Imperial College London
Department of Electrical and Electronic Engineering

Data-Driven Spatial Sparsity for Convolutional Neural Networks

Georgios Zampokas

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of the University of London and
the Diploma of Imperial College, May 2024

Abstract

Advancements in Convolutional Neural Networks (CNNs) have driven significant research efforts towards optimizing their effectiveness and applicability. A promising yet less explored path lies in leveraging spatially adaptive computation within networks—an approach aimed at reducing computation redundancy and bolstering overall efficiency. This work investigates the presence of computational redundancy within CNN computations in vision tasks, and explores dynamic optimization methodologies to increase model efficiency.

Firstly, the thesis explores the balance between accuracy and computational cost in stereo matching CNNs. Employing a data-driven strategy, it seeks to control this trade-off by harnessing spatially adaptive computation to minimize redundant computations effectively. By skipping computations in specific locations, our findings indicate that the efficiency of stereo matching CNNs can be improved while maintaining accuracy.

Expanding the scope, the work tackles the optimization of semantic segmentation CNNs, focusing on either Floating Point Operations (FLOPs) or practical latency. This is achieved through a data-driven optimization methodology which leverages spatially adaptive processing, aiming to meet a specified target budget. When considering practical latency, execution time measurements from target devices are introduced to guide the optimization process.

Lastly, the multifaceted challenge of performing inference at multiple computational budgets while considering the memory footprint of the model weights is explored. Employing spatially adaptive dynamic execution methods, the goal is to strike a balance between memory constraints and computational efficiency, addressing the challenge of optimizing models within restricted memory environments, when multiple performance points are required.

In conclusion, this study explores optimization of CNNs for vision tasks including stereo matching, semantic segmentation and image classification, probing the potential to balance accuracy against multiple axes by considering FLOPs, latency, and memory. It aims to leverage spatially adaptive computation, dynamic data-driven techniques, and novel architectures to enhance efficiency, offering a holistic approach that navigates the trade-offs and boundaries of neural network optimization.

Acknowledgements

I extend my heartfelt gratitude to my supervisor, Prof. Christos-Savvas Bouganis, whose unwavering support and mentorship have been instrumental in shaping this work. His guidance, academic insights, and encouragement have been invaluable, guiding me through challenges and fostering my growth as a researcher. I am deeply grateful for the opportunity to learn and develop under his supervision.

I would also like to express my sincere appreciation to my co-supervisor Dr. Dimitrios Tzovaras for his guidance and support throughout the entirety of my research endeavor.

Lastly, I extend my appreciation to all who generously shared their time and insights, without whom this research would not have been possible.

Dedication

This thesis is a testament to the collective support, encouragement, and love of all those who have played a part in my academic journey. Thank you for believing in me.

Declarations

Declaration of Originality

I herewith certify that the work presented in this thesis is my original own. All material in this thesis which is not my own work has been appropriately referenced.

Declaration of Copyright

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC).

Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose.

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. Where a work has been adapted, you should indicate that the work has been changed and describe those changes. 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.

Contents

Abstract i

Acknowledgements iii

1 Introduction 1

1.1 Computer Vision 1

1.2 Spatially Adaptive Processing 4

1.2.1 Motivation 5

1.2.2 Key Literature 5

1.2.3 Limitations and Gaps 6

1.3 Research Questions 6

1.4 Contributions - Outline 7

1.5 Publications 8

2 Background Theory 9

2.1 Convolutional Neural Networks 9

2.1.1 Dense and Sparse Computation 10

2.2 Quantifying Performance 13

2.3 Introduction to Dynamic Inference 15

2.4 Spatially Adaptive Computation 18

2.4.1 Introduction 18

2.4.2 Static Sparsity Patterns 20

2.4.3 Dynamic Sparsity Patterns 22

2.4.4 Grouping Image Regions 25

2.5 Conclusions and Limitations 30

3 Data-Driven spatial sparsity in stereo-matching 33

3.1 Introduction 33

3.1.1 Depth Estimation 33

3.1.2 Binocular Stereo Matching 34

3.1.3 Prior Art and Current State of the Art 36

3.2 Motivation 49

3.3 Methodology 51

3.3.1 Method Overview 51

3.3.2 Blockwise Data Handling 52

3.3.3 Ground Truth Construction 53

3.3.4 Mask Estimation Sub-network 54

3.3.5 Accuracy Restoration 56

3.3.6 Inference Stage 57

3.4 Experimental Results . . . 58
3.4.1 Implementation Details . . . 58
3.4.2 Quantitative Results . . . 60
3.4.3 Qualitative Results . . . 62
3.4.4 Discussion . . . 63
3.5 Conclusions . . . 64

4 Latency-aware spatially adaptive optimization for semantic segmentation 69

4.1 Introduction . . . 69
4.1.1 Semantic Segmentation . . . 69
4.1.2 Prior and Current State-of-the-Art . . . 71
4.1.3 Contribution . . . 82
4.2 Motivation . . . 83
4.2.1 Background Motivation . . . 83
4.2.2 Hands-on Motivation . . . 84
4.3 Methodology . . . 86
4.3.1 Problem Formulation . . . 87
4.3.2 Sparse Architecture . . . 88
4.3.3 Sparse Training Framework . . . 90
4.3.4 Latency Aware Optimization . . . 94
4.4 Experimental Results . . . 101
4.4.1 Target Networks . . . 101

4.4.2 Train Setting . . . 102
4.4.3 Quantitative Results . . . 103
4.5 Ablation Studies . . . 105
4.5.1 Block Size . . . 106
4.5.2 Target budget . . . 107
4.5.3 Slope and loss boundary terms . . . 108
4.5.4 Qualitative Results . . . 108
4.6 Discussion . . . 109
4.6.1 Class Sparsity Analysis . . . 109
4.6.2 Latency vs FLOPs Optimization . . . 114
4.6.3 Comparison to SegBlocks . . . 115
4.7 Conclusions . . . 115

5 Memory-aware Spatially Adaptive Inference for CNNs 117

5.1 Image Classification . . . 117
5.1.1 Evolution of Methods . . . 118
5.1.2 Datasets . . . 123
5.1.3 Evaluation . . . 126
5.2 Memory-aware Dynamic Performance . . . 126
5.2.1 Introduction . . . 126
5.2.2 Limitations . . . 131
5.2.3 Contributions . . . 132

5.3 Motivation . . . 133
5.3.1 Background Motivation . . . 133
5.3.2 Hands-on Motivation . . . 134
5.4 Methodology . . . 134
5.4.1 Problem Formulation . . . 134
5.4.2 Multi-Budget Training with Sparse Optimization Framework . . . 135
5.5 Experimental Results . . . 138
5.5.1 Image Classification . . . 139
5.5.2 Semantic Segmentation . . . 143
5.5.3 Scalability Analysis . . . 144
5.5.4 Discussion . . . 145
5.5.5 Qualitative Results . . . 149
5.6 Ablation Studies . . . 150
5.7 Conclusions . . . 152

6 Conclusion **153**

6.1 Summary of Thesis Achievements . . . 153
6.2 Limitations and Challenges . . . 155
6.3 Future Work . . . 157
6.3.1 Future Research . . . 157
6.3.2 Short-term Extension . . . 158
6.3.3 Long-term Extension . . . 158

References 174

List of Tables

2.1 Table presenting the most common spatially adaptive computation schemes. F_1 denotes the computation function for important and F_2 for unimportant F_2 regions. Arrows represent upsampling and downsampling operations. 19

2.2 The discussed spatially adaptive dynamic methods of the background chapter, summarizing the relevant key elements. 32

3.1 The theoretic performance of the GPUs, used in the reported methods according to techpowerup.com. 45

3.2 FLOPs analysis of StereoNet architecture for input image with size of 960×540 pixels. 48

3.3 Patch Architecture for $[60, 60]$ patch. 55

3.4 Evaluation on KITTI 2012 and KITTI 2015 datasets. The percentage of pixels with error bigger than 3 is reported, along with the overall EPE. 60

3.5 Evaluation on Sceneflow-finalpass dataset. Codes of cited methods are used from their public repositories. Sparsity in our approaches is denoted with the sparsity percentage in parentheses. Execution times are measured on a Jetson AGX under MAXN power profile. 62

4.1 FLOP bounds for spatially adaptive optimization for DDRNet23-slim, DDRNet23 (Hong et al., 2021) and HRNet (J. Wang et al., 2021) networks, for both full and high resolution branch compression. 102

4.2 Hyperparameters used for training the sparse networks on Cityscapes (Cordts et al., 2016) dataset. 102

4.3 Table presenting the result of optimization of DDRNet23-slim, DDRNet23 and HRNetV2-48 multibranch architectures, evaluated on the validation set of Cityscapes dataset, under various FLOPs reduction regimes. 104

4.4 Latency optimization results for DDRNet23 (Hong et al., 2021) network performed on two different GPU devices: a desktop NVIDIA RTX 3060ti and an embedded NVIDIA Jetson NX. 106

4.5 DDRNet23-slim optimized with target budget $f_b = 0.50$ with varying blocksizes in full image resolution. Latency measured on a NVIDIA RTX 3060ti. 106

4.6 The impact of removing the initial boundary loss term and addition of the slope in the timing function, for latency optimization of the high resolution branch of HRNetV2-48 using $l_b = 0.8$. 107

4.7 Feature table comparing the key elements of the relevant spatially adaptive methods with our approach on dual branch CNNs for semantic segmentation. 115

5.1 Table summarizing the memory cost to store weights for operation at N FLOP regimes, when using each technique to achieve dynamic performance. 130

5.2 The number of parameters needed to store the weights of backbone, mask estimation, and normalization layers, alongside their respective percentages relative to the backbone. 133

5.3 Scalability of parameters for multi-target inference with N targets. W_B, W_M denote backbone and mask estimation module parameters for each method correspondingly. 144

5.4 Feature table comparing the key elements of the relevant spatially adaptive methods with our approach on CNNs for image classification at multiple FLOP budgets. 145

List of Figures

1.1 Figure illustrating the varying importance of content of patches (a) and (b) of different spatial locations for visual understanding tasks. Base figure reprinted from "https://developersbreach.com/convolution-neural-network-deep-learning/" (n.d.) and augmented with patch visualization. 4

2.1 Example of a typical CNN architecture for image classification task. Figure reprinted from "https://hips.hearstapps.com/hmg-prod/images/dog-puppy-on-garden-royalty-free-image-1586966191.jpg?ref=blog.paperspace.com" (n.d.). . . 11

2.2 Figure presenting a categorization of the dynamic networks according to the type of dynamic behavior they entail (sample-wise, spatial-wise and temporal-wise). Figure reprinted from Han, Huang, Song, Yang, Wang, and Wang (2021). 16

2.3 Visualization of feature maps, which highlight the discriminative image regions for classification, e.g. the head of the animal for briard and hen, the plates in barbell, and the bell in bell cote. Figure reprinted from Zhou et al. (2016). . . . 17

2.4 Figure presenting the evolution of spatially adaptive methods over time. Each method is followed with the key contributions or novelties it proposed. Per-layer masking and the Gumbel-Softmax trick Jang et al. (2016) training technique have become standard practice since their first introduction in 2017 and 2020 respectively, and are omitted from more recent methods. Most works experiment with residual-style networks such as ResNet He et al. (2016), therefore mentioning support for this architecture is also omitted. 20

2.5 Perforation binary patterns proposed in Figurnov et al. (2016). 21

2.6 Halting score calculation module from M. Figurnov et al. (2017). 23

2.7 Guided by a sparse sampling mask M based on the input feature map X, features Y_s at active points are calculated. Features at the rest of locations are interpolated by the interpolation module C leading to the output feature map Y^*. Figure reprinted from Z. Xie et al. (2020). 24

2.8 The spatially adaptive computation scheme of Rao et al. (2021) and Rao et al. (2023): Guided by a binary mask decision, tokens are dropped and self-attention is calculated only between the remaining tokens. Figure reprinted from Rao et al. (2021). 27

2.9 The spatially adaptive compuation scheme proposed in Han, Huang, Song, Yang, Zhang, and Jiang (2021), where a base (top) branch processes features at a lower resolution and a refinement (bottom) branch refines the important regions at full resolution. Figure reprinted from Han, Huang, Song, Yang, Zhang, and Jiang (2021). 28

2.10 The framework proposed in Verelst and Tuytelaars (2023). A policy network predicts which blocks are going to be executed in full resolution. Training is performed by calculating a reward for each block, conditioned on the task loss and by supervising the ratio between high and low resolution blocks. Figure reprinted from Verelst and Tuytelaars (2023). 29

3.1 Stereo image rectification 34

3.2 3D reconstruction pipeline using stereo matching. 35

3.3 Samples from (a) sceneflow Mayer et al. (2016) and (b) KITTI 2012 and 2015 Menze et al. (2015), Menze et al. (2018) datasets. Left RGB images are organized left column with their corresponding disparity maps in the right column. 44

3.4 State of the Art EPE on sceneflow dataset 45

3.5 The state-of-the art landscape for stereo matching methods in terms of EPE vs normalized computational complexity on sceneflow test dataset as of 2022. Top (a) image includes all methods, whereas bottom (b) is limited to methods requiring less than 100 GFLOPs. 46

3.6 Figure presenting the StereoNet (Khamis et al., 2018) architecture, noting the typical steps of a stereo matching algorithm (bottom) according to taxonomy from Scharstein and Szeliski, 2002. Figure reprinted from Khamis et al. (2018) and augmented with additional information. 48

3.7 Figure demonstrating the change in EPE for the full sceneflow dataset with respect to sparsity using block format. Orange dots indicate network operation states that skip regions in an error-aware way, whereas blue dots indicate network operation states that skip picture regions at random. 50

3.8 The architecture and flow of StereoNet marked in black color. Our proposed methodology (in red color) guides the the computations to more meaningful regions, according to the output of a parallel Mask Estimation Network. 51

3.9 Figure displaying possible tensor handling methods, along with memory overheads. Red color indicates the input to the processing pipeline, while green is the output: (a) At *pixel* level every pixel is a single element, that carries its whole receptive field , (b) at *full-block* level each non-overlapping block carries the same receptive field, resulting in smaller memory overhead, and (c) at *fast-block* level, we drop the receptive field of border pixels, in favor of performance. 52

3.10 The architectures of the two proposed networks for Mask Estimation: *Full* architecture (top) and *Patch* architecture (bottom). 54

3.11 Graph presenting the spatially adaptive inference process of the disparity refinement part. 57

3.12 Figure presenting the results of our optimized method in sceneflow dataset. The top-left graph depicts the SS-StereoNet sparse operation states with respect to the theoretical FLOPS. The actual execution time of those operation states is presented, by implementing them on the following devices: NVIDIA GTX 1060 (top-right), NVIDIA Jetson AGX under MAXN profile (bottom-left) and NVIDIA Titan X Maxwell (bottom-right). 59

3.13 Evaluation on Sceneflow-finalpass dataset. Codes of cited methods are used from their public repositories. Execution times in milliseconds for a batch with size equal to 1 are measured on a Jetson AGX under MAXN power profile. 61

3.14 Qualitative evaluation of a sample from sceneflow dataset. Sparsity of SS-StereoNet is denoted in parentheses. Third row contains disparity estimate of selected patch samples from the disparity map above them. Note how Patch #2 is refined even at 70% sparsity, whereas refinement of Patch #1 is skipped at higher sparsity. 66

3.15 Qualitative evaluation of sceneflow dataset samples. Columns from left to right include: RGB input, Ground Truth, SS-StereoNet-Patch at 50% sparsity and disparity estimate from AnyNet Y. Wang et al., 2019 method. 67

3.16 Evaluation on KITTI 2015 samples. Columns contain: input image (a), ground truth (b), SS-StereoNet-Full with 50% sparsity (c), SS-StereoNet-Patch with 50% sparsity (d). Top rows of each sample contain sparsity masks, while bottom rows contain disparity estimates. 68

4.1 The semantic segmentation (a) of a scene (b) from Cityscapes Cordts et al. (2016) dataset 70

4.2 Popular backbone architectures for semantic segmentation. Figure reprinted from Hong et al. (2021). 72

4.3 HRNet (J. Wang et al., 2021) architecture for semantic segmentation. Figure rerpinted from J. Wang et al. (2021). 74

4.4 DDRNet (Hong et al., 2021) architecture for semantic segmentation. Figure reprinted from Hong et al. (2021). 74

4.5 Samples from general purpose semantic segmentation datasets (a) ADE20K (Zhou et al., 2019), (Zhou et al., 2017) and (b) PASCAL VOC 2012 (Everingham et al., 2015). Samples are organized in pairs of RGB image and ground truth segmentation mask. 76

4.6 Samples from urban driving semantic segmentation datasets (a) CamVid (Brostow et al., 2009) and (b) Cityscapes (Cordts et al., 2016). Samples are organized in pairs of RGB image and ground truth segmentation mask. 76

4.7 Evolution of state-of-the-art in semantic segmentation in mIOU% on Cityscapes (Cordts et al., 2016) validation dataset, sorted from newer (top) to older (bottom). Letter in parenthesis denotes the architecture category they use according to Figure 4.2. 78

4.8 The state-of-the art landscape for semantic segmentation matching methods in terms of mIOU% vs normalized computational complexity on Cityscapes (Cordts et al., 2016) validation dataset as of 2022. Top (a) image includes all methods, whereas bottom (b) is limited to methods requiring less than 150 GFLOPs and mIOU% over 75%. 80

4.9 The mIOU% on Cityscapes (Cordts et al., 2016) validation dataset over the actual latency on two devices: (a) an NVIDIA GTX 1060 desktop GPU and (b) an NVIDIA Jetson AGX Xavier embedded GPU. 81

4.10 Figure demonstrating the summed feature maps along channel dimension of DDRNet23-slim (Hong et al., 2021) after residual layers of the high resolution branch (top) and the low resolution branch (bottom), from early (left) to latter stages (right). 84

4.11 Introduction of random mask modules (red blocks) in DDRNet (Hong et al., 2021) architecture (Hong et al., 2021). Figure reprinted from (Hong et al., 2021) and augmented with additional information. (a-f) letters denote layers in correspondence with feature maps of Figure 4.10. 85

4.12 Performance of DDRNet23-slim (Hong et al., 2021) by introducing sparsity in a random manner in the both branches (green) and only in high resolution branch (blue). 86

4.13 The adaptation of (a) a vanilla residual block for sparse execution using (b) mock sparsity implementation and (c) practical implementation using Ren et al. (2018) method. 89

