: *Journal of Signal Processing Systems* 92.9 (2020) (pp. 120).

[364] H. Xu, Y. Gao, F. Yu, and T. Darrell. "End-to-End Learning of Driving Models from Large-Scale Video Datasets". In: *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 131).

[365] X. Zhang, Y. Feng, P. Angeloudis, and Y. Demiris. "Monocular visual traffic surveillance: a review". In: *IEEE Transactions on Intelligent Transportation Systems* (2022) (pp. 131).

[366] C. Kyrkou, S. Timotheou, P. Kolios, T. Theocharides, and C. Panayiotou. "Drones: Augmenting Our Quality of Life". In: *IEEE Potentials* 38.1 (2019) (pp. 131).

[367] J. Lee, J. Wang, D. Crandall, S. Šabanović, and G. Fox. "Real-Time, Cloud-Based Object Detection for Unmanned Aerial Vehicles". In: *IEEE International Conference on Robotic Computing (IRC)*. 2017 (pp. 131).

[368] A. Montanari, F. Kringberg, A. Valentini, C. Mascolo, and A. Prorok. “Surveying Areas in Developing Regions Through Context Aware Drone Mobility”. In: *ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*. 2018 (pp. 132).

[369] K. Kanistras, G. Martins, M. J. Rutherford, and K. P. Valavanis. “Survey of Unmanned Aerial Vehicles (UAVs) for Traffic Monitoring”. In: *Springer Handbook of Unmanned Aerial Vehicles, Ch. 110* (2015) (pp. 132, 137).

[370] J. Huang et al. “Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors”. In: *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 132, 133, 136, 137, 143, 152).

[371] M. J. Shafiee, B. Chywl, F. Li, and A. Wong. “Fast YOLO: A fast you only look once system for real-time embedded object detection in video”. In: *arXiv preprint arXiv:1709.05943* (2017) (pp. 132, 142).

[372] Y. J. Hao, L. K. Teck, C. Y. Xiang, E. Jeevanraj, and S. Srigrarom. “Fast Drone Detection using SSD and YoloV3”. In: *International Conference on Control, Automation and Systems (ICCAS)*. IEEE. 2021 (pp. 132, 142).

[373] A. Suleiman, Y. Chen, J. Emer, and V. Sze. “Towards closing the energy gap between HOG and CNN features for embedded vision”. In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. 2017 (pp. 133, 134, 137).

[374] R. Girshick, J. Donahue, T. Darrell, and J. Malik. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014 (pp. 133, 134).

[375] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. “Selective search for object recognition”. In: *International Journal of Computer Vision (IJCV)* 104.2 (2013) (pp. 135).

[376] R. Girshick. “Fast R-CNN”. In: *IEEE International Conferernce on Computer Vision (ICCV)*. 2015 (pp. 135).

[377] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. “The Pascal Visual Object Classes (VOC) Challenge”. In: *International Journal of Computer Vision (IJCV)* 88.2 () (pp. 136, 139, 153).

[378] T. Ringwald et al. “UAV-Net: A Fast Aerial Vehicle Detector for Mobile Platforms”. In: *CVPR Workshops*. 2019 (pp. 136).

[379] S. Jung, S. Hwang, H. Shin, and D. H. Shim. “Perception, Guidance, and Navigation for Indoor Autonomous Drone Racing Using Deep Learning”. In: *IEEE Robotics and Automation Letters (RA-L)* 3.3 (2018) (pp. 136).

[380] P. Pandey, Q. He, D. Pompili, and R. Tron. “Light-Weight Object Detection and Decision Making via Approximate Computing in Resource-Constrained Mobile Robots”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018 (pp. 137).

[381] F. Ozge Unel, B. O. Ozkalayci, and C. Cigla. “The Power of Tiling for Small Object Detection”. In: *CVPR Workshops*. 2019 (pp. 137).

[382] N. Audebert, B. Le Saux, and S. Lefèvre. “Segment-Before-Detect: Vehicle Detection and Classification Through Semantic Segmentation of Aerial Images”. In: *Remote Sensing* 9.4 (2017) (pp. 137, 154).

[383] G. Xia, X. Bai, J. Ding, Z. Zhu, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang. “DOTA: A Large-Scale Dataset for Object Detection in Aerial Images”. In: *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 137, 153).

[384] G. Plastiras, C. Kyrkou, and T. Theocharides. "Efficient ConvNet-based Object Detection for Unmanned Aerial Vehicles by Selective Tile Processing". In: *ACM International Conference on Distributed Smart Cameras (ICDSC)*. 2018 (pp. 137).

[385] L. Ding, Y. Wang, R. Laganière, X. Luo, and S. Fu. "Scale-Aware RPN for Vehicle Detection". In: *Advances in Visual Computing*. 2018 (pp. 137).

[386] Y. Xu, G. Yu, Y. Wang, X. Wu, and Y. Ma. "Car Detection from Low-Altitude UAV Imagery with the Faster R-CNN". In: *Hindawi Journal of Advanced Transportation* 2017 (2017) (pp. 137, 143).

[387] T. N. Mundhenk, G. Konjevod, W. A. Sakla, and K. Boakye. "A large contextual dataset for classification, detection and counting of cars with deep learning". In: *European Conference on Computer Vision (ECCV)*. 2016 (pp. 138).

[388] S. Razakarivony and F. Jurie. "Vehicle detection in aerial imagery: A small target detection benchmark". In: *Journal of Visual Communication and Image Representation* 34 (2016) (pp. 138).

[389] L. Wen, D. Du, Z. Cai, Z. Lei, M.-C. Chang, H. Qi, J. Lim, M.-H. Yang, and S. Lyu. "UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking". In: *Computer Vision and Image Understanding* 193 (2020) (pp. 138).

[390] C. Papageorgiou and T. Poggio. "A trainable system for object detection". In: *International Journal of Computer Vision (IJCV)* 38.1 (2000) (pp. 138).

[391] M.-R. Hsieh, Y.-L. Lin, and W. H. Hsu. "Drone-based object counting by spatially regularized regional proposal network". In: *IEEE International Conference on Computer Vision (ICCV)*. 2017 (pp. 138).