4.14 Figure demonstrating the Gumbel-Softmax trick that resembles discrete sampling. 91

4.15 The impact of softmax temperature τ in the output expectation. Figure reprinted from Jang et al. (2016). 92

4.16 The sparse execution proposed by SBNet (Ren et al., 2018) denoting the basic operations. Figure reprinted from Ren et al. (2018). 95

4.17 Latency functions for the first residual layer of DDRNet23-slim used for training. 98

4.18 Figure presenting the sparse profile for different layers (rows) of DDRNet23-slim (Hong et al., 2021) in two devices: NVIDIA RTX 3060ti (a-d) and NVIDIA JetsonNX (e-h) under various blocksizes (colored lines). Dashed line denotes the dense execution time for the corresponding layer. 99

4.19 FLOPs optimization of DDRNet23-slim (Hong et al., 2021), DDRNet23 (Hong et al., 2021) and HRNetV2-48 (J. Wang et al., 2021) multibranch architectures compared to SoA compression methods using spatial sparsity. Multiple connected points are generated by evaluating the full resolution models at different scales. 104

4.20 Performance of sparse DDRNet23 (Hong et al., 2021) under various target budgets $fb \in [0.1, 0.2.., 1]$ 107

4.21 Percentages of active pixels for each class of sparse DDRNet23 Hong et al., 2021 network. Pixels of *coarser* classes (underlined) are more frequently masked than those from *finer* classes. 109

4.22 Qualitative evaluation for latency optimized DDRNet23-slim for $lb = 0.80$ compared to their dense counterpart. Rows from top to bottom contain RGB input, ground truth, corresponding network dense output, sparse output, summed cost maps and scale colorbar. 110

4.23 Qualitative evaluation for latency optimized HRNetV2-W48 (b) for $lb = 0.80$ compared to their dense counterpart. Rows from top to bottom contain RGB input, ground truth, corresponding network dense output, sparse output, summed cost maps and scale colorbar. 111

4.24 Cityscapes samples (left column) with output from DDRNet23-slim hb optimization under 80% FLOPs (middle) and 80% latency (right) targets. Each sample contains the RGB image (top left), annotated ground truth (bottom left), and middle and right colums show the per-layer estimated binary masks (top) and output segmentation map (bottom). 112

4.25 Sparsity mask inspection for flops (HRNetV2-W48 fb=0.50) and latency (HRNetV2-W48 lb=0.80) optimization of the same model. Figures (a) and (b) contain the semantic segmentation map outputs, and (c) demonstrates the per-layer sparsity mask of each network. 113

4.26 Histogram with actiation rates for sparse layers of HRNetV2-48 optimized for FLOPs ($f_b = 0.5$) and for latency ($l_b = 0.8$). 114

5.1 Figure with ViT architecture (left) from the original paper Dosovitskiy et al. (2020) along with the architecture of the Multi-Head attention block (right). . . 121

5.2 Samples from (a) MNIST L. Deng (2012) and (b) CIFAR 10 and CIFAR 100 Krizhevsky (2012) datasets. 123

5.3 Samples from Imagenet-1k (J. Deng et al., 2009) 124

5.4 top1 accuracy % to GFLOPs visualization of state-of-the-art methods in Imagenet-1k (J. Deng et al., 2009) image classification dataset, from early (lighter colors) to recent (darker colors) deep learning methods. Size of bubble represents the number of parameters of each model. 125

5.5 Graph of the multi-target spatially adaptive optimization framework. It receives a network and target FLOP budgets and generates a spatially adaptive variant able to operate at the target FLOP budgets with minimal increase in number of parameters. 132

5.6 Spatially adaptive residual layers capable for inference at a single budget (a) or supporting N multiple budgets b_i (b). Referencing b_i, a budget-specific mask unit and normalization layer is selected. 136

5.7 Results of multi-target inference with ResNet models on CIFAR10 dataset. . . . 140

5.8 Results of multi-target inference with ResNet models on ImageNet-1k dataset. . 141

5.9 Results of multi-target inference with ConvNeXt models on ImageNet-1k dataset. 142

5.10 Results of multi-target inference with semantic segmentation DDRNet23/23-slim models on Cityscapes dataset. 143

5.11 Scalability of parameters for ResNet and ConvNeXt models as a multiplier to base network parameters W_B. 145

5.12 Aggregated (summed) layer masks (middle column) and individual masks (right column) for target budgets of 30%, 50% and 70% for ResNet101 model. 146

5.13 Aggregated (summed) layer masks (middle column) and individual masks (right column) for target budgets of 30%, 50%, 70% and 90% for ConvNeXt-T model. 147

5.14 Aggregated (summed) layer masks (middle column) and individual masks (right column) for target budgets of 30%, 50%, 70% and 90% for ConvNeXt-B model. 148

5.15 Aggregated layer masks (middle column) and individual masks (right column) for ResNet101 and ConvNeXt-B models. 149

5.16 Ablation studies for target awareness weighting, knowledge distillation setting and range of supported budgets. 151

6.1 Figure presenting the latency of executing different layers of ConvNeXt-T Z. Liu et al. (2022) under varying sparsity levels in two CPU devices: Intel i7-12700 (a-d) and an AMD Ryzen 3 1200 (e-h) under various blocksizes (colored lines). Dashed line denotes the dense execution time for the corresponding layer. 156

Chapter 1

Introduction

In this chapter, several fundamental aspects will be introduced to provide a comprehensive understanding of the research scope. First, the field of computer vision is presented along with its multifaceted tasks, through which computational systems discern and comprehend visual data. Next, the concept and principles of spatially adaptive processing will be introduced, followed by the key literature of the field. Finally, after identifying key limitations, the research questions which guide this research are formulated, followed by the corresponding contributions to address them.

1.1 Computer Vision

Computer vision stands at the intersection of artificial intelligence, machine learning, and image processing, empowering machines to comprehend, interpret, and extract meaningful insights from visual data (Ballard & Brown, 1982). At its core, this field aims to replicate the human visual system's capabilities, enabling computers to perceive and understand the visual world in a similar fashion to humans (Poggio, 1984), (Biederman, 1985). Computer vision finds widespread applications across industries and domains. From enhancing manufacturing through automated quality control (Brosnan & Sun, 2004), (Kumar, 2008) processes to revolutionizing healthcare by aiding in medical image analysis and diagnostics (Cootes & Taylor,

2001), (Heimann & Meinzer, 2009), (Maintz & Viergever, 1998), (Doi, 2007), its impact is profound. In surveillance and security (Coifman et al., 1998), (Oliver et al., 2000), computer vision enables intelligent monitoring and threat detection, contributing to public safety. Moreover, environmental and agricultural sectors benefit from computer vision's ability to automate processes like crop monitoring (Brosnan & Sun, 2002) and environmental analysis (Dandois & Ellis, 2010), (Moor et al., 2023), (Lacoste et al., 2023). These applications collectively underscore the versatility and importance of computer vision solutions in driving efficiency, accuracy, and innovation across diverse fields.

When computer vision first emerged in the middle of the 20th century, its main focus was on fundamental image processing methods, such as edge detection (Marr and Hildreth (1980) and simple feature extraction (Nevatia and Babu (1980). Researchers made significant advancements in feature extraction and pattern recognition algorithms throughout the 1970s, which set the stage for later breakthroughs in classification (Stockham, 1972) and object detection (Stockman et al., 1982). By the 1980s, research on hierarchical visual processing (Adelson et al., 1984) had grown, highlighting the need of comprehending how levels of processing—similar to those found in biological systems—are used to analyse visual information (Riesenhuber & Poggio, 1999). The idea of computer vision systems intended to imitate this hierarchical structure was prominent during this time. With advances in object detection and tracking in the 1990s (Papageorgiou et al., 1998), (Tomasi & Kanade, 1991), computer vision research shifted into domains focused on extracting more complex information from visual data. Statistical models (Denning, 1987), (Denning, 1987) also began to appear at this time, bringing probabilistic and machine learning methods which contributed significantly to the field. These breakthroughs, which occurred in the fields of image analysis, feature extraction, object recognition, and statistical modelling, laid the groundwork for modern computer vision approaches.

Computer vision encompasses a spectrum of tasks aimed at enabling machines to comprehend, interpret, and extract meaningful information from visual data. Each task serves a distinct purpose in understanding and processing visual information, contributing to the broader goal of replicating human-like visual perception in machines.

Image classification, a fundamental task in computer vision, involves assigning predefined labels to images based on their visual content. Its significance spans across industries, aiding in content-based image retrieval, sorting systems, and efficient data organization. Crucially, image classification acts as a foundational step for other tasks, providing input for more complex processes like object detection and segmentation. Typically, it takes one or multiple images as input and outputs labels or categories for the depicted object or scene, from a set of predefined classes.

Object detection focuses on identifying and locating various objects within an image. This task holds paramount importance in diverse applications, including surveillance and robotics. It extends beyond image classification by not only recognizing objects but also determining their specific positions within the image. Object detection takes images as input and outputs identified objects along with their corresponding spatial information, contributing significantly to understanding the visual content in scenes.

Semantic segmentation assigns semantic labels to every pixel within an image, enabling a detailed understanding of object boundaries and different regions. This task holds significance in fields like medical imaging and autonomous driving, providing detailed scene comprehension. Unlike object detection, semantic segmentation offers pixel-level classification, delineating object boundaries and labeling each pixel with its corresponding semantic class. While providing per-pixel labeling for an image, it does not make distinction between object instances as object detection. Typically taking an image as input, the output consists of pixel-wise semantic labels, aiding in detailed and precise interpretation of visual data.

Stereo matching attempts to estimate the depth of each pixel in an image by finding corresponding pixels in two or more images taken from different viewpoints, usually acquired from stereo cameras. These correspondences are encoded as the displacement in a disparity image which can be converted to depth values using camera parameters. Stereo matching is used for 3D reconstruction and is a key component of applications such as virtual and augmented reality. The input for stereo matching comprises of stereo image pairs, while the output typically includes depth maps indicating the perceived depth or the estimated disparity map.

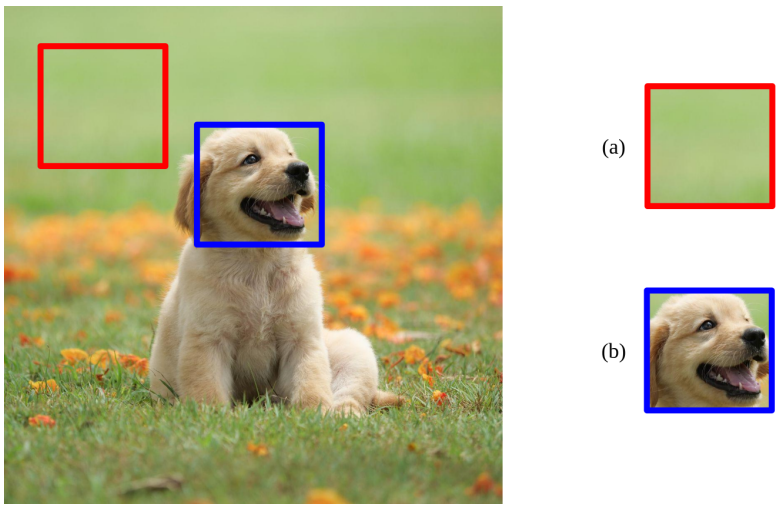


Figure 1.1: Figure illustrating the varying importance of content of patches (a) and (b) of different spatial locations for visual understanding tasks. Base figure reprinted from "https://developersbreach.com/convolution-neural-network-deep-learning/" (n.d.) and augmented with patch visualization.

To solve the above computer vision tasks, traditional methods relied heavily on handcrafted features, such as edges, corners, and textures. However, these techniques were not always successful when dealing with complicated variations in data, such viewpoint shifts, illumination variations, and occlusions. Moreover, the reliance on manually engineered features limited their adaptability to diverse scenarios and led to issues of robustness and scalability. To overcome these difficulties, pipelines involving feature extraction, transformation, and classification were employed. Traditional methods also struggled to generalize effectively across diverse datasets and demonstrated challenges in handling the diversity present in real-world visual data. Consequently, robustness, adaptability, and scalability constraints of traditional computer vision methods, highlighted the necessity of a paradigm shift in tackling these core issues.

1.2 Spatially Adaptive Processing

In this section, the concept of spatially adaptive processing will be introduced, discussing the core principle and motivation behind it. Moreover, a high-level overview of the key literature in the field will be provided, highlighting significant contributions. Finally, the limitations and gaps identified in the current research will be discussed, setting the stage for the research

questions that this thesis aims to address.

1.2.1 Motivation

To achieve visual understanding, computer vision methods and algorithms process an input image and extract meaningful information for the task at hand. The standard practice involves performing computations on the whole image, spatially distributing computations uniformly. However, the importance of information for visual understanding varies wildly between different parts of the image. An example of such case can be seen in Figure 1.1, where the blue patch contains most of the critical information, whereas the red patch contains a part of the background which less relevant data. This contrast in the importance of information across different image regions highlights the potential inefficiency of spending an equal amount of computational resources on all areas.

Motivated by the fact that spending equal amount of compute seems non-optimal, the family of spatially adaptive methods has emerged. The core idea behind these methods is to vary the type and amount of computation based on the significance of different regions within the image. By implementing rules to distinguish between important and less important areas, these methods aim to enhance efficiency by reducing redundant computations and focusing resources where they are most needed.

1.2.2 Key Literature

The field of spatially adaptive processing has evolved significantly over time, beginning with early works that focused on identifying regions in images where less computational resources could be spent. These initial approaches introduced the concept of skipping or reducing computations in certain areas based on static patterns, which were defined a priori (Figurnov et al., 2016), (Ren et al., 2018). To address the rigidity of static patterns, dynamic and data-driven approaches for importance estimation were introduced, where the importance of image regions was estimated based on the each input sample (M. Figurnov et al., 2017), (Verelst &

Tuytelaars, 2020), (Z. Xie et al., 2020). These adaptive methods have since dominated the field, offering increased flexibility and better alignment with the underlying data, leading to improved efficiency.

While spatially adaptive methods achieved computational cost savings, improvements in practical aspects like latency and memory remained limited due to the overheads related to applying spatially adaptive computation in practice. To address these challenges, a set of methods emerged that focused on reducing latency overheads by using block granularity for importance estimation patterns (Rao et al., 2021), (Rao et al., 2023), (Han et al., 2022), (Han et al., 2023), (Verelst & Tuytelaars, 2023).

1.2.3 Limitations and Gaps

Despite significant progress, the literature on spatially adaptive processing still faces several limitations. One key challenge is the lack of a consistent methodology for holistically addressing the computational aspects necessary for deployment on actual devices, particularly concerning latency and the memory required for storing network parameters. Additionally, it is crucial to address the challenge of increasing the efficiency of recent modern architectures, which are already more compact and efficient compared to older ones, leaving reduced room for optimization. These limitations set the stage for the research questions of this thesis, which aims to address these challenges and towards advancing the field of spatially adaptive processing.

1.3 Research Questions

The research questions in this thesis are the fundamental inquiries that guide the study of spatially adaptive processing for CNNs in computer vision tasks. They aim to address key aspects or gaps in understanding how spatially adaptive processing can be used as a means to increase the efficiency of CNNs by considering multiple aspects. To summarise, the main thesis research questions are the following:

- How can we use data-driven spatially adaptive methods to increase the efficiency of a stereo-matching CNN?
- How can we address computational aspects of data-driven spatially adaptive methods focusing on semantic segmentation CNNs?
- How can dynamic inference for multiple target computational budgets be achieved while focusing on optimizing memory usage?

1.4 Contributions - Outline

The thesis' contributions are organised as follows:

Chapter 2 introduces the background on dynamic networks and delves into the relevant literature on the field of dynamic spatially adaptive computation. It is followed by a discussion on the contributions and limitations of the relevant works, facilitating the introduction of the thesis' own contributions.

Chapter 3 presents a method which exploits spatial sparsity of computations of a stereo-matching CNN, to increase its efficiency and achieve inference for different computational budgets. To identify redundant computations, a parallel CNN is used, tasked with estimating the importance of refining image regions. Guided by this information and employing a spatially adaptive processing scheme, refinement of regions is only executed where the impact exceeds a specific threshold, controlling a trade-off between performance and computational cost. Additionally, the chapter highlights key insights into the challenges encountered during the training of these rules and the crucial considerations regarding design choices and constraints in deploying sparse models in real devices.

Chapter 4 introduces a tool-flow designed to incorporate spatially adaptive computation into CNNs for semantic segmentation, aiming to enhance their practical efficiency within a latency-aware framework. Latency information from real benchmarks on target devices is key to guiding the optimization process, whereas a block-granular spatial adaptiveness is used to facilitate

practical gains on GPU devices. This presents a strategic approach tailored to incorporate spatial sparsity in an end-to-end trainable optimization process, achieving device-tailored spatially sparse optimization.

Chapter 5 considers the growing number of parameters of spatially adaptive methods for achieving controllable performance at multiple target computational budgets. The base spatially adaptive architecture is updated to operate at multiple budgets on-demand, while a training scheme is designed, to facilitate optimal performance at all desired budgets. It presents a methodology towards achieving controllable dynamic sparsity in CNNs using spatially adaptive computations, navigating the trade-offs between performance, computational cost, and number of parameters.

Finally, **Chapter 6** provides a discussion on the findings and contributions of this thesis, and outlines limitations and challenges that exist in the field of spatially adaptive dynamic computation, and suggests potential directions for future research.

1.5 Publications

This section lists the key peer-reviewed publications arising from this thesis research, highlighting the contributions and outcomes achieved.

[P1] Zampokas, G., Bouganis, C-S. & Tzovaras D. (2022) *Pushing the efficiency of StereoNet: Exploiting spatial sparsity*, in 17th International Conference on Computer Vision Theory and Applications (VISAPP) 2022.

[P2] Zampokas, G., Bouganis, C-S. & Tzovaras D. (2024) *Latency Driven Spatially Sparse Optimization for Multi-Branch CNNs for Semantic Segmentation*, in Large Language and Vision Models for Autonomous Driving (LLVM-AD) Workshop, held in conjuction with IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) 2024.

[P3] Zampokas, G., Bouganis, C-S. & Tzovaras D. (2024) *Budget-aware Dynamic Spatially Adaptive Inference*, in the 35th British Machine Vision Conference (BMVC) 2024.