[392] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese. "Learning social etiquette: Human trajectory understanding in crowded scenes". In: *European Conference on Computer Vision (ECCV)*. Springer. 2016 (pp. 138).

[393] R. Krajewski, J. Bock, L. Kloeker, and L. Eckstein. "The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems". In: *International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018 (pp. 138).

[394] M. Mueller, N. Smith, and B. Ghanem. "A benchmark and simulator for uav tracking". In: *European Conference on Computer Vision (ECCV)*. Springer. 2016 (pp. 138).

[395] I. Bozcan and E. Kayacan. "Au-air: A multi-modal unmanned aerial vehicle dataset for low altitude traffic surveillance". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020 (pp. 138).

[396] D. Du, Y. Qi, H. Yu, Y. Yang, K. Duan, G. Li, W. Zhang, Q. Huang, and Q. Tian. "The unmanned aerial vehicle benchmark: Object detection and tracking". In: *European Conference on Computer Vision (ECCV)*. 2018 (pp. 138).

[397] P. Zhu, L. Wen, X. Bian, H. Ling, and Q. Hu. "Vision Meets Drones: A challenge". In: *arXiv preprint arXiv:1804.07437* (2018) (pp. 138).

[398] P. Petrides, C. Kyrkou, P. Kolios, T. Theocharides, and C. Panayiotou. "Towards a Holistic Performance Evaluation Framework for Drone-based Object Detection". In: *International Conference on Unmanned Aircraft Systems (ICUAS)*. 2017 (pp. 139).

[399] C. Kyrkou, S. Timotheou, P. Kolios, T. Theocharides, and C. G. Panayiotou. "Optimized Vision-Directed Deployment of UAVs for Rapid Traffic Monitoring". In: *IEEE International Conference on Consumer Electronics (ICCE)*. 2018 (pp. 143, 150).

[400] J. Gleason, A. V. Nefian, X. Bouyssounousse, T. Fong, and G. Bebis. "Vehicle detection from aerial imagery". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2011 (pp. 144).

[401] Y. Lin and S. Saripalli. "Road detection from aerial imagery". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2012 (pp. 150).

[402] Y. Lyu, L. Bai, and X. Huang. "Road segmentation using cnn and distributed lstm". In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2019 (pp. 150).

[403] K. Schmid, T. Tomic, F. Ruess, H. Hirschmüller, and M. Suppa. "Stereo vision based indoor/outdoor navigation for flying robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2013 (pp. 165).

[404] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar. "Multi-sensor fusion for robust autonomous flight in indoor and outdoor environments with a rotorcraft MAV". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2014 (pp. 165).

[405] K. Çelik and A. K. Somani. "Monocular vision SLAM for indoor aerial vehicles". In: *Journal of Electrical and Computer Engineering* (2013) (pp. 165).

[406] P. Chakravarty, K. Kelchtermans, T. Roussel, S. Wellens, T. Tuytelaars, and L. V. Eycken. "CNN-based single image obstacle avoidance on a quadrotor". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017 (pp. 165, 169, 170).

[407] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena. "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017 (pp. 166).

[408] N. Imanberdiyev, C. Fu, E. Kayacan, and I. Chen. "Autonomous navigation of UAV by using real-time model-based reinforcement learning". In: *International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2016 (pp. 166).

[409] F. Sadeghi and S. Levine. "Cad2rl: Real single-image flight without a single real image". In: *Robotics Science and Systems Conference (RSS)* (2017) (pp. 166).

[410] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza. "Flightmare: A flexible quadrotor simulator". In: *Conference on Robot Learning (CoRL)*. PMLR. 2021 (pp. 166).

[411] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. "Target-driven visual navigation in indoor scenes using deep reinforcement learning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017 (pp. 166, 197, 253).

[412] W. H. Warren, B. A. Kay, W. D. Zosh, A. P. Duchon, and S. Sahuc. "Optic flow is used to control human walking". In: *Nature neuroscience* 4.2 (2001) (pp. 166).

[413] K. Souhila and A. Karim. "Optical flow based robot obstacle avoidance". In: *International Journal of Advanced Robotic Systems* 4.1 (2007) (pp. 166).

[414] V. Grabe, H. H. Bülthoff, and P. R. Giordano. "On-board velocity estimation and closed-loop control of a quadrotor UAV based on optical flow". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2012 (pp. 166).

[415] K. McGuire, G. De Croon, C. De Wagter, K. Tuyls, and H. Kappen. "Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone". In: *IEEE Robotics and Automation Letters (RA-L)* 2.2 (2017) (pp. 166).

[416] D. K. Kim and T. Chen. "Deep neural network for real-time autonomous indoor navigation". In: *arXiv preprint arXiv:1511.04668* (2015) (pp. 168, 170).

[417] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert. "Learning monocular reactive uav control in cluttered natural environments". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2013 (pp. 168, 170).

[418] A. Giusti et al. “A machine learning approach to visual perception of forest trails for mobile robots”. In: *IEEE Robotics and Automation Letters (RA-L)* 1.2 (2016) (pp. 168).

[419] S. Yang, S. Konam, C. Ma, S. Rosenthal, M. Veloso, and S. Scherer. “Obstacle Avoidance through Deep Networks based Intermediate Perception”. In: *arXiv preprint arXiv:1704.08759* (2017) (pp. 168).

[420] L. Pinto and A. Gupta. “Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016 (pp. 169).

[421] S. Zhou and K. Iagnemma. “Self-supervised learning method for unstructured road detection using Fuzzy Support Vector Machines”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2010 (pp. 169).

[422] S. Pillai and J. J. Leonard. “Towards visual ego-motion learning in robots”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 (pp. 169).

[423] D. Gandhi, L. Pinto, and A. Gupta. “Learning to Fly by Crashing”. In: (2017) (pp. 169, 171, 174, 177, 184, 186, 187, 191, 192).