Chapter 2

Background Theory

This chapter provides a comprehensive overview of the foundational theories and background information relevant to this research. It begins with an exploration of Convolutional Neural Networks (CNNs), focusing on the distinction between dense and sparse computation within these models. Next, performance and compute metrics essential for evaluating CNN efficiency are presented. Finally, an introduction to dynamic inference is presented, with a particular emphasis on spatially adaptive techniques, including a review of the key literature in this area. Together, these sections lay the groundwork for the subsequent analysis presented in the contribution chapters.

2.1 Convolutional Neural Networks

The shortcomings of traditional computer vision methods, because of their reliance on manually crafted features (Lowe, 1999), (Bay et al., 2008), (Rublee et al., 2011), underscored the need for innovative approaches to ensure improved generalization and robustness (O'Mahony et al., 2020), (Voulodimos et al., 2018), (Georgiou et al., 2019). In a pursuit to mimic the remarkable capabilities of the human brain, neural networks are an effort to replicate cognitive functions in artificial systems (Fukushima et al., 1983), (Fukushima, 1988), (Fukushima, 1980), (Weng et al., 1992). CNN architecture is inspired by the connectivity inside the human brain – in

particular, the visual cortex (Hubel & Wiesel, 1962), (Cadieu et al., 2014), which plays a key role in processing visual stimuli. The artificial neurons in a CNN are arranged hierarchically and sequentially to efficiently process visual information, enabling these models to process entire images or videos.

CNNs revolutionized computer vision by introducing the concept of learnable features, departing from traditional handcrafted features (LeCun et al., 2015). At their core, CNNs employ convolutional operations as building blocks, which involve sliding learnable filters across input data and extracting features. Stacking one or more convolutional layers constitutes a CNN, with multiple layers forming a deep CNN, which is able to learn increasingly complex representations (Krizhevsky et al., 2012). The weights of the convolution kernels are iteratively adjusted to a distribution, by undergoing a training process using optimization algorithms such as backpropagation and gradient descent. This training phase involves exposing the network to labeled data, allowing it to learn and adapt its internal representations to match the corresponding data distribution, ultimately providing the ability to classify, detect, or segment visual information (Rawat & Wang, 2017).

Overall, CNNs represent an important advancement in computer vision, demonstrating a paradigm shift in the domain. The use of learnable features which can adapt to complex visual data has positioned them as the cornerstone of various vision-related problems, where they consistently exhibited state-of-the-art performance (Redmon et al., 2016), (Badrinarayanan et al., 2017), (Chang & Chen, 2018).

2.1.1 Dense and Sparse Computation

When an image is processed by a CNN, each base color used in the image (e.g. red, green, blue) is represented as a matrix of numerical values. These values are stacked into 3D data structures (for color images) representing feature maps for pixels or regions of the image (O'shea & Nash, 2015). As the network execution progresses, tensors are created by passing the image through a sequence of convolutional, normalization, pooling and non-linear layers. These layers act as filters, extracting meaningful cues from the image. Figure 2.1 depicts a visual representation

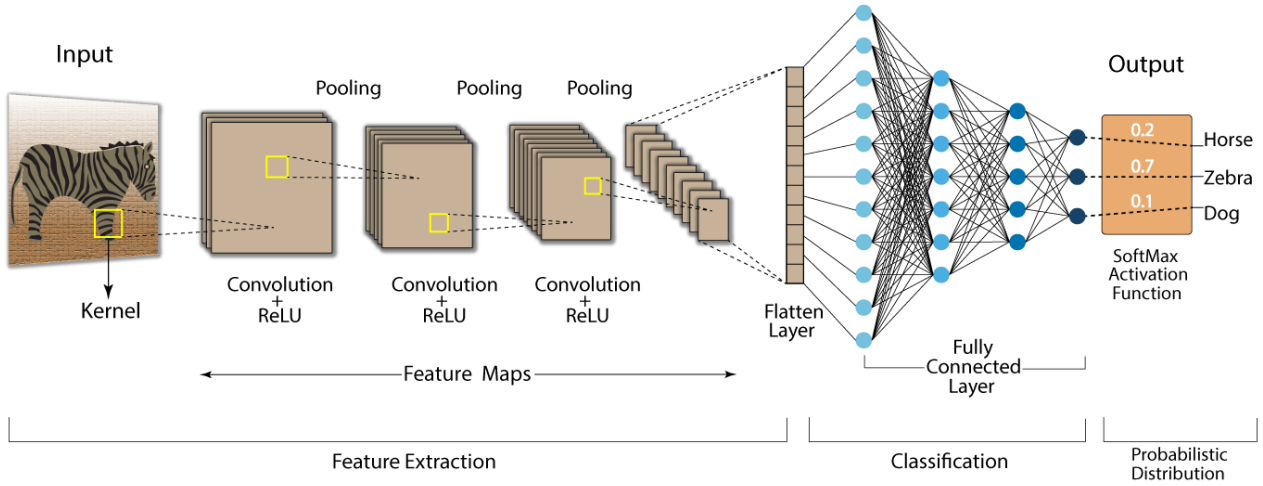


Figure 2.1: Example of a typical CNN architecture for image classification task. Figure reprinted from "https://hips.hearstapps.com/hmg-prod/images/dog-puppy-on-garden-royalty-free-image-1586966191.jpg?ref=blog.paperspace.com" (n.d.).

of a typical CNN execution. Finally, the features extracted from the convolutional layers, are typically processed by a head module, which is responsible for generating predictions and is relevant to each computer vision task. In general, as more convolutional layers are used, the expressive capability of the network is increased, which generally leads to improved predictions.

Let X represent the input feature map or image with dimensions $H \times W \times C$, where H is height, W is width, and C is the number of channels. To perform the convolution K filters K_i with dimensions $k_x \times k_y$ are applied across the input X, where k_x and k_y represent the horizontal and vertical filter kernel dimensions. The convolution operation is performed by sliding each filter K_i across the input X and computing the element-wise multiplication followed by summation. The result is a feature map Y_i for each filter K_i. These individual feature maps are then sequentially stacked or concatenated to form the final feature map Y with dimensions $(H - k_y + 1) \times (W - k_x + 1) \times K$, which represents the output of the convolutional layer. Mathematically, the output of a single convolutional layer can be represented as:

$$Y_i = \sigma\left(\sum_{j=1}^{C}(X * K_{i,j}) + b_i\right), \tag{2.1}$$

where Y_i is the i-th feature map, σ represents the activation function (e.g., ReLU), $K_{i,j}$ denotes the i-th filter's weights for the j-th input channel, and b_i represents the bias term for the i-th

feature map.

For a sequential network consisting of N convolutional layers the output feature maps from the i-th layer serve as input to the $i+1$-th layer. Thus, the output of the N-layer CNN Y_N, can be represented as a the sequence of N convolutional operations. The process described, characterized by the application of convolutions across the entirety of the input space, embodies a typical forward pass process of CNNs. The default execution strategy in CNNs to achieve the forward pass is by using **dense inference**, which uniformly distributes computations across all pixels or regions within the feature maps, ensuring that each element or pixel contributes to the network's computations

In contrast to dense, **sparse inference**, departs from the uniform computation approach of processing all elements or pixels within the input data. Instead, the allocation of computational resources varies spatially, using computations on specific regions or features of interest within the input. The motivation behind employing sparse inference comes from its potential to exploit computational redundancy within deep networks. The image representation is inherently sparse compared to other data representations such as text, as the pixel-based nature of images often results a large percentage of pixels containing repetitive or irrelevant information, contrasting the more concise and symbolic nature of textual information. This sparse nature of information within images implies that not all pixels contribute equally to the network's decision-making process, since locations within an image are often characterized by varying information densities.

Sparse inference within CNNs is based on selective computation on certain regions or features of the input. A key concept is a selection mechanism to identify and focus computations on areas where critical information or features are present, bypassing the need to process the entire input. Given a selection mechanism M in place which outputs the indices that are identified for computation as set S as $M : X \rightarrow S$, the operation in sparse inference can be formulated as:

$$Y_i = \sigma\left(\sum_{j \in S}(X * K_{i,j}) + b_i\right). \tag{2.2}$$

Sparse inference in CNNs provides an alternative computational strategy, emphasizing selective computations on regions of interest. This flexible approach aims to reduce computational

redundancy while preserving crucial information, offering a trade-off between computational cost and performance.

2.2 Quantifying Performance

Measuring the performance capabilities and attributes of a deep learning model is an important process within the context of computer vision and artificial intelligence, since metrics and quantifiable attributes serve as guiding indicators, offering insights into a model's efficiency, efficacy, and computational requirements. This information enables informed decision making during model construction or deployment, considering the limitations of target hardware and the requirements of the task at hand (Ma et al., 2018), (A. G. Howard et al., 2017), (Sandler et al., 2018), (A. Howard et al., 2019), (Hoang & Jo, 2019), (Tan & Le, 2019), (Tan & Le, 2021). Moreover, these metrics drive model optimization efforts, towards advancements aimed at enhancing performance, reducing computational and memory overhead, and improving deployment feasibility. In this work, we focus on larger GPU-accelerated models commonly used in high-performance applications. When discussing "embedded" systems, our reference is to powerful platforms like the Jetson embedded GPUs, rather than resource-constrained microcontrollers. The key characteristics and attributes that characterize a model for a specific task are presented below.

The **performance** of a model is determined by its effectiveness on solving a target task and is measured with task-related metrics (e.g. mIOU% for the semantic segmentation task). To gauge the performance of different models using a common reference, researchers have established benchmark datasets for various vision tasks. These datasets serve as a common baseline, enabling the computation of relevant performance metrics across different methods for the same problem.

The **computational cost** of a model refers to the amount of computational resources, which are required by the network to process a given input. It is determined by a number of factors, such as the size of the input data, the number and type of layers in the network, and the number

of parameters in each layer. For example, convolutional layers, which apply a set of filters to an input, are more computationally intensive compared to pooling layers, which downsample the input, since convolution operation typically requires more computations to be executed. While FLOPs in larger GPU-accelerated models primarily account for convolution operations, though this may differ in smaller/mobile models with highly optimized architectures.

Latency of a model during inference refers to the actual amount of time it takes for a model to process a given input and produce a prediction. Latency is influenced by a variety of factors, such as the size and architecture of the model, the size of the input data, the hardware used for inference and the software implementation of various operations. Computational cost is usually used as proxy for the real latency of a model, since they are both analogous to the amount of computation to process the input data. In the context of deep learning it is measured in FLOPs (Floating-point operations per second) and it mostly accounts for convolution operations. This makes the FLOPs a suitable but not precise proxy to actual latency, lacking the true information on how well this number translates to latency on real hardware using a software implementation.

Memory footprint of a model during inference refers to the memory that is required by the model to make predictions on data, which also varies depending on the size of the input data, the number of parameters, the computational cost of the the model, the target device and the software implementation. Finally, the **storage size** of a model is the amount of memory required to store the model's parameters, which include the weights of each layer. It is typically much smaller than the memory footprint during inference (especially in vision tasks), because the latter also includes the memory required to store the input data and intermediate results.

The attributes discussed above need to be considered individually and in combination with each other when evaluating models for vision tasks, according to the task requirements and hardware limitations. On one hand, increasing the model size can improve performance but may lead to a large model which can't be stored on some devices, or exhibit lower latency than desired. On the other hand, using a small number of operations can produce a fast and lightweight model, which might not perform well in the given task. This trade-off is defined as **efficiency** in CNNs and refers to the ability of the model to make accurate predictions using a small

amount of resources, such as memory and computational budget. Thus, an efficient network exhibits a favorable trade-off, and makes accurate predictions using a small amount of memory and compute. Finally, to facilitate informed decisions on the discussed trade-offs, it is useful to consider the pareto front of performance to other attributes, ensuring the identification of designs with favorable trade-off between the two or more objectives.

2.3 Introduction to Dynamic Inference

In terms of information processing and representation, most of the prevalent deep learning models perform inference in a static manner, by keeping both the computational graph and the network parameters fixed once trained. This static nature, while offering stability and predictability, may limit the representation power, efficiency, and interpretability of these models.

In contrast to static, dynamic models have the capacity to adapt their structures or parameters during inference based on the input they receive. This is critical from an efficiency perspective, since on-demand activation allows them to focus computational resources where needed, significantly reducing redundant computations and improving overall efficiency. Moreover, this allows dynamic models to strike a desired trade-off between accuracy and varying computational or memory budgets on the fly. Therefore, they can adapt to different hardware platforms and environments, compared to static models with a fixed computational graph and parameters.

Dynamic models are generally separated into three categories according to the taxonomy of Han, Huang, Song, Yang, Wang, and Wang (2021) (Figure 2.2). **Sample-wise dynamic networks** represent a family of data-driven approaches for handling different inputs, focusing on increasing their efficiency. This can be achieved by either a) allocating a varying amount of computation for each sample, essentially reducing redundant computation or b) by adapting network parameters to each input sample while maintaining the same computation graph, increasing the representation power with a small increase of computational cost.

Methods from the first category maintain a dynamic computation graph, which adapts conditioned on each sample. The basic principle is to save redundant computation when processing

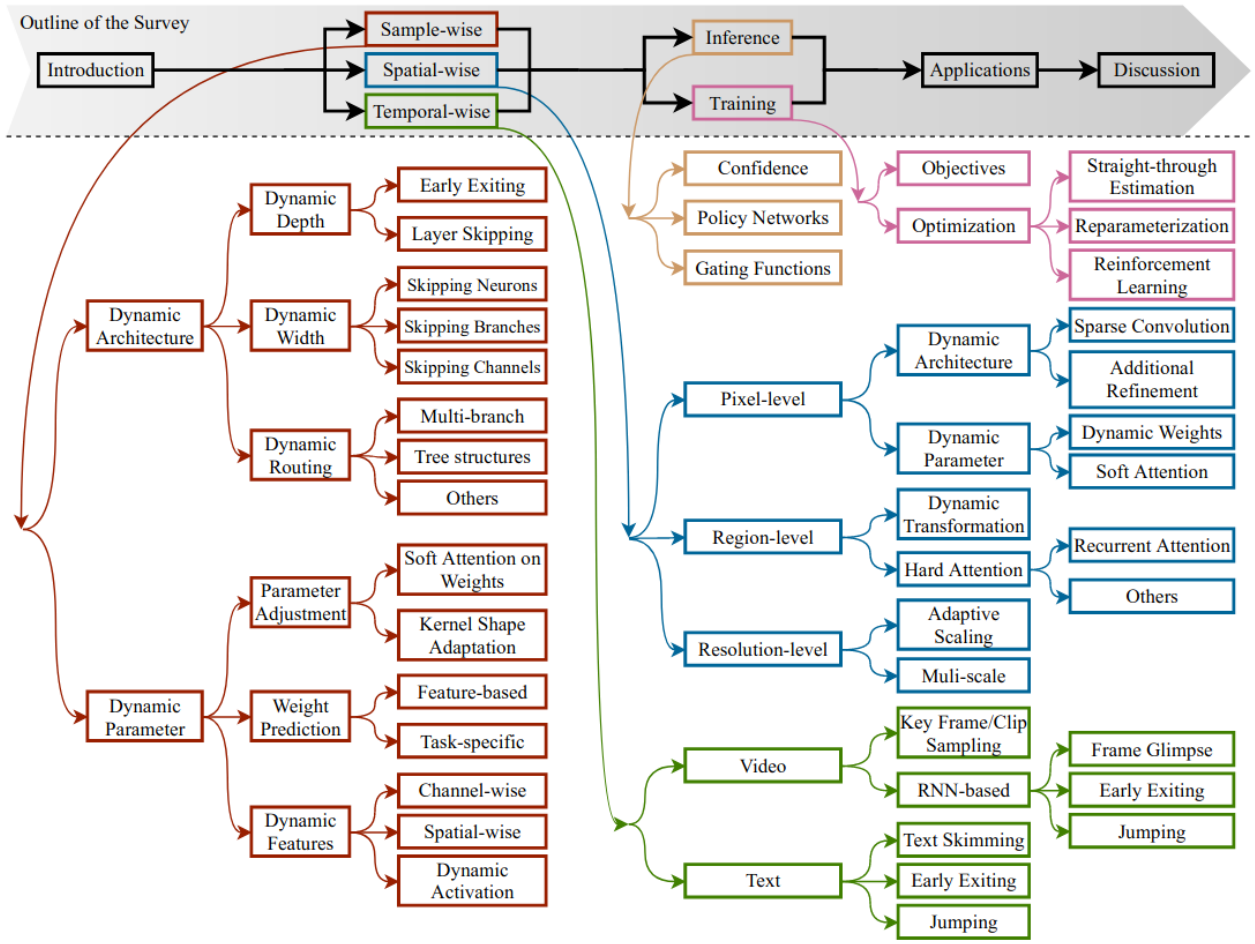


Figure 2.2: Figure presenting a categorization of the dynamic networks according to the type of dynamic behavior they entail (sample-wise, spatial-wise and temporal-wise). Figure reprinted from Han, Huang, Song, Yang, Wang, and Wang (2021).

canonical ("easy") samples while also preserving their representation power for non-canonical "hard") samples. The computation graph can be altered by adapting the depth of the network, where early exiting and layer skipping methods either halt computation at a specific stage (Teerapittayanon et al., 2016), (Passalis et al., 2020), (Z. Lin et al., 2023), (Y. Li et al., 2023), or completely skip specific parts of the network (Graves, 2016), if sufficient accuracy can be achieved for a given sample. Alternatively, in cases of models with multiple paths or branches, using a dynamic routing approach, one of multiple candidate modules at each layer cares selectively executed (Cai et al., 2021), (Y. Li et al., 2020).

While the aforementioned methods attempt to increase efficiency by altering the computation graph to reduce the computations when possible, another line of work adapts network parameters to different inputs while keeping the architectures fixed, aiming for favorable efficiency

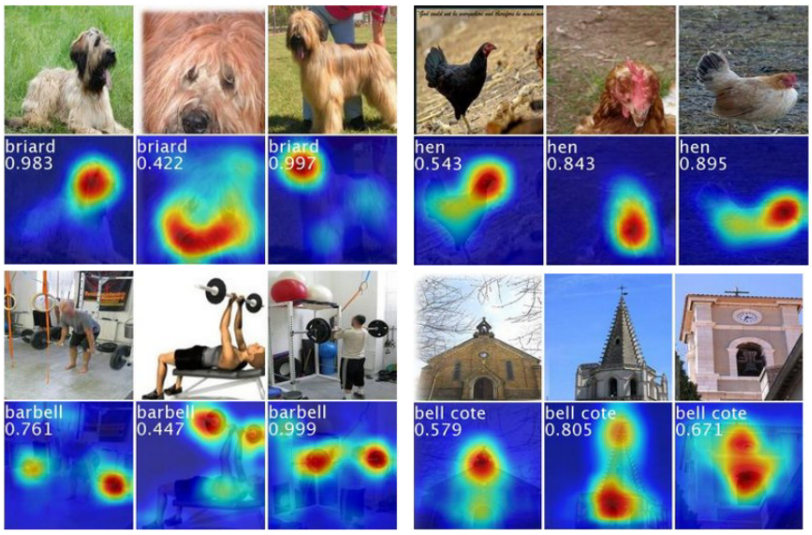


Figure 2.3: Visualization of feature maps, which highlight the discriminative image regions for classification, e.g. the head of the animal for briard and hen, the plates in barbell, and the bell in bell cote. Figure reprinted from Zhou et al. (2016).

trade-off by improving the performance. More specifically, to achieve that, they alter the static layer parameters conditioned on the input sample, while the computation remains the same, (Hu et al., 2018), (Hou et al., 2021). The most prominent approaches of dynamic parameter adaptation either adjust the trained parameters based on the input, or use soft attention mechanisms to rescale the features.

Unlike traditional approaches that treat all regions equally, **spatially adaptive dynamic networks** dynamically allocate computational resources on different image regions, based on their significance. This way, the issue of computation redundancy in image processing is addressed by leveraging evidence suggesting that not all locations contribute equally to the final prediction of CNNs Zhou et al. (2016) and Transformers Selvaraju et al. (2017), Abnar and Zuidema (2020), Chefer et al. (2021). Figure 2.3 presents visual evidence of CNNs focusing on specific image parts for classifying images. Spatially adaptive approaches typically involve an estimation step which produces the importance of performing computations on various spatial regions. According to the granularity of the spatial categorization methods operate in either pixel or block level. Finally, after identifying which pixels or regions are crucial and which ones are less significant, two different methods of adjusting computations based on spatial importance are implemented. In areas deemed important, computations proceed as usual. In less

important areas, computations might be skipped altogether, or they might be replaced with a lighter form of computation.

Similar to spatial dimension, adaptive computation can extend to the **temporal** dimension of sequential data, containing either texts or videos. Broadly speaking, enhancing network efficiency involves dynamically allocating reduced or no computation to inputs at less crucial temporal locations. Just as spatially adaptive methods allocate varying levels of computational resources to different regions of an image based on their importance, temporal adaptive methods dynamically adjust computation based on the significance of different time steps or frames.

In conclusion, a brief overview of various types of dynamic networks was provided. In the upcoming section, focus will be directed towards a more in-depth exploration of spatially adaptive dynamic networks, shedding light on their mechanisms and applications.

2.4 Spatially Adaptive Computation

2.4.1 Introduction

Spatially adaptive dynamic methods are based on the principle of performing different computations on various regions according to their importance. A core element of spatially adaptive dynamic methods is establishing a rule to distinguish important and and unimportant regions. This decision is often formulated in a binary mask which encodes importance score. The granularity of the binary mask dictates how finely the dynamic computation is enforced among the input. Without loss of generality, given an input feature map $X \in \mathbb{R}^{H \times W \times C}$, computation of the output feature map $Y \in \mathbb{R}^{H \times W \times C}$ is given by

$$Y = F(X), \tag{2.3}$$

where F denotes a computation block, typically involving combinations of one or more convolutional or fully connected layers, normalization and activation functions. In a dense computation

Method	Important (F_1)	Unimportant (F_2)
Skipping/Sparsity	$F_1 = F(X)$	$F_2 = 0$
Adaptive	$F_1 = F(X)$	$F_2 = \tilde{F}(X)$, with $\text{FLOPs}_{\tilde{F}(X)} \ll \text{FLOPs}_{F(X)}$
Low-Resolution	$F_1 = F(X)$	$F_2 = \uparrow (F(\downarrow (X)))$
Interpolation	$F_1 = F(X)$	$F_2 = Interpolation(X, Y_1)$, with $Y_1 = F_1(X)$

Table 2.1: Table presenting the most common spatially adaptive computation schemes. F_1 denotes the computation function for important and F_2 for unimportant F_2 regions. Arrows represent upsampling and downsampling operations.

case, the output is the result of computing F over all input pixels. However, in the spatially adaptive case, the following process is followed. First, a decision function for the importance of input pixels is required, which is often formulated as a binary mask $M : \mathbb{R}^{H \times W} \rightarrow \{0, 1\}$:

$$M(X_m) = \begin{cases} 1, & \text{if } \phi(X_m) \geq s \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

Here, $\phi(x)$ is a function capturing the rule to classify important and unimportant regions, s sets the threshold for this decision and X_m is the input to mask estimation (often $X = X_m$). Then, the computation is defined as a two-part function, conditioned on the guidance of a decision $M \in \mathbb{R}^{H \times W}$:

$$F(x) = \begin{cases} F_1(x), & \text{if } M(X_m) = 1 \\ F_2(x), & \text{if } M(X_m) = 0 \end{cases} \tag{2.5}$$

with F_1, F_2 denoting two different computation functions for processing important and unimportant regions respectively. In most cases, the computation function for important regions is set to $(F_1 = F)$, whereas computation for unimportant regions involves either skipping computations altogether, employing lightweight operations, or processing in low-resolution for the unimportant regions, as illustrated in Table 2.1.

This two-branch spatially adaptive computation scheme is relevant across a spectrum of approaches, spanning from earlier methodologies to the most recent ones. A brief overview of key innovations within this field over time are presented in Figure 2.4, primarily revolving around

a) mask calculation and training, b) optimization objectives and c) target architectures and computer vision tasks. Next, the fundamental and most influential works revolving around spatially adaptive computation are presented, focusing on their key elements and novelties. A comparison of key elements of spatially adaptive methods is presented in the end of this Chapter, and reader is strongly encouraged to refer to it for a comprehensive overview and better contextual understanding.

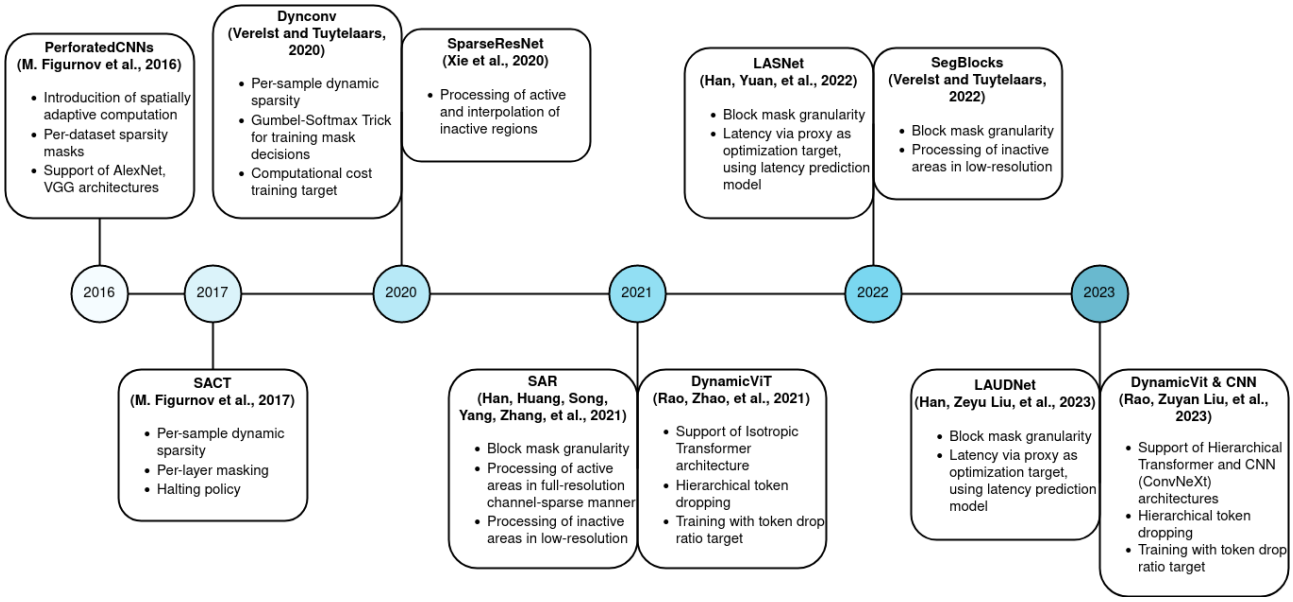
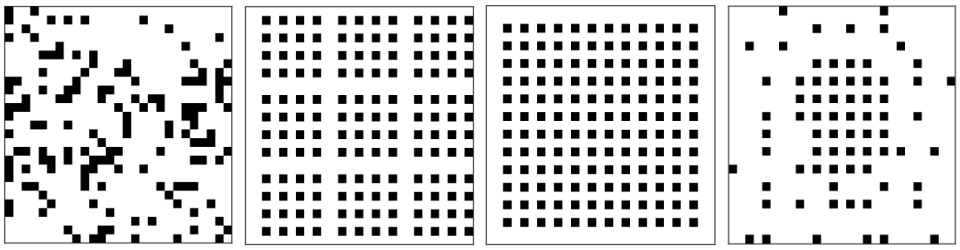


Figure 2.4: Figure presenting the evolution of spatially adaptive methods over time. Each method is followed with the key contributions or novelties it proposed. Per-layer masking and the Gumbel-Softmax trick Jang et al. (2016) training technique have become standard practice since their first introduction in 2017 and 2020 respectively, and are omitted from more recent methods. Most works experiment with residual-style networks such as ResNet He et al. (2016), therefore mentioning support for this architecture is also omitted.

2.4.2 Static Sparsity Patterns

The early work of Figurnov et al. (2016) proposed an approach to reduce spatial redundancy in convolutional layers by avoiding computations of convolution operation in some of the spatial positions. To achieve that, they propose to convert the regular convolutional layers to a perforated convolutional layer. During inference, in a small fraction of spatial positions, the outputs of the proposed layer are equal to the outputs of a regular convolutional layer, whereas the remaining values are interpolated using the nearest neighbor from this set of positions. The



(a) Random uniform (b) Grid (c) Pooling (d) Impact mask

Figure 2.5: Perforation binary patterns proposed in Figurnov et al. (2016).

locations on which convolution is performed are encoded in a binary perforation mask. They propose several ways of generating the perforation masks (Figure 2.5); uniform-random selection, grid or pooling pattern and the more sophisticated impact mask. The latter estimates the impact of perforation of each position on the CNN loss function over the whole dataset, and then removes the least important positions. The averaged impact tends to be higher in the center since images in ImageNet (J. Deng et al., 2009) dataset usually contain a centered object. It is worth noting that all the proposed perforation patterns are either globally fixed or predefined on a dataset, and do not explore input-dependent perforation masks. The perforation technique can achieve practical speedups close to theoretical ones on both CPU and GPU, while requiring only a minor modification of the convolution layer and without changing the architecture of a CNN.

However, the effectiveness of this optimization method compared in recent deep models implemented in modern hardware and software is unknown, and is projected to be less successful due to the following factors. First, Figurnov et al. (2016) experimented on hardware (Intel 4^{th} Gen CPU and NVIDIA Geforce series 9 GPU) and software libraries (Caffe (Jia et al., 2014) and MatConvNet (Vedaldi & Lenc, 2015)) that were released around 10 years ago. Since then, a lot of effort has been dedicated to optimizing popular CNN operations and especially the convolution, resulting in the release of more efficient implementations (cuDNN for GPUs), which are integrated in widely used deep learning frameworks such as Pytorch (Paszke et al., 2019) and Tensorflow (Martín Abadi et al., 2015). Therefore, the baselines for comparison has become relatively faster than those of previous years, introducing a greater challenge in improving practical execution time efficiency of spatially adaptive inference over baselines in modern hardware

and software. Secondly, authors experiment with networks such as AlexNet (Krizhevsky et al., 2012) and VGG-16 (Simonyan & Zisserman, 2015) which were released a decade ago. Those networks are far less optimized, efficient and compact compared to more modern networks (like ResNets (He et al., 2016), ResNeXts (S. Xie et al., 2017) and ConvNeXts (Z. Liu et al., 2022)), potentially including a higher percentage of redundant computations. Thus, increasing the percentage of computation sparsity would lead to a quicker degradation of performance in modern networks compared to AlexNet and VGG. Finally, reliance on static skipping patterns can limit accuracy of the spatially adaptive models. This limited flexibility of sparsity patterns is a key direction of spatially adaptive computation research and is addressed in following work. Therefore, it is highly likely that the relative trade-offs achieved by Figurnov et al. (2016) would not be transferable to the same extent on recent models using modern hardware and software libraries.

2.4.3 Dynamic Sparsity Patterns

Towards adapting the computation to each sample, M. Figurnov et al. (2017) propose a method to dynamically control the number of executed layers for various image regions. Their work is inspired by the Adaptive Computation Time (ACT) (Graves, 2016) mechanism and adds a spatial dimension to it. In ACT, a halting score is estimated for each layer in a network as a scalar value in the range [0, 1]. As soon as the cumulative sum of the halting score reaches one, all following residual units in this block are skipped, similar to an early-exit strategy. Based on that, they introduce Spatially Adaptive Computation Time (SACT), adjusting the per-position amount of computation by applying ACT to each spatial position of the block. They essentially introduce more granularity to the original ACT method, adding an additional (spatial) dimension. Computation is performed only on active regions, which are defined as the spatial locations where the cumulative halting score is less than one, whereas computations in inactive regions is skipped. The evaluation of a block can be stopped completely as soon as all the positions become inactive, similar to the original ACT method. The halting score $H^l(X)$

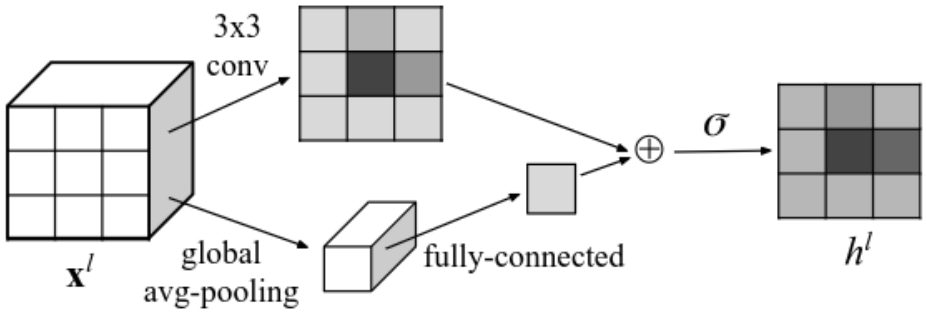


Figure 2.6: Halting score calculation module from M. Figurnov et al. (2017).

for layer l is calculated by

$$H^l(X) = \sigma(W_c^l * X + W^l avgpool(X) + b^l), \tag{2.6}$$

where $*$ is a 3x3 convolution operation with 1 output channel, avgpool denotes a global average pooling operation and $\sigma(t) = \frac{1}{1+e^{-t}}$, as seen in Figure 2.6.

Verelst and Tuytelaars (2020) proposed a method to dynamically apply convolutions conditioned on the input image, by introducing a computational block where a small gating branch learns which spatial positions should be evaluated. Similar to previous works, the output of the gating branch is a binary mask that indicates the areas where the convolution operation will be applied and where it will be skipped. To compute that, the module from M. Figurnov et al. (2017) is used for classification tasks, whereas a 1x1 2D convolution is used for pose estimation. However, the output of the mask estimation module is in continuous float format and must be converted to a binary mask of discrete values to represent a decision. Performing that conversion using the *argmax* function is plausible during inference, however the *argmax* function is not a differentiable operation. To generate binary masks during training in a differentiable process, the Gumbel-Softmax trick (Jang et al., 2016) is used, allowing end-to-end training. More information on this process is given in Section 4.3.3. Training is performed by providing a target computational budget $\theta \in [0, 1]$, which indicates the ratio of computations that should be used to the total possible computations. To achieve that, a sparsity loss is minimized, taking the squared difference of the total number of computations for each combination of mask decisions and the provided computational budget.

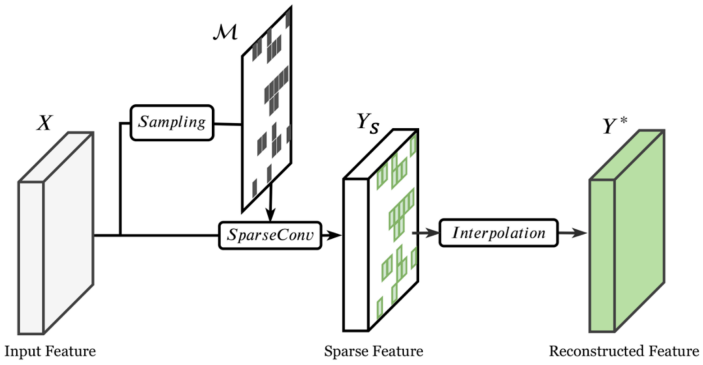


Figure 2.7: Guided by a sparse sampling mask M based on the input feature map X, features Y_s at active points are calculated. Features at the rest of locations are interpolated by the interpolation module C leading to the output feature map Y^*. Figure reprinted from Z. Xie et al. (2020).

The work of Verelst and Tuytelaars (2020) shares many similarities with M. Figurnov et al. (2017), however there are some key differences. First, computation conditioning is introduced in every layer independently rather than defining a spatial halting policy for a group of layers within a block. Moreover, training is performed by providing a target computation budget ratio, whereas the latter uses a set of ponder penalties τ, a less intuitive quantity with respect to the target model computational cost. In conclusion, Verelst and Tuytelaars (2020) facilitated a spatially adaptive process for training a residual-style networks and allow a performance to computational cost trade-off, by providing a computational cost budget as target.