[424] U. Shah, R. Khawad, and K. M. Krishna. “DeepFly: towards complete autonomous navigation of MAVs with monocular camera”. In: *Indian Conference on Computer Vision, Graphics and Image Processing*. ACM. 2016 (pp. 170).

[425] A. Abobakr, M. Hossny, and S. Nahavandi. “Body joints regression using deep convolutional neural networks”. In: *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2016 (pp. 171).

[426] T. Naseer and W. Burgard. “Deep regression for monocular camera-based 6-DoF global localization in outdoor environments”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 (pp. 171).

[427] N. Patel, A. Choromanska, P. Krishnamurthy, and F. Khorrami. “Sensor modality fusion with CNNs for UGV autonomous driving in indoor environments”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 (pp. 171).

[428] H. Chen, Y. F. Li, and D. Su. “M3Net: Multi-scale multi-path multi-modal fusion network and example application to RGB-D salient object detection”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 (pp. 171).

[429] R. Zhao, H. Ali, and P. van der Smagt. “Two-stream RNN/CNN for action recognition in 3D videos”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 (pp. 171).

[430] J. Delmerico and D. Scaramuzza. “A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018 (pp. 174).

[431] J. Carreira and A. Zisserman. “Quo vadis, action recognition? a new model and the kinetics dataset”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 177).

[432] G. Koch, R. Zemel, R. Salakhutdinov, et al. “Siamese neural networks for one-shot image recognition”. In: *ICML Deep Learning workshop*. Vol. 2. Lille. 2015 (pp. 177).

[433] G. Farnebäck. “Two-frame motion estimation based on polynomial expansion”. In: *Scandinavian conference on Image analysis*. Springer. 2003 (pp. 177, 190).

[434] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. “High accuracy optical flow estimation based on a theory for warping”. In: *European Conference on Computer Vision (ECCV)*. Springer. 2004 (pp. 177).

[435] K. Simonyan and A. Zisserman. “Two-stream convolutional networks for action recognition in videos”. In: *Advances in Neural Information Processing Systems (NeurIPS)* 27 (2014) (pp. 177, 181, 190).

[436] N. J. Sanket, C. D. Singh, K. Ganguly, C. Fermüller, and Y. Aloimonos. “Gapflyt: Active vision based minimalist structure-less gap detection for quadrotor flight”. In: *IEEE Robotics and Automation Letters (RA-L)* 3.4 (2018) (pp. 177, 185).

[437] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. “Flownet 2.0: Evolution of optical flow estimation with deep networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 177, 190).

[438] Y. Zhu, Z. Lan, S. Newsam, and A. G. Hauptmann. “Guided optical flow learning”. In: *arXiv preprint arXiv:1702.02295* (2017) (pp. 178).

[439] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. “Greedy layer-wise training of deep networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)* 19 (2006) (pp. 180).

[440] I. Misra, C. L. Zitnick, and M. Hebert. “Shuffle and learn: unsupervised learning using temporal order verification”. In: *European Conference on Computer Vision (ECCV)*. Springer. 2016 (pp. 180).

[441] B. D. Lucas, T. Kanade, et al. *An iterative image registration technique with an application to stereo vision*. Vol. 81. Vancouver, 1981 (pp. 190).

[442] C. Zach, T. Pock, and H. Bischof. “A duality based approach for realtime tv-l 1 optical flow”. In: *Joint pattern recognition symposium*. Springer. 2007 (pp. 190).

[443] F. Bonin-Font, A. Ortiz, and G. Oliver. “Visual navigation for mobile robots: A survey”. In: *Journal of Intelligent and Robotic Systems* 53 (2008) (pp. 197).

[444] W. Agnew, C. Xie, A. Walsman, O. Murad, Y. Wang, P. Domingos, and S. Srinivasa. “Amodal 3d reconstruction for robotic manipulation via stability and connectivity”. In: *Conference on Robot Learning (CoRL)*. PMLR. 2021 (pp. 197).

[445] M. Zollhöfer, P. Stotko, A. Görlitz, C. Theobalt, M. Nießner, R. Klein, and A. Kolb. “State of the art on 3D reconstruction with RGB-D cameras”. In: *Computer Graphics Forum*. Vol. 37. 2. Wiley Online Library. 2018 (pp. 197).

[446] D. Shin, C. C. Fowlkes, and D. Hoiem. “Pixels, voxels, and views: A study of shape representations for single view 3d object shape prediction”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 197, 198, 204, 207, 213).

[447] A. Manni, D. Oriti, A. Sanna, F. De Pace, and F. Manuri. “Snap2cad: 3D indoor environment reconstruction for AR/VR applications using a smartphone device”. In: *Computers & Graphics* 100 (2021) (pp. 197).

[448] L. Jinyu, Y. Bangbang, C. Danpeng, W. Nan, Z. Guofeng, and B. Hujun. “Survey and evaluation of monocular visual-inertial SLAM algorithms for augmented reality”. In: *Virtual Reality & Intelligent Hardware* 1.4 (2019) (pp. 197).

[449] R. C. Smith and P. Cheeseman. “On the representation and estimation of spatial uncertainty”. In: *International Journal of Robotics Research (IJRR)* 5.4 (1986) (pp. 197, 202).

[450] H. Durrant-Whyte and T. Bailey. “Simultaneous localization and mapping: part I”. In: *IEEE Robotics & Automation Magazine (RA-M)* 13.2 (2006) (pp. 197).

[451] B. Huang, J. Zhao, and J. Liu. “A survey of simultaneous localization and mapping with an envision in 6g wireless networks”. In: *arXiv preprint arXiv:1909.05214* (2019) (pp. 197).

[452] T. Taketomi, H. Uchiyama, and S. Ikeda. “Visual SLAM algorithms: A survey from 2010 to 2016”. In: *IPSJ Transactions on Computer Vision and Applications* 9.1 (2017) (pp. 197).

[453] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. "MonoSLAM: Real-time single camera SLAM". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 29.6 (2007) (pp. 197, 203).

[454] R. A. Newcombe et al. "Kinectfusion: Real-time dense surface mapping and tracking". In: *IEEE International Symposium on Mixed and Augmented Reality*. IEEE. 2011 (pp. 197, 203, 213, 229, 231).