In a similar fashion, Z. Xie et al. (2020) attempt to eliminate redundancy by computing features only at sparsely sampled locations, which are probabilistically chosen according to activation responses and then densely reconstruct the feature map with an efficient interpolation procedure. To achieve that, the predicted activation map is considered as a probability field and a sparse set of locations based on their probabilities are stochastically sampled, with the features at these samples interpolated to reconstruct the rest of the feature map. Given an input feature map $X \in \mathbb{R}^{C\times H\times W}$ with elements $x \in \mathbb{R}^C$, and a convolution function F, the output feature map $Y^* \in \mathbb{R}^{C\times H\times W}$ is calculated in two steps. First, based on the binary mask $M \in \mathbb{R}^{H\times W}$,

the output $Y_s \in \mathbb{R}^{C\times H\times W}$ is calculated as:

$$Y_s(x_i) = \begin{cases} F(x_i), & \text{if } M(x_i) = 1 \\ 0, & \text{if } M(x_i) = 0 \end{cases} \tag{2.7}$$

Then, the features of inactive elements are constructed by using an interpolation module C, leading to the final expression of the output feature map (Figure 2.7),

$$Y^*(x_i) = \begin{cases} F(x_i), & \text{if } M(x_i) = 1 \\ C(Y_s(x_i)), & \text{if } M(x_i) = 0 \end{cases} \tag{2.8}$$

This departs from previous work, since instead exhausting all the computation on areas estimated to have the most impact, smaller amount of computation is allocated to the rest of the areas, constituting more concise data representation during execution. The Gumbel-Softmax (Jang et al., 2016) trick is again used to enable differentiation for training a content-aware stochastic sampling module, responsible for identifying sparse points for interpolation.

The approaches presented in this section replace static skipping patterns with data-driven and learnable rules to make them more flexible, further unlocking the potential of spatially adaptive methods. The approaches presented in this section replace static skipping patterns with data-driven and learnable rules. This allows skipping patterns to adapt to each input sample, changing according to the input content. This increased flexibility can potentially identify larger amount of redundancy, thereby further unlocking the potential of spatially adaptive methods.

2.4.4 Grouping Image Regions

The methods introduced thus far perform spatially adaptive inference at the finest level, dynamically determining what type of computation should be performed at each input element. To estimate the important elements while maintaining a small computational cost, researchers focused on designing sparsity prediction modules which involve minimal computation. While

this strategy is effective when considering FLOPs as a measure of computational cost, it often falls short in accurately gauging the latency of dynamic methods, particularly in multi-core architectures like GPUs. The reason is that spatially adaptive computation introduces unstructured data accessing patterns (non-contiguous memory access) which are not well-optimized in popular deep learning framework implementations (Z. Wang, 2020), (Zhu & Xie, 2020), (Guo et al., 2020).

To address that, researchers resorted to performing the selection of important regions at a coarser level, essentially grouping elements into regions/blocks. This way, some amount of local structure is introduced in data, leading to more contiguous memory accesses which is essential for achieving competitive latency reductions in multi-core architectures. While this approach may be initially seem as a compromise to performance, studies (Han et al., 2022), [P2] have shown that this grouping leads to a beneficial regularizing effect in training deep neural networks. A potential reason is the large reduction in complexity of mask estimation, given the redundancy of information of images in the spatial domain. For those reasons, the block strategy is adopted in most recent works.

Employing block granularity for spatially adaptive computation is effective for mitigating the incurred overheads of data transformation, however latency overheads are still not captured when performing optimization in the FLOPs domain. To bridge the gap between FLOPs and practical efficiency, Han et al. (2022) propose a latency-aware spatial-wise dynamic network (LASNet), which performs spatially adaptive computation using the guidance of a latency prediction model. This model estimates the inference latency of dynamic networks by simultaneously considering algorithms, scheduling strategies, and hardware properties of target devices. The authors also introduce a custom algorithm for sparse computation, whose latency is predicted by their latency prediction model. Guided by the latency prediction model, latency can be mapped to density similar to how number of computations is mapped to density in (Verelst & Tuytelaars, 2020) and a similar optimization framework can be used. In fact, (Han et al., 2022) is essentially a variant of (Verelst & Tuytelaars, 2020), using block instead of pixel level granularity for spatial sparsity. In their following work, Han et al. (2023) extend their approach to also support two additional dynamic paradigms; dynamic layer skipping and channel

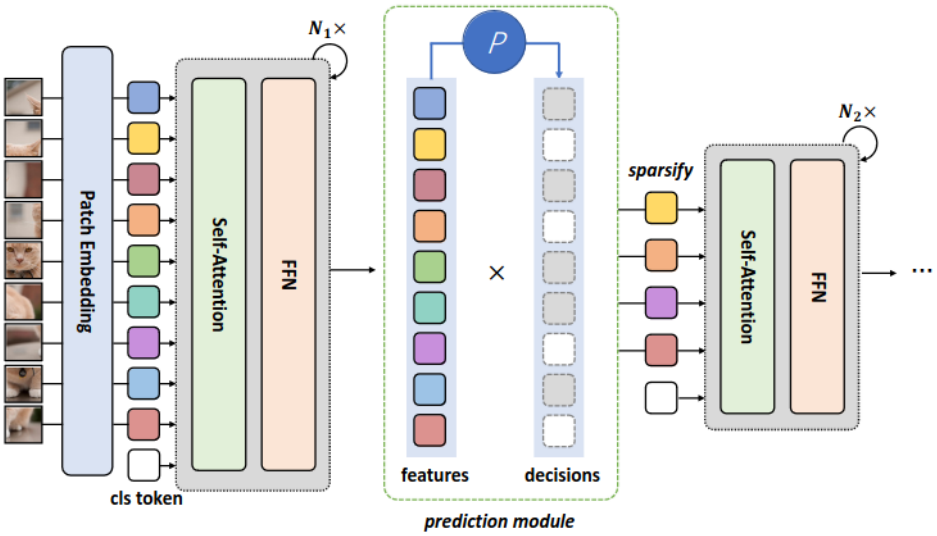


Figure 2.8: The spatially adaptive computation scheme of Rao et al. (2021) and Rao et al. (2023): Guided by a binary mask decision, tokens are dropped and self-attention is calculated only between the remaining tokens. Figure reprinted from Rao et al. (2021).

skipping.

The emergence of Transformer (Dosovitskiy et al., 2020) architectures for vision tasks, which often come with large computational costs, inspired researchers to try spatially adaptive methods to increase their efficiency. Transformers rely on the self-attention mechanism which is inherently sparse, since it relates various image regions with others. Rao et al. (2021) observed that the final prediction in vision transformers is only based on a subset of most informative tokens, which suffices for accurate image recognition. Based on that, they propose a dynamic token sparsification framework to prune redundant tokens progressively and dynamically based on the input (Figure 2.8). To achieve that, a selection module is added to stages of the network to prune uninformative tokens in a hierarchical manner. A lightweight architecture is used for the selection module, to estimate the importance of tokens using the input feature map. Similar to previous work, the Gumbel-Softmax (Jang et al., 2016) is employed to train the selection mechanism and enable end to end training. Even if in spatially adaptive computation breaks dense data consistency and structure in CNNs, using data conversion mechanisms, the overall dense structure in dense format is maintained during execution. This needs to be considered in Transformer architecture, where tokens can't be directly dropped nor zeroed out, since gradients are needed for back-propagation on them and zeroing out affects the calculation

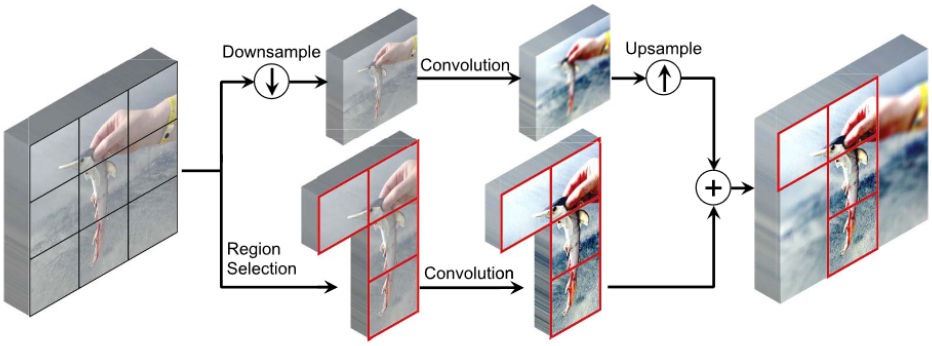


Figure 2.9: The spatially adaptive compuation scheme proposed in Han, Huang, Song, Yang, Zhang, and Jiang (2021), where a base (top) branch processes features at a lower resolution and a refinement (bottom) branch refines the important regions at full resolution. Figure reprinted from Han, Huang, Song, Yang, Zhang, and Jiang (2021).

of attention matrix. Thus, an attention masking strategy is proposed by authors, essentially eliminating the connection from abandoned tokens to all other tokens in the attention matrix based on the binary selection mask. During inference however, Transformers are a more "natural fit" for sparsity and present an advantage compared to CNNs with respect to latency gains. The reason is that tokens can be directly dropped during execution without requiring to maintain complete structure. Therefore, there is no need to use specialized and efficient sparse implementations to achieve realistic gains as in CNNs.

In their extended journal work, Rao et al. (2023) extend the previous work beyond isotropic, to hierarchical vision Transformer architectures and CNNs. The key contribution is the idea of asymmetric computation instead of completely dropping tokens or skipping image areas. This is implemented by applying lightweight fast paths to less informative features and expressive, more computationally intensive slow paths to important locations, maintaining the complete structure of feature maps during execution. The effectiveness of spatially adaptive computation is investigated through extensive experiments on modern architectures and different visual tasks (semantic segmentation and object detection).

Han, Huang, Song, Yang, Zhang, and Jiang (2021) propose a novel Spatially Adaptive feature Refinement (SAR) method, which introduces a two-branch approach to sparse computation. One branch conducts standard convolution on input features at a lower spatial resolution, and the other one selectively refines a set of regions at the original resolution, as seen in Figure

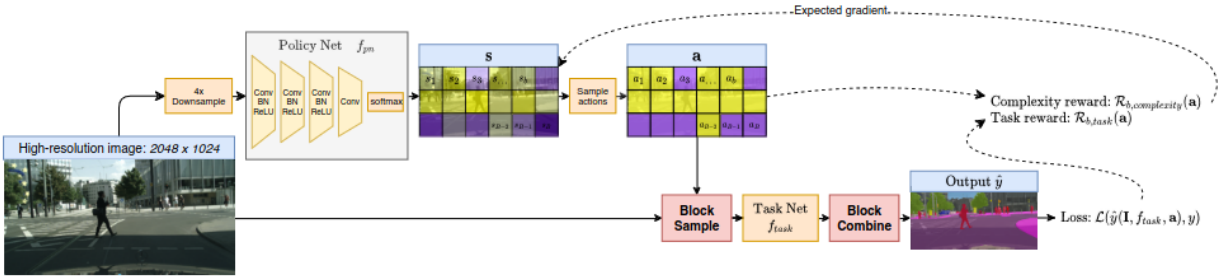


Figure 2.10: The framework proposed in Verelst and Tuytelaars (2023). A policy network predicts which blocks are going to be executed in full resolution. Training is performed by calculating a reward for each block, conditioned on the task loss and by supervising the ratio between high and low resolution blocks. Figure reprinted from Verelst and Tuytelaars (2023).

2.9. The refinement branch is further split into sub-branches along the channel dimension, assigning different groups to various image regions, introducing more flexibility. Focusing on practical efficiency, pixels are grouped in regions, leading to binary decision masks with block granularity.

Based on the ability of CNNs to perceive objects at various scales and the existence of inherent spatial redundancy of information in high-resolution images, Verelst and Tuytelaars (2023) propose a spatially adaptive approach for the semantic segmentation task. The computational cost of existing CNNs is reduced, by dynamically adjusting the processing resolution of image regions based on their complexity. More specifically, the method processes image blocks in either high or low resolution, by splitting an image into blocks, and then downsamples non-important blocks. The same operations are used for processing inputs at both resolutions To minimize data transformation overheads from dense to sparse and sparse to dense domains, two parallel branches with low and high resolution are maintained during execution. Using standard padding operations often leads to discontinuities and border artifacts. Hence, to ensure spatial consistency, authors propose a custom padding module, which mitigates these issues by copying features from neighboring blocks into the padding border, preserving feature propagation between blocks.

To select which blocks should be processed at each resolution level, a lightweight policy network is trained using reinforcement learning. It outputs probabilities per block, which indicate the likelihood that the block should be processed in high resolution based on the input image.

Decision per block is considered an action which is assigned a reward, calculated from the task segmentation loss. The task reward encourages the policy network to select regions with a high segmentation loss for high resolution processing, while the ratio of high to low resolution blocks is used to constrain the number of high resolution blocks to a desired number. The whole process is depicted in Figure 2.10.

2.5 Conclusions and Limitations

In conclusion, the exploration of spatially adaptive dynamic methods has shed light on key methods that contributed significantly to the field. These methods have been presented emphasizing in their novelties, which distinctly add value and innovation to the spatially adaptive computation domain. The findings are presented in a comprehensive table 2.2, summarizing the pivotal elements inherent in these sparse methods, providing a convenient reference for researchers and practitioners in the field.

This thesis explores spatially adaptive computation approaches focusing on CNNs. While conducting this research, Transformer architecture for vision has gained increased popularity due to the state-of-the-art results in vision tasks. However, despite this trend, CNNs remain the architecture of choice for time-sensitive applications and applications particularly focusing on resource-constrained hardware (X. Liu et al., 2023), (C. Chen et al., 2024). Moreover, Transformers employ a different computation pattern than CNNs, which allows for a more proportional relationship between FLOPs and practical latency on GPU devices. Therefore, Rao et al. (2021), Rao et al. (2023) which focus on reducing FLOPs, also achieve a corresponding latency reduction as byproduct, holding limited research value to investigate optimizing latency directly.

While there has been significant progress and important contributions proving the effectiveness of dynamic optimizations, it is essential to acknowledge certain limitations of methods of this field. Most known work use the number of FLOPs as a primary metric for gauging computational cost. While FLOPs have been widely accepted as a measure of computational

cost, it is crucial to recognize that this metric does not always provide an accurate reflection of latency. Therefore, it is useful to consider the actual latency in spatially adaptive computation, to provide a more holistic evaluation of the performance of dynamic optimization techniques. Moreover, a lack of consideration for the number of parameters is also evident. To enable practical dynamic behavior, multiple model weights need to be stored and loaded for inference, increasing the number of parameters linearly. This is an important consideration, since real-world devices often need to operate within finite storage constraints for model weights. The introduction of a systematic evaluation of the memory footprint, represented by the number of parameters, overlooks a crucial aspect of the overall efficiency and feasibility of these methods and should also be addressed.

In the following chapters, this thesis presents its contributions by systematically addressing key research questions related to spatially adaptive computation as a strategy to mitigate redundancy in CNNs.

Method	Sparse decisions	Computation Scheme	Mask granularity	Training Target	Target Architectures
PerforatedCNN (Figurnov et al., 2016)	N	Interpolate	Pixel	-	AlexNet (Krizhevsky et al., 2012) VGG (Simonyan & Zisserman, 2015)
SACT (M. Figurnov et al., 2017)	3	Skip/Sparse	Pixel	Halting target scores	ResNet (He et al., 2016)
DynConv (Verelst & Tuytelaars, 2020)	N	Skip/Sparse	Pixel	Computational budget	ResNet (He et al., 2016) MobileNet-v2 (Sandler et al., 2018)
Stochastic-Interp (Z. Xie et al., 2020)	N	Interpolate	Pixel	Gamma weight of sparse loss	ResNet (He et al., 2016)
DynamivViT & DynamicCNN & (Rao et al., 2021), (Rao et al., 2023)	3	Adaptive	Pixel/Token	Hierarchical Pruning ratios	LV-ViT (Jiang et al., 2021) DeiT (Touvron et al., 2021) Swin (Z. Liu et al., 2021) ConvNeXt (Z. Liu et al., 2022)
LASNet (Han et al., 2022) & LAUDNet (Han et al., 2023)	N	Skip/Sparse	Block	Latency ratio via proxy	ResNet (He et al., 2016) RegNet (Radosavovic et al., 2020)
SAR (Han, Huang, Song, Yang, Zhang, & Jiang, 2021)	N	Low-resolution	Block	Target ratio between refined and base blocks	ResNet (He et al., 2016) MobileNet-v2 (Sandler et al., 2018)
Segblocks (Verelst & Tuytelaars, 2023)	1	Low-resolution	Block	Target ratio between high and low resolution blocks	SwiftNet (Orsic et al., 2019)

Table 2.2: The discussed spatially adaptive dynamic methods of the background chapter, summarizing the relevant key elements.

Chapter 3

Data-Driven spatial sparsity in stereo-matching

3.1 Introduction

This chapter addresses the first research question: *How can we use data-driven spatially adaptive methods to increase the efficiency of a stereo-matching CNN?*. Part of the work described in this chapter is based on the conference paper: *Pushing the Efficiency of StereoNet: Exploiting Spatial Sparsity*, which was published and presented in the 2022 International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP) [P1].

3.1.1 Depth Estimation

3D reconstruction is a fundamental problem, within the field of computer vision. Its goal is to estimate the 3D geometry and structure of objects and scenes from one or multiple RGB images. This can be accomplished by methods which belong in two main categories, according to data acquisition process, namely *active* or *passive* methods. In *active* methods, projection of light in the form of a coded pattern is employed to aid the matching process, which is

particularly helpful in areas with insufficient lighting or texture variation. On the other hand, *passive* methods rely only the input of sensors, which capture the radiance emitted or reflected by objects in the scene, without introducing additional light sources. Thus, they are easier and cheaper to deploy, but often suffer from poor performance in areas with inadequate lighting.

A second distinction is between the number of perspectives from which the scene is viewed and/or illuminated (Moons et al., 2010). When the system components (sensors) are placed close to each other and can be considered as their views almost coincide, methods are considered as *single-vantage*, whereas when the capture sensors are placed with some distance between each other, methods are considered as *multi-vantage*. As a result, *single-vantage* methods do not suffer from multi-view occlusions as is the case with *multi-vantage ones*, however, the latter can provide more context and richer representation of the scene.

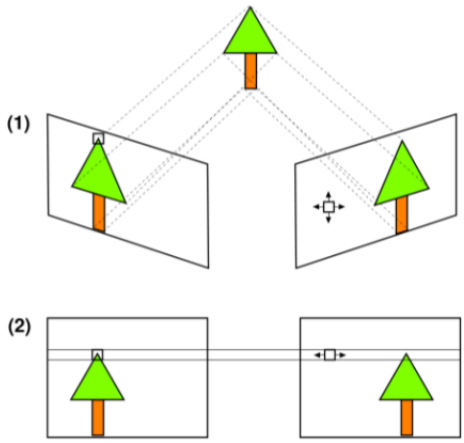


Figure 3.1: Stereo image rectification. Initial image pair before rectification (1) and after (2). Notice how the corresponding points belong in the same epipolar line. Figure reprinted from Domínguez-Morales et al. (2012).