[455] R. Li, S. Wang, and D. Gu. "Ongoing evolution of visual SLAM from geometry to deep learning: Challenges and opportunities". In: *Cognitive Computation* 10 (2018) (pp. 197).

[456] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. "Semantic scene completion from a single depth image". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 198, 205).

[457] J. McCormac, R. Clark, M. Bloesch, A. Davison, and S. Leutenegger. "Fusion++: Volumetric object-level slam". In: *International Conference on 3D vision (3DV)*. IEEE. 2018 (pp. 198, 203, 213, 225, 231, 235, 236, 241, 242).

[458] A. Rosinol, M. Abate, Y. Chang, and L. Carlone. "Kimera: an open-source library for real-time metric-semantic localization and mapping". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020 (pp. 198).

[459] R. A. Newcombe, D. Fox, and S. M. Seitz. "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (pp. 198, 203).

[460] A. J. Davison. "FutureMapping: The computational structure of spatial AI systems". In: *arXiv preprint arXiv:1803.11288* (2018) (pp. 198).

[461] Z. Sui, H. Chang, N. Xu, and O. C. Jenkins. "Geofusion: Geometric consistency informed scene estimation in dense clutter". In: *IEEE Robotics and Automation Letters (RA-L)* 5.4 (2020) (pp. 198, 205).

[462] A. X. Chang et al. "Shapenet: An information-rich 3d model repository". In: *arXiv preprint arXiv:1512.03012* (2015) (pp. 198, 204, 222, 253).

[463] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. "Scannet: Richly-annotated 3d reconstructions of indoor scenes". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 198).

[464] J. McCormac, A. Handa, S. Leutenegger, and A. J. Davison. "Scenenet rgb-d: Can 5m synthetic images beat generic imagenet pre-training on indoor segmentation?" In: *IEEE International Conference on Computer Vision (ICCV)*. 2017 (pp. 198).

[465] G. Gkioxari, J. Malik, and J. Johnson. "Mesh r-cnn". In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019 (pp. 198).

[466] S. Tulsiani, R. Tucker, and N. Snavely. "Layer-structured 3d scene inference via view synthesis". In: *European Conference on Computer Vision (ECCV)*. 2018 (pp. 198, 208, 213).

[467] H. Fan, H. Su, and L. J. Guibas. "A point set generation network for 3d object reconstruction from a single image". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 198, 204).

[468] R. Girdhar, D. F. Fouhey, M. Rodriguez, and A. Gupta. "Learning a predictable and generative vector representation for objects". In: *European Conference on Computer Vision (ECCV)*. Springer. 2016 (pp. 198, 204).

[469] A. Dai, C. Ruizhongtai Qi, and M. Nießner. "Shape completion using 3d-encoder-predictor cnns and shape synthesis". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 198, 205).

[470] J. Varley, C. DeChant, A. Richardson, J. Ruales, and P. Allen. “Shape completion enabled robotic grasping”. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017 (pp. 198, 205).

[471] S. Ji, W. Xu, M. Yang, and K. Yu. “3D convolutional neural networks for human action recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 35.1 (2012) (pp. 198, 205).

[472] D. Maturana and S. Scherer. “Voxnet: A 3d convolutional neural network for real-time object recognition”. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2015 (pp. 199, 205).

[473] A. Dai, D. Ritchie, M. Bokeloh, S. Reed, J. Sturm, and M. Nießner. “Scancomplete: Large-scale scene completion and semantic segmentation for 3d scans”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 199, 205).

[474] W. Chen, H. Liang, Z. Chen, F. Sun, and J. Zhang. “Transsc: Transformer-based shape completion for grasp evaluation”. In: *arXiv preprint arXiv:2107.00511* (2021) (pp. 199).

[475] N. Muller, Y.-S. Wong, N. J. Mitra, A. Dai, and M. Nießner. “Seeing behind objects for 3D multi-object tracking in RGB-D sequences”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021 (pp. 199).

[476] K. Li and J. Malik. “Amodal instance segmentation”. In: *European Conference on Computer Vision (ECCV)*. Springer. 2016 (pp. 199, 209).

[477] Z. Deng and L. Jan Latecki. “Amodal detection of 3d objects: Inferring 3d bounding boxes from 2d ones in rgb-depth images”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 199).

[478] X. Zhan, X. Pan, B. Dai, Z. Liu, D. Lin, and C. C. Loy. “Self-supervised scene de-occlusion”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (pp. 199, 206).

[479] J. Shade, S. Gortler, L.-w. He, and R. Szeliski. “Layered depth images”. In: *Annual Conference on Computer Graphics and Interactive Techniques*. 1998 (pp. 199, 207).

[480] Z. Cao, Q. Huang, and R. Karthik. “3d object classification via spherical projections”. In: *International Conference on 3D Vision (3DV)*. IEEE. 2017 (pp. 199).

[481] H. Dhamo, K. Tateno, I. Laina, N. Navab, and F. Tombari. “Peeking behind objects: Layered depth prediction from a single image”. In: *Pattern Recognition Letters* 125 (2019) (pp. 199, 208).

[482] A. Nicastro, R. Clark, and S. Leutenegger. “X-section: Cross-section prediction for enhanced RGB-D fusion”. In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019 (pp. 199, 208, 210, 217, 231, 232, 234–236, 242).

[483] D. Shin, Z. Ren, E. B. Sudderth, and C. C. Fowlkes. “3d scene reconstruction with multi-layer depth and epipolar transformers”. In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019 (pp. 199, 204, 207, 213).

[484] Y. Yao, N. Schertler, E. Rosales, H. Rhodin, L. Sigal, and A. Sheffer. “Front2back: Single view 3d shape reconstruction via front to back prediction”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (pp. 199).

[485] M. Sundermeyer, Z.-C. Marton, M. Durner, M. Brucker, and R. Triebel. “Implicit 3d orientation learning for 6d object detection from rgb images”. In: *European Conference on Computer Vision (ECCV)*. 2018 (pp. 199).

[486] Y. Movshovitz-Attias, T. Kanade, and Y. Sheikh. “How useful is photo-realistic rendering for visual learning?” In: *ECCV Workshops*. Springer. 2016 (pp. 199).

[487] L. G. Roberts. "Machine perception of three-dimensional solids". PhD thesis. Massachusetts Institute of Technology, 1963 (pp. 202).

[488] R. A. Hamzah and H. Ibrahim. "Literature survey on stereo vision disparity map algorithms". In: *Journal of Sensors* 2016 (2016) (pp. 202).

[489] S. Ullman. "The interpretation of structure from motion". In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 203.1153 (1979) (pp. 202).

[490] O. Özyeşil, V. Voroninski, R. Basri, and A. Singer. "A survey of structure from motion*." In: *Acta Numerica* 26 (2017) (pp. 202).

[491] D. Nistér, O. Naroditsky, and J. Bergen. "Visual odometry". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1. Ieee. 2004 (pp. 202).

[492] D. Scaramuzza and F. Fraundorfer. "Visual odometry [tutorial]". In: *IEEE Robotics & Automation Magazine (RA-M)* 18.4 (2011) (pp. 202).

[493] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart. "Robust visual inertial odometry using a direct EKF-based approach". In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2015 (pp. 202).

[494] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. "On-manifold preintegration for real-time visual–inertial odometry". In: *IEEE Transactions on Robotics (T-RO)* 33.1 (2016) (pp. 202).

[495] A. Howard. "Real-time stereo visual odometry for autonomous ground vehicles". In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2008 (pp. 202).

[496] M. Pollefeys, R. Koch, and L. V. Gool. "Self-calibration and metric reconstruction inspite of varying and unknown intrinsic camera parameters". In: *International Journal of Computer Vision (IJCV)* 32.1 (1999) (pp. 202).

[497] S. J. Prince. *Computer vision: models, learning, and inference*. Cambridge University Press, 2012 (pp. 202).

[498] X. Gao and T. Zhang. *Introduction to visual SLAM: from theory to practice*. Springer Nature, 2021 (pp. 202).

[499] A. Kelly. *Mobile robotics: mathematics, models, and methods*. Cambridge University Press, 2013 (pp. 202).

[500] P. Newman, J. Leonard, J. D. Tardós, and J. Neira. "Explore and return: Experimental validation of real-time concurrent mapping and localization". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 2. IEEE. 2002 (pp. 202).

[501] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt. "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration". In: *ACM Transactions on Graphics (ToG)* 36.4 (2017) (pp. 202, 203).

[502] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age". In: *IEEE Transactions on Robotics (T-RO)* 32.6 (2016) (pp. 203).

[503] J. Engel, T. Schöps, and D. Cremers. "LSD-SLAM: Large-scale direct monocular SLAM". In: *European Conference on Computer Vision (ECCV)*. Springer. 2014 (pp. 203).

[504] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system". In: *IEEE Transactions on Robotics (T-RO)* 31.5 (2015) (pp. 203).

[505] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. "Kintinuous: Spatially extended kinectfusion". In: (2012) (pp. 203).

[506] R. Mur-Artal and J. D. Tardós. "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras". In: *IEEE Transactions on Robotics (T-RO)* 33.5 (2017) (pp. 203).

[507] G. Klein and D. Murray. In: *IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE. 2007 (pp. 203).

[508] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. "DTAM: Dense tracking and mapping in real-time". In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE. 2011 (pp. 203).

[509] T. Whelan, S. Leutenegger, R. Salas-Moreno, B. Glocker, and A. Davison. "ElasticFusion: Dense SLAM without a pose graph". In: Robotics Science and Systems Conference (RSS). 2015 (pp. 203).

[510] N. Sünderhauf, T. T. Pham, Y. Latif, M. Milford, and I. Reid. "Meaningful maps with object-oriented semantic mapping". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017 (pp. 203).

[511] A. Rosinol, A. Gupta, M. Abate, J. Shi, and L. Carlone. "3D dynamic scene graphs: Actionable spatial perception with places, objects, and humans". In: *arXiv preprint arXiv:2002.06289* (2020) (pp. 203).

[512] G. Narita, T. Seno, T. Ishikawa, and Y. Kaji. "Panopticfusion: Online volumetric semantic mapping at the level of stuff and things". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019 (pp. 203).

[513] I. A. Kazerouni, L. Fitzgerald, G. Dooly, and D. Toal. "A Survey of State-of-the-Art on Visual SLAM". In: *Expert Systems with Applications* (2022) (pp. 203).

[514] R. Scona, M. Jaimez, Y. R. Petillot, M. Fallon, and D. Cremers. "Staticfusion: Background reconstruction for dense rgb-d slam in dynamic environments". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018 (pp. 203).

[515] B. Bescos, J. M. Fácil, J. Civera, and J. Neira. "DynaSLAM: Tracking, mapping, and inpainting in dynamic scenes". In: *IEEE Robotics and Automation Letters (RA-L)* 3.4 (2018) (pp. 203).

[516] M. Rünz and L. Agapito. "Co-fusion: Real-time segmentation, tracking and fusion of multiple objects". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017 (pp. 203).

[517] M. Runz, M. Buffier, and L. Agapito. "Maskfusion: Real-time recognition, tracking and reconstruction of multiple moving objects". In: *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE. 2018 (pp. 203).

[518] B. Xu, W. Li, D. Tzoumanikas, M. Bloesch, A. Davison, and S. Leutenegger. "Mid-fusion: Octree-based object-level multi-instance dynamic slam". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2019 (pp. 203).

[519] N. Hughes, Y. Chang, and L. Carlone. "Hydra: A real-time spatial perception engine for 3d scene graph construction and optimization". In: *arXiv preprint arXiv:2201.13360* (2022) (pp. 203).

[520] M. Strecke and J. Stuckler. "Em-fusion: Dynamic object-level slam with probabilistic data association". In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019 (pp. 203).