3.1.2 Binocular Stereo Matching

The setup where two cameras, offset horizontally from each other are used to provide visual input, is called binocular stereo. This setup is inspired by human binocular vision, where each eye captures a slightly different view of the same scene. As a result, binocular stereo is a passive, multi-vantage method, and is widely adopted for 3D reconstruction applications.

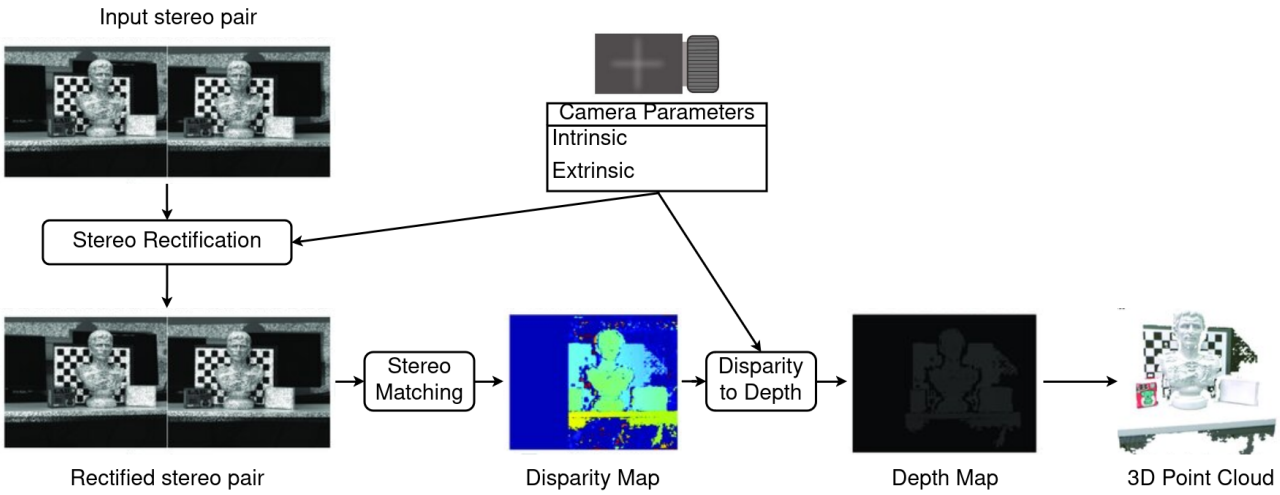


Figure 3.2: 3D reconstruction pipeline using stereo matching.

To calculate the depth of a scene given a stereo setup, pixels in the left image need to be matched with their corresponding pixels in the right image, resulting in a task called stereo matching. This initial 2D-correspondence problem can be reduced to a lower dimension by performing stereo image rectification, which uses the intrinsic (characteristics inherent to the camera, such as focal length and lens distortion) and extrinsic (describing the camera's position and orientation in space) camera parameters. By rectifying the stereo images based on these parameters, the problem is transformed into a 1-D correspondence along a specific line called the epipolar line, which represents the intersection of the image planes of the two cameras and simplifies the matching process by restricting the search space to a single dimension (Fig. 3.1). This is particularly useful for stereo matching methods, since the search space is reduced, resulting in smaller computational complexity. This correspondence is encoded as pixel distance or horizontal offset, and is stored into a data structure, where each pixel has value equal to its distance with the corresponding pixel in the other view. This image is called **disparity map** and its values are inversely proportional to scene depth.

The disparity map along with the corresponding parameters of the cameras of stereo setup, can be used to directly calculate the depth of the depicted scene. More specifically, considering two rectified cameras with parallel optic axes and a point in a scene matched x_l and x_r horizontal

coordinates in left and right image correspondingly, its depth Z can be calculated as

$$Z = \frac{b \cdot f}{d}, \tag{3.1}$$

where $d = x_r$ - x_l ($x_r > x_l$ by definition), b is the baseline distance between the cameras and f is the focal length of the sensor. The full stereo matching pipeline that is typically used in a real system is depicted in Figure 3.2. First, the input images are rectified using the camera parameters, resulting in the rectified pair, which is provided to the stereo matching algorithm to extract the disparity map. Finally, a depth map or 3D point cloud can be obtained from the disparity map, by converting disparity values to depth or 3D coordinates.

3.1.3 Prior Art and Current State of the Art

As previously stated, the stereo matching problem boils down to finding correspondences between pixels of an image pair (left, right) and encoding matching into a disparity map (as horizontal displacement). According to the taxonomy proposed by Scharstein et al. (2001), stereo matching algorithms typically perform the following four steps:

- Matching cost computation
- Cost (support) aggregation
- Disparity computation / optimization
- Disparity refinement

In the upcoming section, we will discuss the four critical steps presented above and explore the relevant prior and current state-of-the-art techniques and methodologies. Next, the most important datasets for evaluating stereo-matching algorithms will be presented, followed by an evaluation of performance of the presented methods.

Matching Costs

The initial stage in nearly every stereo matching approach involves computing costs for matching pixels from left to right images. Before 2015 and the rise of deep learning approaches, methods relied on handcrafted cost metrics to capture similarity between pixels in the form of template matching. **Absolute intensity differences (AD)** (Kanade et al., 1995), **Squared intensity differences (SD)** (Anandan, 1989), (Hannah, 1974), **Sum of squared differences (SSD)** measure the radiometric difference between the two patches in the RGB colorspace. They are the most basic measures and remain effective for the most "ideal" illumination scenarios. Extending the above metrics, in **Normalized Cross-Correlation (NCC)** (Hannah, 1974), (Gottesfeld, 1992) and **Zero-Mean Normalized Cross-Correlation (ZNCC)** (Gottesfeld, 1992) incorporate second order statistics, demonstrating 30-50% improved performance in matching compared to AD, SD and SSD, and 10-20% against NCC. The impact is especially marked in scenes with challenging illumination (Stoyanov et al., 2010), (Zampokas et al., 2018). This can lead to accuracy improvements of around 10-20% in disparity error rates on benchmark datasets, depending on the specific conditions. The aforementioned metrics constitute *parametric* measures of matching, incorporating of domain-specific knowledge and assumptions to achieve matching accuracy. However, the effectiveness of these methods often depends on the appropriateness of the chosen parameters and the underlying assumptions in different scenarios.

On the other hand, *non-parametric* measures of matching refer to techniques that do not rely on explicit mathematical models or predefined parameters to establish correspondences between pixels in stereo image pairs. Instead, they often employ data-driven approaches or algorithms that directly compare pixel values or features without assuming specific characteristics about the scene or imaging process. Zabih and Woodfill (1994) introduced a set of image transformations, which are calculated based on the relative ordering of color intensities within an area, and not on the intensity values themselves like parametric measures. The **Rank Transform** measures ordering of the local intensity, while **Census Transform** is a measure of local structure. The latter is extensively utilized, given its performance and low complexity (Hernandez-Juarez

et al., 2016).

To store the matching costs for each pixel pair across a range of possible disparity levels, a (matching) cost volume is constructed in the four-dimensional space. More formally, in the case of images with height and width dimensions H, W, dimension of similarity measure vector equal to C, and D disparity levels, the volume is represented as a $H \times W \times D \times C$ grid. Mathematically, the volume V can be defined as:

$$V(i, j, d) = G(d, x_l(i, j), x_r(i, j)), \tag{3.2}$$

where $V(i, j, d)$ represents the matching cost for the pixel pair at coordinates (i, j) in the left image and its corresponding candidate pixel at disparity level d in the right image and $G(d, x_l(i, j), x_r(i, j))$ is the matching cost function, which quantifies the dissimilarity between the corresponding pixels at the specified disparity level d.

Cost Aggregation/Regression

Multiple methods have been proposed to calculate disparity from any matching cost between pixels. One such method is the **Winner-takes-all (WTA)** approach, where the match with the highest cost is chosen for each pixel individually. However, this approach does not take into account global information, producing several outliers. To compensate that, **Cross-based cost aggregation** forms a neighborhood of similar intensities around each pixel, and averages the matching cost inside that region. Sun et al. (2002) proposed a solution for stereo matching using three coupled Markov random fields (MRF). They encapsulated penalties for smoothness, discontinuities and occlusions and the final MAP solution is found using **Belief Propagation**. In the seminal work of Hirschmüller (2008), a novel energy term is introduced, which includes matching costs and penalties for local and global discontinuities. The cost is calculated by aggregating one dimensional matching costs from 16 directions around each pixel, using a dynamic programming solution, called **Semi-Global Matching**.

Fundamental Deep Learning Approaches

By 2015 CNNs have revolutionized feature learning in computer vision by automatically extracting hierarchical and task-specific features from raw data. This marked a departure from traditional hand-crafted feature extraction methods, as CNNs excel at capturing intricate patterns and representations directly from the data, reducing the need for human-designed features. The first reported use of CNNs in the task of depth from stereo came from Žbontar and LeCun (2015), who introduced the concept of replacing manually engineered features with learnable ones. Their approach involved employing a siamese CNN architecture, which was trained end-to-end to learn features directly from stereo image pairs. Specifically, the network was trained to match 9×9 patches in a fully supervised manner. The output of convolutions is either compared using a cosine similarity measure (*fast* architecture) or entered in a sub-network of fully connected layers and a final sigmoid non-linear activation function, to produce a similarity score (*accurate* architecture). The similarity scores are further aggregated with either cross-based cost aggregation or semi-global matching, to produce the final disparity map.

On a similar path, Mayer et al. (2016) proposed encoder-decoder architectures for stereo matching which either concatenate left and right image (DispNetSimple) or produce separate feature representations for the two images which are aggregated using a custom correlation layer (DispNetCorr). In the same work, they also introduced FlyingThings3D, Monkaa and Driving datasets for stereo matching and optical flow estimation. Those datasets are currently referenced as scene-flow dataset (Mayer et al., 2016), which currently serves as a benchmark dataset for training and evaluation by almost all recent deep learning methods for stereo matching.

In an attempt to create an efficient approach, Pang et al. (2018), introduced a novel architecture called cascade residual learning, which generates disparity using a hierarchical approach and a separate disparity refinement step for each scale. Based on the previous work of Mayer et al. (2016) and Ilg et al. (2016), a version of DispNetC with extra up-convolution layers is used, to produce a full resolution disparity map. Then a separate network, DispResNet is responsible for the refinement of the disparity map. Considering the advantages of residual learning, He et al. (2016) attempt to estimate a residual, which is added to the disparity at a previous

scale, to produce the next disparity. Those residuals are estimated and supervised for every scale in the expanding part of the network. Since residual learning is less likely to overfit than supervising the whole signal, this approach offers improved performance in training schedules which include finetuning.

The previous approaches successfully leveraged the power of deep learning feature representations to achieve improved performance in stereo matching. However, they mostly focus on feature extraction, whereas the rest of the steps still relied on older aggregation methods. The reason is that the basic Winner-Takes-All aggregation is accomplished using the *argmax* function, which is non-differentiable. Therefore, it cannot be incorporated into an end-to-end training process with back-propagation. To achieve a fully end-to-end deep learning approach, Kendall et al. (2017) introduced a disparity regression in a differentiable manner, introducing the *softmin* function. After extracting deep features from the two images, the **4D cost volume** V with size $H \times W \times D \times C$ is formed and disparity is estimated as the weighted average between the normalized outputs using the $\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$ function and the corresponding disparity indices, as:

$$softmin = \sum_{d=0}^{D} d \times softmax(-V(i,j,d)) \tag{3.3}$$

This way, disparity regression is impacted by all values, yet disparities with lower costs have a greater influence on the final outcome. This makes *softmin* sensitive to multi-modal distributions, where the final weighted average may not accurately converge to the true minimum.

Towards improving disparity estimation in the ill-posed regions, Chang and Chen (2018) proposed an effective feature extraction and a 3D volume filtering methodology. Motivated by the fact that local features are not strong discriminators in textureless areas, they combined them with the global cues, to exploit global context information. This was achieved by utilizing Spatial Pooling Pyramid (SPP) (He et al., 2015) for the feature extraction step, which merges local and global cues by effectively enlarging the receptive fields. They also propose the use of 3D convolutions for cost volume regularization and adopt the disparity regression technique based on *softmin* (Kendall et al., 2017).

In conclusion, the advent of deep learning methodologies marked a significant advancement in

stereo matching beyond traditional approaches, since the first have consistently demonstrated superior performance, surpassing the state-of-the-art in disparity estimation tasks by demonstrating around 2 times smaller disparity errors (Žbontar & LeCun, 2015). Moreover, the formulation of end-to-end training approaches, where individual modules for stereo matching are mapped and integrated into deep learning architectures, has streamlined the development process and paved the way for future research.

Efficient Deep Learning Approaches

The use of CNNs for stereo matching has revolutionized the field by improving performance, outpacing traditional approaches by a large margin. However, their use introduces increased computational demands since deep features are calculated using multiple convolution operations, while an increased memory budget is also required to store the full matching cost volume. Therefore, multiple works have been proposed to reduce the required computational and memory costs, towards more efficient approaches.

Lu et al. (2018) tried to address the large memory demands of modern CNNs, arguing cost volume contains redundant information. Thus, they combine left and right features in a sparse manner to effectively compress the cost volume, leading to lower memory cost and faster execution.

Based on the success of Spatial Pooling Pyramid (SPP) (He et al., 2015) for feature aggregation Chang and Chen (2018), Chabra et al. (2019) proposed the use of Vortex Pooling method (C.-W. Xie et al., 2018) for feature extraction, which is reportedly more efficient than SPP, achieving a 1.5% higher mIoU on the PASCAL VOC 2012 validation set with comparable computational costs. Du et al. (2019) replaced SPP with an Atrous Multiscale module, by using ResNet layers with depth separable convolutions rather than regular ones. The resulting D-ResNet blocks result in a larger receptive field while the number of parameters remains small. Features extracted from D-ResNet blocks are then passed through a series of 3x3 dilated convolutions with increasing dilation factors ($[1, 2, 2, 4, 4, ..., k/2, k/2, k]$), called Atrous Multiscale (AM) modules.

The use of 3D convolutions for cost volume filtering introduced by Kendall et al. (2017) is effective but computationaly costly. Therefore, inspired from the Semi-Global Matching (SGM) method, F. Zhang et al. (2019) proposed a novel Semi-Global Aggregation Layer (SGA), as the differentiable implementation of SGM. Similarly, it performs aggregation from four different directions (left, right, up and down), leading to better performance in areas lacking texture or strong features, as is the case with the classic SGM. To address fine details in the disparity map, a Local Aggregation Layer (LGA) is also added to the final part of the network. According to their analysis, replacing the 3D convolution with the SGA layers leads to 100 times fewer Floating-point operations.

Duggal et al. (2019) proposes a method to prune the search space, based on the seminal work of Barnes et al. (2009), which argues that out of several random disparity samples, some are good guesses. Neighboring pixels also demonstrate coherency, so information can be propagated into neighboring areas, and as a result the search space can be reduced. Thus, they have implemented the PatchMatch algorithm into a fully differentiable recurrent neural network layer, where each step represents an iteration of the original algorithm. The search space can be reduced by almost 10 times while maintaining good accuracy.

Khamis et al. (2018) argue that such a coarse cost volume already contains all the useful information for stereo matching and at the same time has a large receptive field, leading to better performance in ill-posed regions. Therefore they opt for an aggresive downsampling scheme, leading to cost volume of 1/8 or 1/16 of the original image resolution. This volume is filtered with a series of 3D resblocks and an initial low resolution disparity map is estimated. Finally, an edge-aware disparity refinement is proposed by applying a series of residual blocks to the concatenated RGB and disparity image in full resolution. This refinement method is effective in bringing fine details into the coarse disparity map, but fails to recover details which are completely missing from the downsampled cost volume.

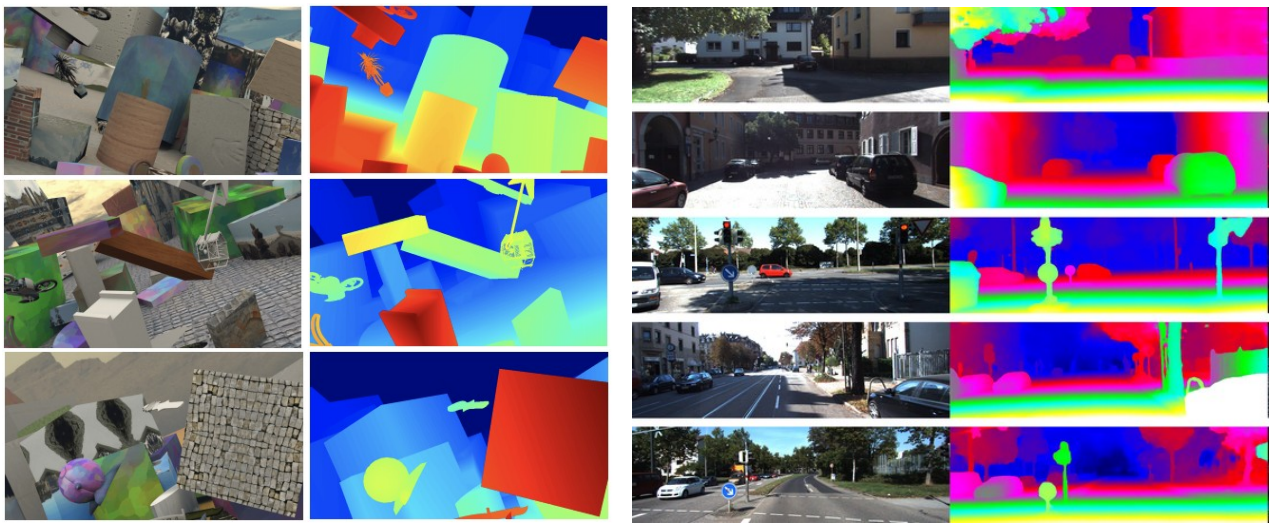
Y. Wang et al. (2019) propose a lightweight pyramidal model that extracts feature maps from the images and then builds a compact 4D volume by computing the L1 distance between left and (warped) right features. The model follows a coarse-to-fine architecture, while predicting

disparities at multiple stages. This provides the AnyNet model with an early-stopping functionality at anytime, i.e., inference can be shortened to obtain disparity at any stage, allowing for speed-accuracy trade-offs, which can be useful for real-world applications with limited resources. Chang et al. (2020) harness the efficiency of blueprint separable convolutions Haase and Amthor (2020) to build an attention-aware feature extraction method with heavily reduced computational cost. Furthermore, an attention-aware feature aggregation is also proposed to improve the representational capacity of features, while a cascaded 3D CNN architecture is further used to estimate disparity maps in multiple resolutions allowing the realization of an anytime prediction setting.