[521] A. Criminisi. "Single-view metrology: Algorithms and applications". In: *Pattern Recognition*. Springer. 2002 (pp. 203).

[522] M. Prasad and A. Fitzgibbon. "Single view reconstruction of curved surfaces". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2. IEEE. 2006 (pp. 204).

[523] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang. "Pixel2mesh: Generating 3d mesh models from single rgb images". In: *European conference on computer vision (ECCV)*. 2018 (pp. 204).

[524] H. Kim, J. Moon, and B. Lee. "RGB-to-TSDF: Direct TSDF prediction from a single RGB image for dense 3D reconstruction". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019 (pp. 204).

[525] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee. "Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision". In: *Advances in Neural Information Processing Systems (NeurIPS)* 29 (2016) (pp. 204).

[526] A. Kundu, Y. Li, and J. M. Rehg. "3d-rcnn: Instance-level 3d object reconstruction via render-and-compare". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 204).

[527] A. Sharma, O. Grau, and M. Fritz. "Vconv-dae: Deep volumetric shape learning without object labels". In: *European Conference on Computer Vision (ECCV)*. Springer. 2016 (pp. 204).

[528] J. Gwak, C. B. Choy, M. Chandraker, A. Garg, and S. Savarese. "Weakly supervised 3d reconstruction with adversarial constraint". In: *International Conference on 3D Vision (3DV)*. IEEE. 2017 (pp. 204).

[529] M. Gadelha, S. Maji, and R. Wang. "3d shape induction from 2d views of multiple objects". In: *International Conference on 3D Vision (3DV)*. IEEE. 2017 (pp. 204).

[530] O. Wiles and A. Zisserman. "Silnet: Single-and multi-view reconstruction by learning from silhouettes". In: *arXiv preprint arXiv:1711.07888* (2017) (pp. 204).

[531] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. "3d-r2n2: A unified approach for single and multi-view 3d object reconstruction". In: *European Conference on Computer Vision (ECCV)*. Springer. 2016 (pp. 204).

[532] S. A. Eslami et al. "Neural scene representation and rendering". In: *Science* 360.6394 (2018) (pp. 204, 213).

[533] K. Piaskowski, R. Staszak, and D. Belter. "Generate What You Can't See-a View-dependent Image Generation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019 (pp. 204, 205, 213).

[534] S. Sajjan, M. Moore, M. Pan, G. Nagaraja, J. Lee, A. Zeng, and S. Song. "Clear grasp: 3d shape estimation of transparent objects for manipulation". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020 (pp. 205).

[535] A. Kar, S. Tulsiani, J. Carreira, and J. Malik. "Category-specific object reconstruction from a single image". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (pp. 205).

[536] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison. "Slam++: Simultaneous localisation and mapping at the level of objects". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013 (pp. 205).

[537] S. Gupta, P. Arbeláez, R. Girshick, and J. Malik. "Aligning 3D models to RGB-D images of cluttered scenes". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (pp. 205).

[538] W. Kuo, A. Angelova, T.-Y. Lin, and A. Dai. "Mask2CAD: 3D shape prediction by learning to segment and retrieve". In: *European Conference on Computer Vision (ECCV)*. Springer. 2020 (pp. 205).

[539] M. Firman, O. Mac Aodha, S. Julier, and G. J. Brostow. "Structured prediction of unobserved voxels from a single depth image". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (pp. 205).

[540] G. Riegler, A. O. Ulusoy, H. Bischof, and A. Geiger. "Octnetfusion: Learning depth fusion from data". In: *International Conference on 3D Vision (3DV)*. IEEE. 2017 (pp. 205, 206).

[541] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. "3d shapenets: A deep representation for volumetric shapes". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (pp. 205).

[542] E. Vespa, N. Nikolov, M. Grimm, L. Nardi, P. H. Kelly, and S. Leutenegger. "Efficient octree-based volumetric SLAM supporting signed-distance and occupancy mapping". In: *IEEE Robotics and Automation Letters (RA-L)* 3.2 (2018) (pp. 206).

[543] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. "A papier-mâché approach to learning 3d surface generation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 206).

[544] G. Riegler, A. Osman Ulusoy, and A. Geiger. "Octnet: Learning deep 3d representations at high resolutions". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 206).

[545] P. Purkait, C. Zach, and I. Reid. "Seeing behind things: Extending semantic segmentation to occluded regions". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019 (pp. 206).

[546] K. Ehsani, R. Mottaghi, and A. Farhadi. "Segan: Segmenting and generating the invisible". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 206, 209).

[547] J. Wu, Y. Wang, T. Xue, X. Sun, B. Freeman, and J. Tenenbaum. "Marrnet: 3d shape reconstruction via 2.5 d sketches". In: *Advances in Neural Information Processing Systems (NeurIPS)* 30 (2017) (pp. 207).

[548] X. Zhang, Z. Zhang, C. Zhang, J. Tenenbaum, B. Freeman, and J. Wu. "Learning to reconstruct shapes from unseen classes". In: *Advances in Neural Information Processing Systems (NeurIPS)* 31 (2018) (pp. 207).

[549] M. Kazhdan, B. Chazelle, D. Dobkin, T. Funkhouser, and S. Rusinkiewicz. "A reflective symmetry descriptor for 3D models". In: *Algorithmica* 38 (2004) (pp. 207).

[550] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield. "Deep object pose estimation for semantic robotic grasping of household objects". In: *Conference on Robot Learning (CoRL)* (2018) (pp. 208, 258).

[551] L. Qi, L. Jiang, S. Liu, X. Shen, and J. Jia. "Amodal instance segmentation with kins dataset". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (pp. 209).

[552] S. Tulsiani, S. Gupta, D. F. Fouhey, A. A. Efros, and J. Malik. "Factoring shape, pose, and layout from the 2d image of a 3d scene". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 211).

[553] D. Eigen, C. Puhrsch, and R. Fergus. "Depth map prediction from a single image using a multi-scale deep network". In: *Advances in Neural Information Processing Systems (NeurIPS)* 27 (2014) (pp. 213).

[554] M. Tatarchenko, A. Dosovitskiy, and T. Brox. "Multi-view 3d models from single images with a convolutional network". In: *European Conference on Computer Vision (ECCV)*. Springer. 2016 (pp. 213).

[555] S. R. Richter and S. Roth. "Matryoshka networks: Predicting 3d geometry via nested shape layers". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 213).

[556] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. "Generative adversarial text to image synthesis". In: *International Conference on Machine Learning (ICML)*. PMLR. 2016 (pp. 213).

[557] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. "Indoor segmentation and support inference from rgbd images." In: *European Conference on Computer Vision (ECCV)* 7576 (2012) (pp. 222).

[558] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. "Sparse convolutional neural networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (pp. 222).

[559] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. "Benchmarking in manipulation research: The ycb object and model set and benchmarking protocols". In: *arXiv preprint arXiv:1502.03143* (2015) (pp. 222, 224).

[560] P. J. Besl and N. D. McKay. "Method for registration of 3-D shapes". In: *Sensor fusion IV: control paradigms and data structures*. Vol. 1611. Spie. 1992 (pp. 226).

[561] B. Curless and M. Levoy. "A volumetric method for building complex models from range images". In: *Annual Conference on Computer Graphics and Interactive Techniques*. 1996 (pp. 228).

[562] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart. "Signed distance fields: A natural representation for both mapping and planning". In: *RSS Workshop: Geometry and Beyond-Representations, Physics, and Scene Understanding for Robotics*. University of Michigan. 2016 (pp. 228).

[563] D. Antensteiner, S. Štolc, and T. Pock. "A review of depth and normal fusion algorithms". In: *Sensors* 18.2 (2018) (pp. 228).

[564] S. D. Roth. "Ray casting for modeling solids". In: *Computer Graphics and Image Processing* 18.2 (1982) (pp. 229).

[565] Z. Zhang. "A flexible new technique for camera calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 22.11 (2000) (pp. 229).

[566] Y. Wu et al. *Tensorpack*. https://github.com/tensorpack/. 2016 (pp. 231).

[567] Q.-Y. Zhou, J. Park, and V. Koltun. "Open3D: A modern library for 3D data processing". In: *arXiv preprint arXiv:1801.09847* (2018) (pp. 231).

[568] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih. "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation". In: *Parallel Computing* 38.3 (2012) (pp. 231).

[569] Y. Wu, L. Liu, J. Bae, K.-H. Chow, A. Iyengar, C. Pu, W. Wei, L. Yu, and Q. Zhang. "Demystifying learning rate policies for high accuracy training of deep neural networks". In: *IEEE International Conference on Big Data*. IEEE. 2019 (pp. 231).

[570] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. "Deeper depth prediction with fully convolutional residual networks". In: *International Conference on 3D vision (3DV)*. IEEE. 2016 (pp. 236).

[571] A. Bicchi and V. Kumar. "Robotic grasping and contact: A review". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 1. IEEE. 2000 (pp. 243).

[572] H.-S. Fang, C. Wang, M. Gou, and C. Lu. "Graspnet-1billion: A large-scale benchmark for general object grasping". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (pp. 243, 253).

[573] A. Mousavian, C. Eppner, and D. Fox. "6-dof graspnet: Variational grasp generation for object manipulation". In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019 (pp. 243).

[574] S. Chitta. "MoveIt!: an introduction". In: *Robot Operating System (ROS) The Complete Reference (Volume 1)* (2016) (pp. 243).

[575] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox. "Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021 (pp. 244).

[576] C. Kerry et al. "Protecting privacy in an AI-driven world". In: *https://www. brookings. edu/research/protecting-privacy-in-an-ai-driven-world/* (2020) (pp. 248).

[577] A. R. Zamir, T.-L. Wu, L. Sun, W. B. Shen, B. E. Shi, J. Malik, and S. Savarese. "Feedback networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 249).

[578] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos. "Stripes: Bit-serial deep neural network computing". In: *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. 2016 (pp. 249).

[579] J. Gawlikowski et al. "A survey of uncertainty in deep neural networks". In: *arXiv preprint arXiv:2107.03342* (2021) (pp. 249).

[580] S. I. Venieris, C.-S. Bouganis, and N. D. Lane. "Multi-DNN Accelerators for Next-Generation AI Systems". In: *arXiv preprint arXiv:2205.09376* (2022) (pp. 250).

[581] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh. "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network". In: *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. IEEE. 2018 (pp. 250).

[582] J. Frankle and M. Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks". In: *arXiv preprint arXiv:1803.03635* (2018) (pp. 250).

[583] X. Ma et al. "Non-structured DNN weight pruning—Is it beneficial in any platform?" In: *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)* 33.9 (2021) (pp. 251).

[584] J. Hu, G. Zhao, S. You, and C. J. Kuo. "Evaluation of multimodal semantic segmentation using RGB-D data". In: *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*. Vol. 11746. SPIE. 2021 (pp. 251).

[585] M. Teutsch and W. Krüger. "Detection, segmentation, and tracking of moving objects in UAV videos". In: *IEEE Ninth International Conference on Advanced Video and Signal-Based Surveillance*. IEEE. 2012 (pp. 251).

[586] X. Li, Z. Liu, P. Luo, C. Change Loy, and X. Tang. "Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 251).

[587] S. Wang, J. Cao, and P. Yu. "Deep learning for spatio-temporal data mining: A survey". In: *IEEE Transactions on Knowledge and Data Engineering* (2020) (pp. 252).

[588] D. Sahoo, Q. Pham, J. Lu, and S. C. Hoi. "Online deep learning: Learning deep neural networks on the fly". In: *arXiv preprint arXiv:1711.03705* (2017) (pp. 252).

[589] G. Csurka. "Domain adaptation for visual applications: A comprehensive survey". In: *arXiv preprint arXiv:1702.05374* (2017) (pp. 252).

[590] J. Schneider and M. Vlachos. "Personalization of deep learning". In: *International Data Science Conference (IDSC)*. Springer. 2021 (pp. 252).