The exploration of efficient stereo matching techniques has yielded notable advancements, with various approaches that prioritize computational efficiency while acknowledging a trade-off with accuracy. Strategies such as sparse cost volume computation, alternative formulations of 3D convolutions, hierarchical cost volume processing, and early stopping methodologies have emerged as useful innovations. This collective effort underscores the balance between computational cost and accuracy, highlighting the success of these methods in significantly reducing computational cost while addressing the demands of real-world applications in computer vision.

Datasets

The construction of reliable and diverse datasets for stereo matching remains a challenging task, due to the problem of generating reliable disparity ground truths for real images. Real-world datasets rely on registering depth sensors such LIDAR or IR cameras for the provision of scene depth, which often requires a complex setup. This technique introduces multiple errors: (a) a registration error between cameras, (b) the measurement estimation error of the given sensor and (c) the potential difference between granularity or resolution between the stereo cameras and the depth estimation sensor. Therefore, researchers have turned to synthetically generated datasets, where not only ground truth can be calculated with exact accuracy, but also a large number of samples can be generated. However, while synthetic datasets can be much larger with precise ground truth disparities, they lack the realistic information that exists in real-



(a) Sceneflow (b) KITTI 2012 and 2015

Figure 3.3: Samples from (a) sceneflow Mayer et al. (2016) and (b) KITTI 2012 and 2015 Menze et al. (2015), Menze et al. (2018) datasets. Left RGB images are organized left column with their corresponding disparity maps in the right column.

world images. For this reason, a good practice typically involves training models on large and accurate synthetically generated data and finetune them on smaller real-world datasets. Next, some of the most widely used datasets for stereo matching will be introduced.

Sceneflow (Mayer et al., 2016) is a large synthetic dataset with 35,454 training and 4,370 test images of 960×540 resolution, suitable for optical flow and stereo matching problems. It consists of three subsets, Flyingthings3D, Driving, and Monkaa each containing dense and accurate ground-truth for training and testing. This dataset serves as a benchmark of performance for almost all deep learning based stereo matching methods.

KITTI 2012 (Menze et al., 2015) and **KITTI 2015** (Menze et al., 2018) are two real-world datasets collected from driving cars. KITTI 2012 provides 194 training and another 195 testing image pairs, while KITTI 2015 has 200 training and 200 testing image pairs. Each training image pair is accompanied with sparse ground-truth disparity from a LIDAR sensor.

Evaluation

In order to extract conclusions on the effectiveness of the presented methods, evaluation under the same base must be performed. End-point Error (EPE) is used as the metric to determine

accuracy in the stereo matching literature, which is the Euclidean distance between the predicted vector of disparities and the ground truth disparities. It is a preferred metric for stereo matching as it directly measures the accuracy of depth estimation at the pixel level, offering a widely accepted benchmark that is easily interpretable and insensitive to scale variations. All deep learning methods until today use the presented sceneflow (Mayer et al., 2016) dataset for training, therefore evaluation scores will be reported on the test set of sceneflow dataset.

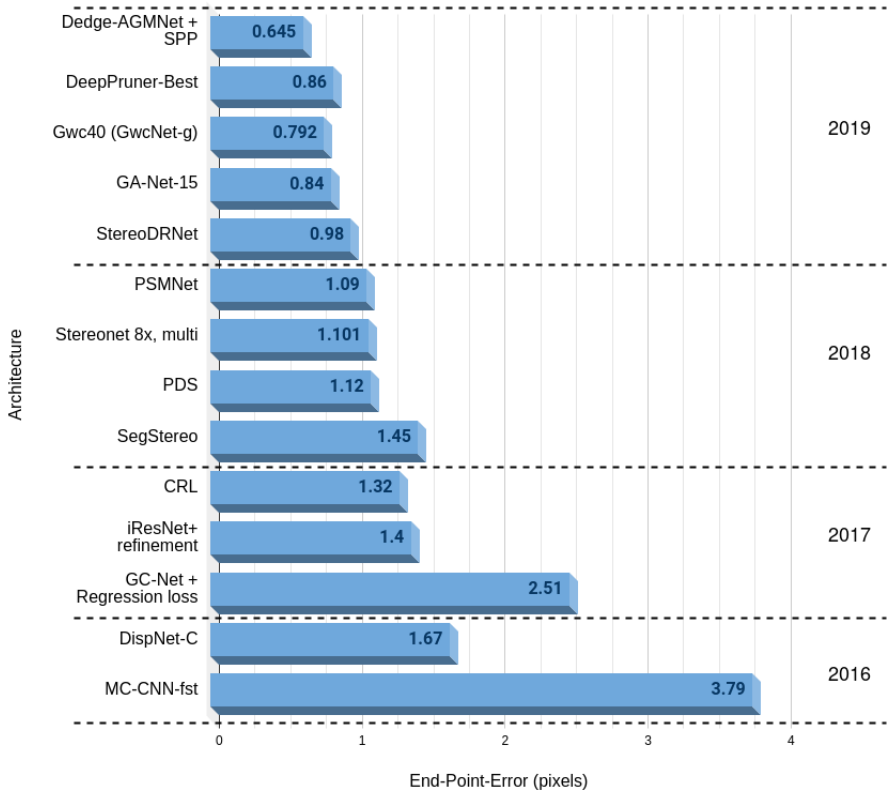


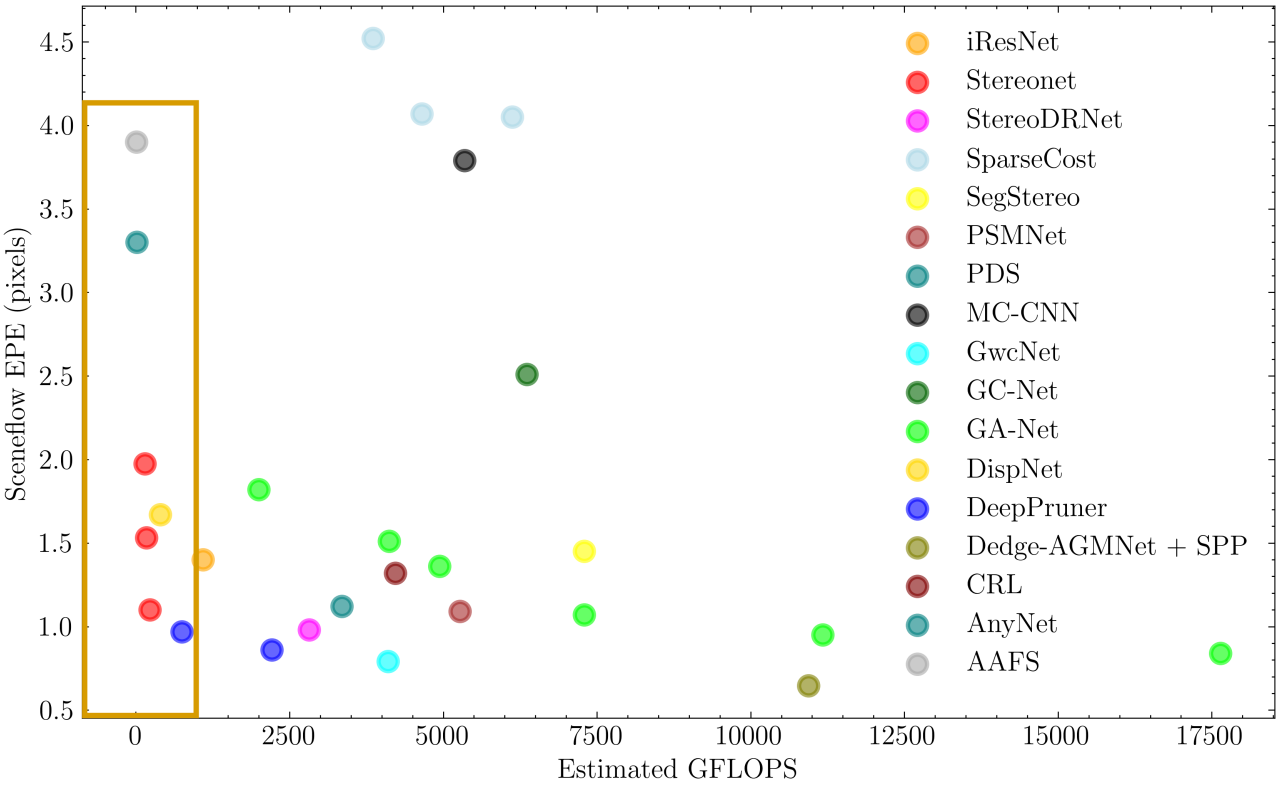
Figure 3.4: State of the Art EPE on sceneflow dataset, sorted from newer (top) to older (bottom).

GPU Model	Tesla P40	Titan XP	Titan X	1080 Ti	1080	1060 3GB
TFLOPS	11.76	12.15	6.69	11.34	8.87	3.93

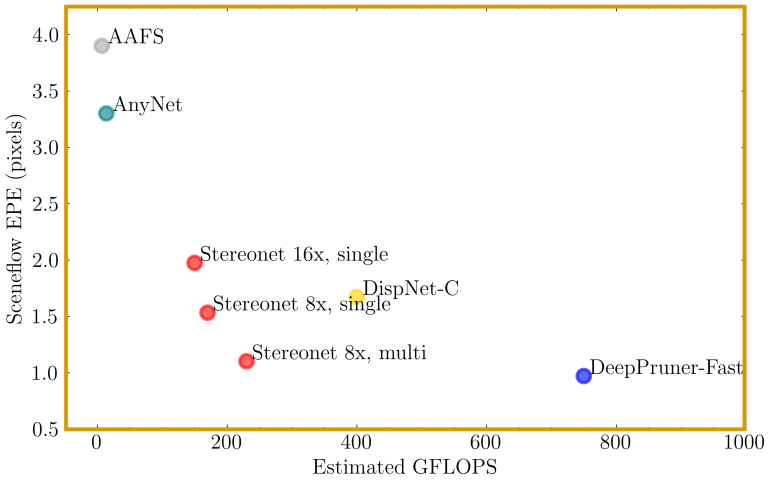
Table 3.1: The theoretic performance of the GPUs, used in the reported methods according to techpowerup.com.

In pure accuracy terms, deep learning methods have improved greatly since their emergence for stereo matching (MC-CNN (Žbontar & LeCun, 2015), 3.79 EPE) to recent approaches, which achieve EPE under to 1 pixel, demonstrating the effectiveness of more sophisticated feature extraction architectures (SPP, Vortex Pooling, MCUA). They provide more context around

each pixel, effectively enlarging the receptive field of convolutions of the network, which leads to more accurate matching even in textureless areas.



(a)



(b)

Figure 3.5: The state-of-the art landscape for stereo matching methods in terms of EPE vs normalized computational complexity on sceneflow test dataset as of 2022. Top (a) image includes all methods, whereas bottom (b) is limited to methods requiring less than 100 GFLOPs.

Accuracy is an important indicator on the general performance of an algorithm, however it often comes with increased computational or memory demands, which can prohibit or limit the deployment on real devices. Therefore, it is useful to examine each method for the trade-off

between the achieved accuracy and the required computational cost. In this analysis, focus will be given on the computational cost of methods, measured in FLOPs. However, rather than reporting complexity, a common practice in the stereo matching literature is to report execution time on a target hardware, which is heavily influenced by factors such as the target hardware capabilities and the software implementation. Thus, to establish a common base and to provide a fair comparison between older and more recent methods, a normalization is performed. Under the assumption that stereo matching methods are mostly computation bound, since convolutions are the main computational burden of the algorithms, the device computing capability (Table 3.1) is used to normalize the performance. The reported execution time is divided by the corresponding reported GPU computational power, which was used to produce the results. Thus, an estimation of the computational workload is calculated, reported in GFLOPS.

In terms of pure accuracy, deep learning-based techniques demonstrate state-of-the-art accuracy in popular stereo-matching datasets (KITTI (Menze et al., 2015), (Menze et al., 2018), sceneflow (Mayer et al., 2016)). As seen in Figure 3.5, there is a denser concentration of methods closer to the origin of the axes. This is expected, since the focus has been to reach minimal error, while keeping the number of computations low, representing maximal efficiency. *DeepPruner-Fast* and *Stereonet* variants, are able to achieve EPE close to 1 pixel, while requiring low number of FLOPs, putting them on the **efficiency pareto** front. *AnyNet* and *AAFS* also belong on the **efficiency pareto** of even lower FLOP methods, however, this comes with a significant reduction in accuracy with 3.3 and 3.9 EPE respectively. Lower EPE is demonstrated by a group of methods (*GwC-Net, StereoDRNet, DeepPruner-Best*) but this comes at orders of magnitude larger requirement of computations.

Contribution

According to the dynamic execution literature, there is evidence that models can be equipped with the ability to change the distribution of computations during inference, by selectively activating processing modules, based on the input. Motivated by that, in an attempt to create

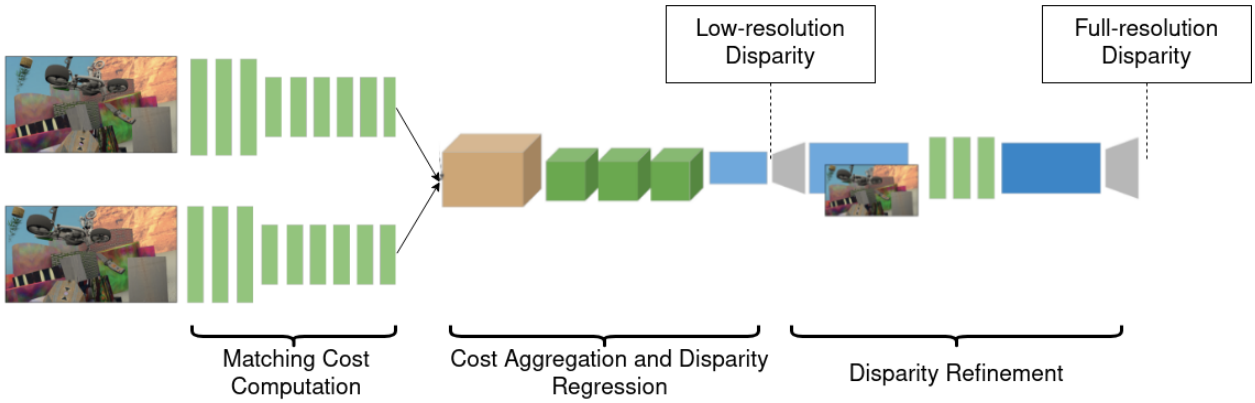


Figure 3.6: Figure presenting the StereoNet (Khamis et al., 2018) architecture, noting the typical steps of a stereo matching algorithm (bottom) according to taxonomy from Scharstein and Szeliski, 2002. Figure reprinted from Khamis et al. (2018) and augmented with additional information.

Network Part	GFLOPs
Cost Computation, Aggregation and Disparity Regression	54.42
Disparity Refinement	58.83
Full Network	113.25

Table 3.2: FLOPs analysis of StereoNet architecture for input image with size of 960×540 pixels.

more flexible and efficient stereo matching CNN design, a spatially sparse adaptation of a pretrained dense stereo matching CNN is proposed, which attempts to reduce redundancies in processing while maintaining accuracy close to the original dense network.

StereoNet Khamis et al. (2018), a state-of-the-art CNN in terms of accuracy to latency efficiency is selected as the target network. It is a lightweight stereo disparity estimation architecture that offers a reasonable trade-off between disparity error and computational cost, placing it near the top of the performance/cost pareto front, as seen in Figure 3.5. Operating on an already highly efficient network may limit the potential gains from our method, since redundancies are much less prevalent than other over-parameterized approaches. However, it provides more impactful results and conclusions, while also showcasing the challenges of using spatial sparsity optimization towards more realistic cases. The full architecture, presented in Figure 3.6, consists of a 2D feature extractor and 3D convolution blocks, which regress a low-resolution disparity map. Then this map is bilinearly upsampled to full resolution and further processed to produce a high-resolution disparity estimate. Low computational cost is achieved through aggressive

downsampling of the 4D cost volume, whereas the disparity refinement section is responsible for refining high frequency details in the final disparity map, accounting for over 60% of the total computations (Table 3.2).

3.2 Motivation

As most of the CNN-based stereo matching methods, StereoNet operates on a dense manner, performing the same computation on all pixels within an image. While this is helpful in capturing all the available information, potential computational redundancy could exist when equally processing all image pixels. StereoNet employs an edge-aware refinement approach, focusing on recovering high-frequency details which have been lost from multiple downsamplings of the input. Thus, we assume that when low-frequency areas are processed by the refinement module, they often produce small improvements, whereas areas with high-frequency details, such as edges, contribute more to the final output.

To investigate the varying impact of processing different regions on the final result, an experiment is conducted by selectively skipping computations on specific image regions during inference on sceneflow (Mayer et al., 2016) dataset, focusing on the refinement stage of StereoNet (Figure 3.6). This stage takes the low-resolution disparity map as input and uses a sequence of residual blocks to estimate a residual to add to it, to produce the full-resolution disparity map. To implement spatially adaptive processing for this stage during inference, a binary mask is multiplied with the estimated residual, essentially negating the refinement effect for some image regions.