[591] I. Leontiadis, S. Laskaridis, S. I. Venieris, and N. D. Lane. "It's always personal: Using early exits for efficient on-device CNN personalisation". In: *International Workshop on Mobile Computing Systems and Applications*. 2021 (pp. 252).

[592] S. Chen, Y. Li, and N. M. Kwok. "Active vision in robotic systems: A survey of recent developments". In: *International Journal of Robotics Research (IJRR)* 30.11 (2011) (pp. 253).

[593] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart. "Receding horizon" next-best-view" planner for 3d exploration". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016 (pp. 253).

[594] A. Doumanoglou, R. Kouskouridas, S. Malassiotis, and T.-K. Kim. "Recovering 6D object pose and predicting next-best-view in the crowd". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (pp. 253).

[595] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011 (pp. 253).

[596] S. Tulsiani, A. A. Efros, and J. Malik. "Multi-view consistency as supervisory signal for learning shape and pose prediction". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (pp. 253).

[597] D. Schiebener, A. Schmidt, N. Vahrenkamp, and T. Asfour. "Heuristic 3D object shape completion based on symmetry and scene context". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016 (pp. 253).

[598] M. Strecke and J. Stuckler. "Where does it end?-reasoning about hidden surfaces by object intersection constraints". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (pp. 253).

[599] J. Zhang, M. Henein, R. Mahony, and V. Ila. "VDO-SLAM: a visual dynamic object-aware SLAM system". In: *arXiv preprint arXiv:2005.11052* (2020) (pp. 253).

[600] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov. "Spatially adaptive computation time for residual networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (pp. 254).

[601] S. Cai, Y. Shu, and W. Wang. "Dynamic routing networks". In: *IEEE/CVF Winter Conference on Applications of Computer Vision*. 2021 (pp. 254).

[602] Y. Gal and Z. Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning". In: *International Conference on Machine Learning (ICML)*. PMLR. 2016 (pp. 254).

[603] A. Kendall and Y. Gal. "What uncertainties do we need in bayesian deep learning for computer vision?" In: *Advances in Neural Information Processing Systems (NeurIPS)* 30 (2017) (pp. 254).

[604] B. Lakshminarayanan, A. Pritzel, and C. Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles". In: *Advances in Neural Information Processing Systems (NeurIPS)* 30 (2017) (pp. 254).

[605] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. "On calibration of modern neural networks". In: *International Conference on Machine Learning (ICML)*. PMLR. 2017 (pp. 254).

[606] J. Yang, K. Zhou, Y. Li, and Z. Liu. "Generalized out-of-distribution detection: A survey". In: *arXiv preprint arXiv:2110.11334* (2021) (pp. 255).

[607] D. Hendrycks and K. Gimpel. "A baseline for detecting misclassified and out-of-distribution examples in neural networks". In: *arXiv preprint arXiv:1610.02136* (2016) (pp. 255).

[608] D. Ulmer. "A survey on evidential deep learning for single-pass uncertainty estimation". In: *arXiv preprint arXiv:2110.03051* (2021) (pp. 255).

[609] M. Sensoy, L. Kaplan, and M. Kandemir. "Evidential deep learning to quantify classification uncertainty". In: *Advances in Neural Information Processing Systems (NeurIPS)* 31 (2018) (pp. 255).

[610] S. Thrun. "Probabilistic robotics". In: *Communications of the ACM* 45.3 (2002) (pp. 255).

[611] A. Bendale and T. E. Boult. “Towards open set deep networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (pp. 255).

[612] A. Bendale and T. Boult. “Towards open world recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (pp. 255).

[613] K. D. Nguyen and C. Ha. “Development of hardware-in-the-loop simulation based on gazebo and pixhawk for unmanned aerial vehicles”. In: *International Journal of Aeronautical and Space Sciences* 19 (2018) (pp. 255).

[614] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr. “WRPN: Wide reduced-precision networks”. In: *arXiv preprint arXiv:1709.01134* (2017) (pp. 257).

[615] A. Pashevich, R. Strudel, I. Kalevatykh, I. Laptev, and C. Schmid. “Learning to augment synthetic images for sim2real policy transfer”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019 (pp. 258).

[616] K. Arndt, M. Hazara, A. Ghadirzadeh, and V. Kyrki. “Meta reinforcement learning for sim-to-real domain adaptation”. In: *IEEE international conference on robotics and automation (ICRA)*. IEEE. 2020 (pp. 258).

[617] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. “Attention is all you need”. In: *Advances in Neural Information Processing Systems (NeurIPS)* 30 (2017) (pp. 259).

[618] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. “Swin transformer: Hierarchical vision transformer using shifted windows”. In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021 (pp. 259).

[619] P. Dhariwal and A. Nichol. “Diffusion models beat gans on image synthesis”. In: *Advances in Neural Information Processing Systems (NeurIPS)* 34 (2021) (pp. 259).

[620] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. “Nerf: Representing scenes as neural radiance fields for view synthesis”. In: *Communications of the ACM* 65.1 (2021) (pp. 259).

[621] H. Parry, L. Xun, A. Sabet, J. Bi, J. Hare, and G. V. Merrett. “Dynamic transformer for efficient machine translation on embedded devices”. In: *ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)*. IEEE. 2021 (pp. 259).

[622] M. Elbayad, J. Gu, E. Grave, and M. Auli. “Depth-adaptive transformer”. In: *arXiv preprint arXiv:1910.10073* (2019) (pp. 259).

[623] Y. Benny and L. Wolf. “Dynamic Dual-Output Diffusion Models”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022 (pp. 259).

[624] Z. Lyu, X. Xu, C. Yang, D. Lin, and B. Dai. “Accelerating diffusion models via early stop of the diffusion process”. In: *arXiv preprint arXiv:2205.12524* (2022) (pp. 259).

[625] J. Fang, T. Yi, X. Wang, L. Xie, X. Zhang, W. Liu, M. Nießner, and Q. Tian. “Fast dynamic radiance fields with time-aware neural voxels”. In: *SIGGRAPH Asia Conference*. 2022 (pp. 259).