First, the StereoNet-8x-Oneshot architecture employing a single disparity refinement step is trained according to Khamis et al. (2018) reaching 2.26 and 1.59 EPE for the low-resolution and full-resolution disparity estimation respectively. Given this pretrained network, various sparsity patterns are introduced on the refinement part during inference, to enable the spatially adaptive computation. More specifically, sparsity is introduced to the network gradually, in steps of 10% of total pixels resulting in 10 sparsity stages. Pixels for processing are selected

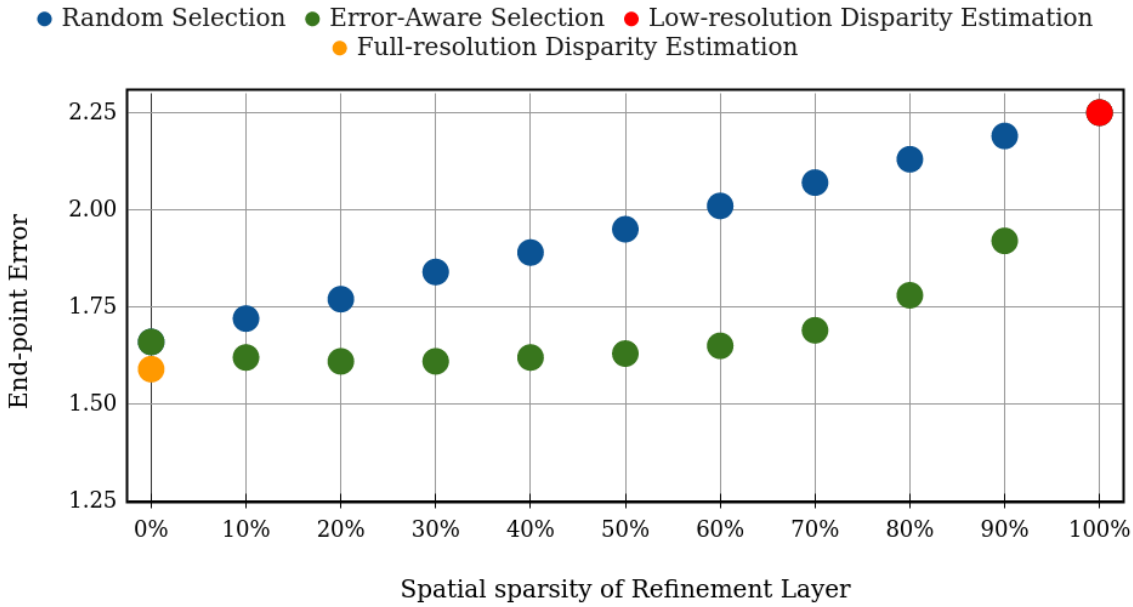


Figure 3.7: Figure demonstrating the change in EPE for the full sceneflow dataset with respect to sparsity using block format. Orange dots indicate network operation states that skip regions in an error-aware way, whereas blue dots indicate network operation states that skip picture regions at random.

with two different methods; in a random and in an error-aware manner. Using the latter, pixels are sorted according to the error reduction that refinement produces on them and the *top-k* are considered active for the various sparsity levels. In an attempt to reduce the dependency of performance from different hardware specifications and capabilities, the analysis focuses on FLOPs for describing the algorithm's computational complexity and provide a more rigorous metric than inference time measurements. Furthermore, convolutions over floating point tensors account for the majority of calculations, further justifying the use of FLOPs. FLOPs are calculated analogously for 10 sparsity levels as $F(s) = F_{\text{Low}} + (1 - s) * F_{\text{Ref}}$, where s denotes the sparsity of tensor entering the refinement part and F_{Low}, F_{Ref} the FLOPs required by the low resolution estimation and the refinement part respectively.

The findings show that, on average, a rise in sparsity leads to an decrease in accuracy (Figure 3.7). Error and sparsity are linearly associated in random selection scenarios, however using the actual refinement error difference as a sparsity mask causes the error to move in a more **desirable curve**, outperforming random selection and defining the potential benefit using an oracle. In light of the foregoing, introducing spatial sparsity in the refinement component, a set of operating states for this network are yielded, as shown in Figure 3.7. The goal of this

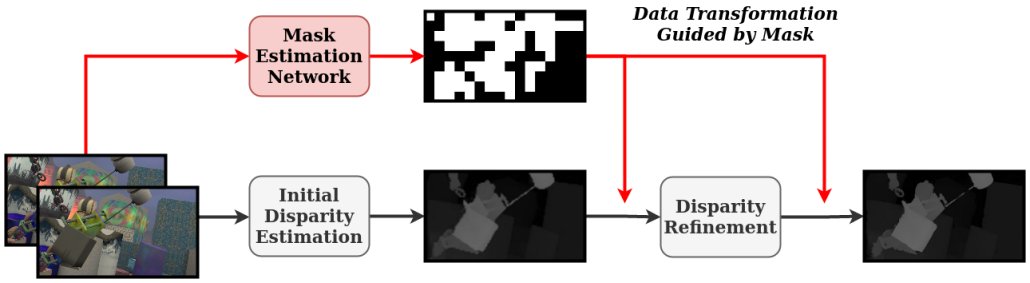


Figure 3.8: The architecture and flow of StereoNet marked in black color. Our proposed methodology (in red color) guides the the computations to more meaningful regions, according to the output of a parallel Mask Estimation Network.

work is to attempt to produce the best performing sparse states and emerge with an **improved performance-to-cost ratio**. It is worth noting that, the analysis above considers an ideal scenario regarding computational cost, since introduce additional computational overheads are introduced in practice, in the process of estimating a selection mask and transforming data structures. It does, however, provide an ideal performance-cost curve as well as an upper bound on predicted performance gains. This finding is also promising in terms of performance optimization, since it gives a possible direction for the application of spatial sparsity in edge-aware refinement pipelines. Moreover, since the network was trained with the fully dense refinement architecture, it does not expect the exclusion of some areas from the final refinement layers, therefore potential accuracy of sparse states could increase by a finetuning of the sparse refinement step.

3.3 Methodology

3.3.1 Method Overview

The proposed Spatially Sparse architecture (SS-StereoNet) adds a **parallel CNN mask estimation** network to the main StereoNet network. A low-resolution disparity is estimated similar to the original dense network, whereas refinement is executed in a spatially adaptive way. According to the output of the mask estimation network, the input is separated into active and non-active elements before the disparity refining section is executed. Training of the mask

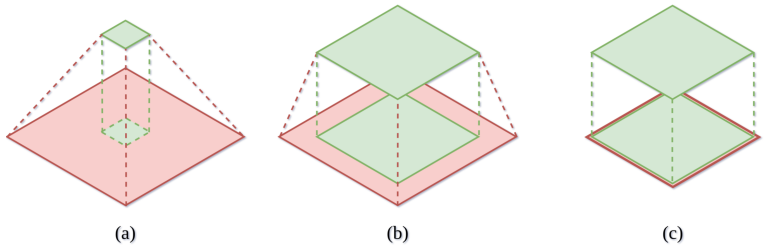


Figure 3.9: Figure displaying possible tensor handling methods, along with memory overheads. Red color indicates the input to the processing pipeline, while green is the output: (a) At *pixel* level every pixel is a single element, that carries its whole receptive field , (b) at *full-block* level each non-overlapping block carries the same receptive field, resulting in smaller memory overhead, and (c) at *fast-block* level, we drop the receptive field of border pixels, in favor of performance.

estimation network is conducted separately from the base network using custom ground truth data. Density d is defined as the ratio of active to the total number of elements, or active plus non-active elements, whereas sparsity can be defined as $s = 1 - d$. Then, refinement processing is applied only to the active blocks, which are combined together with the unrefined blocks to form the final full-resolution disparity map. Therefore, the final result contains both refined and unrefined areas. A graph presenting the full spatially adaptive refinement pipeline is found in Figure 3.8.

3.3.2 Blockwise Data Handling

To replicate the performance of the fully dense network on a given input tensor X $[H, W, C]$, the refinement processing for each pixel should be left intact. Considering sparse decisions at pixel granularity, each pixel is an element that can be either refined or not. This requires each output pixel to view the same receptive field r it views in the dense network, resulting in using data from r^2 surrounding pixels. The data requirement is also augmented with the use of dilated convolutions, which further enlarge the receptive field, resulting in significant memory overheads. In practice, the refinement part of StereoNet has a receptive field of 34 pixels when processing an image from the sceneflow dataset with resolution of 960×540 pixels, therefore considering each pixel for sparsity individually results in handling $34 \times d \times 960 \times 540$ elements.

To alleviate the growing memory requirements, we propose to reduce the granularity of sparse

decisions spatially, by grouping image regions into blocks. Therefore, the data overhead of the receptive field is mitigated since it applies to a group of pixels rather than each one. More specifically, each block carries the receptive field overhead only of border pixels, resulting in a smaller overhead, reduced by a factor of $1/(b_{\mathrm{H}} \times b_{\mathrm{W}})$, where b_{W}, b_{H} denote block dimensions. Therefore, the memory overhead can be controlled by the block size hyperparameter, which is set to square blocks of 60 pixels for our experiments. Moreover, after empirically evaluating using the block format with the full receptive field, a choice is made to ignore conveying the block receptive fields (which are relevant to border pixels only) entirely. This approach allows reduced memory usage and less processing time with minimal impact in accuracy. The types of data handling with pixel or block granularity along with their receptive fields can be seen in Figure 3.9.

3.3.3 Ground Truth Construction

The motivation section emphasized on the significance of accurately estimating the importance of different areas for disparity refinement, where the error-aware technique emerged as a promising solution, exhibiting substantial improvements over the random selection. Therefore, we opt to train the importance estimator to emulate this selection mechanism. To achieve this, a dataset based on the error-aware method is created, to leverage it as ground truth for training the mask estimation model.

For a given blocksize b, a set of *perfect* masks $M_{\mathrm{p}} = \{m_{\mathrm{p}} \in R^{n_W \times n_H}\}$, where $n_W = W/b$, $n_H = H/b$ are the number of blocks in the horizontal and vertical dimension, is calculated for every sample in the dataset and is stored as an additional temporary dataset. Masks contain the end-point error difference between disparity estimates before and after refinement $U_{\mathrm{ref}} \in R^{W \times H}$, essentially representing the mean refinement upside for refining disparity at each image block (x, y). To calculate this for each sample in the dataset, disparity estimates before d_{low} and after d_{high} refinement along with the ground truth disparity map d_{gt} are used as:

$$U_{\mathrm{ref}} = |d_{\mathrm{low}} - d_{\mathrm{gt}}| - |d_{\mathrm{high}} - d_{\mathrm{gt}}|, \tag{3.4}$$

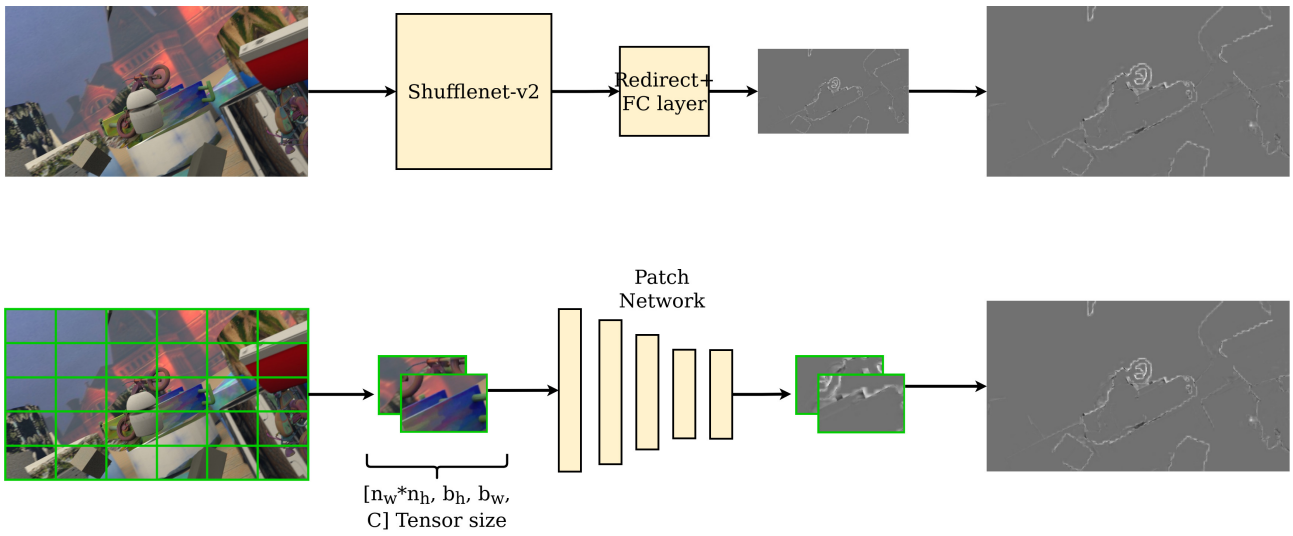


Figure 3.10: The architectures of the two proposed networks for Mask Estimation: *Full* architecture (top) and *Patch* architecture (bottom).

Then, the average value of U_{ref} is extracted for each block,

$$m_{\mathrm{p}}(x,y) = \frac{1}{b_{\mathrm{W}} b_{\mathrm{H}}} \sum_{i}^{b_W} \sum_{j}^{b_H} U_{ref}(i,j). \tag{3.5}$$

Consequently, the training can be performed in an end-to-end supervised manner, using M_{p} as ground truth targets for the mask estimation network.

3.3.4 Mask Estimation Sub-network

The mask estimation network is a fundamental component of the proposed approach, with the purpose of detecting the most optimal regions for refinement in the input image. It receives the left RGB image as input and outputs a float value (decision) for each block, representing a probability of it being important for refinement processing.

Training

To train the mask estimation network, the constructed error-aware ground truth data that indicate the importance of different image regions are used, providing a continuous representation of importance scores ranging between 0 and 1 for each block. Given that our objective is bi-

Layer	Output size	Kernel	Channels	Stride
conv1	30×30	7×7	32	2
conv2	15×15	3×3	64	2
conv3	8×8	3×3	64	2
conv4	4×4	3×3	64	2
conv5	4×4	3×3	32	1
FC	$n_{\mathrm{W}} \times n_{\mathrm{H}}$	$(32 \times n_{\mathrm{W}} \times n_{\mathrm{H}})$-$dim\ fc$		

Table 3.3: Patch Architecture for $[60, 60]$ patch.

nary classification of image regions based on a predefined importance threshold, the continuous ground truth values are binarized. Specifically, a thresholding operation to convert the continuous importance scores into binary labels is applied, denoting the desired density of refinement processing. Thus, areas with importance scores above the threshold are assigned a label of 1, indicating importance, while those below the threshold are assigned a label of 0, indicating non-importance. With the binarized ground truth labels prepared, a binary cross-entropy loss is adopted as the primary objective function for training our mask estimation network.

This problem could be also supervised using a distance metric loss like Mean Absolute Error, which would directly measure the distance between continuous values of predicted and ground truth importance scores. While this method may encourage the model to produce outputs close to the ground truth values, it does not explicitly enforce the binary classification criterion or prioritize correct classification decisions around the threshold. Therefore, the choice of binary cross-entropy loss aligns better with the nature of our task, which involves binary classification of image regions based on their importance scores relative to a predefined threshold. This threshold denotes the desired density for refinement processing and is introduced in training using the binary classification approach compared to a density-agnostic distance minimization approach.

Architecture

The Mask Estimation network is designed to be computationally light in order to reduce the amount of overhead added on top of the base network. We present two distinct architectures, according to the input image format; *SS-StereoNet-Full* uses the full resolution left RGB image

and *SS-StereoNet-Patch* receives the input image divided in non-overlapping patches. Figure 3.10 shows the network diagrams for both architectures.

In the first case, Shufflenet-v2 (Ma et al., 2018) network is selected to serve as the backbone, as it is pre-trained on a large dataset and has the advantage of producing strong features and a good starting point for training. An additional convolution is added to the end of the network, which redirects the backbone output to a 4×4 tensor with 32 features, along with a final Fully-Connected layer to aggregate information and produce a $m_{\text{est}} \in R^2$ of size $[n_{\text{W}}, n_{\text{H}}]$. As described above, *SS-StereoNet-Patch* operates on regions of the input RGB image. Drawing our inspiration from the patch architecture proposed in Cao and Zhang (2017), *SS-StereoNet-Patch* includes a lightweight patch classification architecture, presented in Table 3.3, where each layer contains a series of 2D Convolution, Batch Normalization and Leaky ReLU activation function. The mask estimation subnetwork receives patches of the RGB image corresponding to the spatial positions of U as input As a result, each patch's computations are carried out independently of the rest of the image.

3.3.5 Accuracy Restoration

With the proposed selection mechanism in place, there is a rule for selecting the most significant blocks for refinement. Only those selected blocks undergo refinement, while the rest are skipped. However, the refinement weights of the base network are tailored for dense image processing and are not optimized for the new topology. As a result, an extra accuracy restoration step has been added to adjust the weights of the disparity refinement part of the network to ensure achieve performance under the new sparse structure. To facilitate this, an extra training step is introduced after the mask estimation training, focusing on refining these weights only. This step mimics the original network training process: the weights of mask estimation are frozen, and Huber loss between disparity estimates and ground truth is minimized.

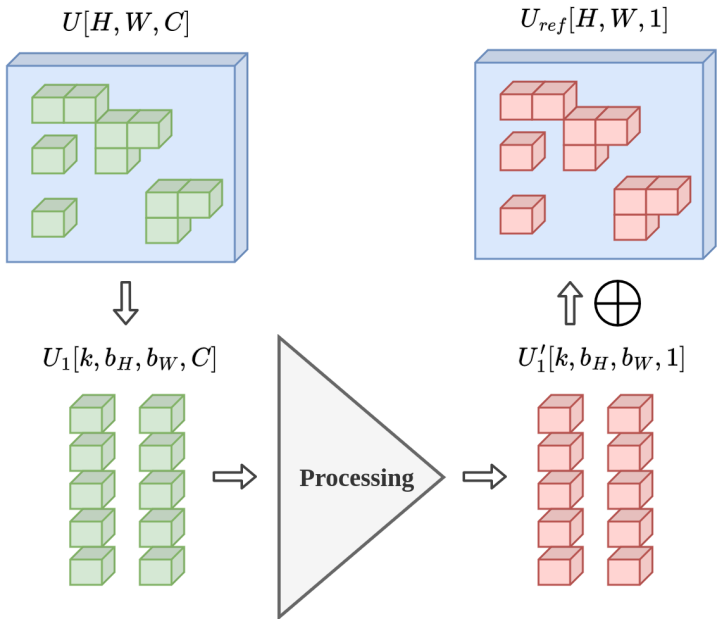


Figure 3.11: Graph presenting the spatially adaptive inference process of the disparity refinement part.

3.3.6 Inference Stage

During the inference stage of our methodology, the refinement stage of the network is executed in a spatially adaptive manner. More specifically, image blocks that are deemed important are processed by the disparity refinement part, whereas non-important blocks are not refined. The estimated importance scores for all image blocks are derived from the selection mechanism's importance estimation, in continuous float format. Given a provided sparsity factor k, these importance scores are sorted and the *top-k* blocks are considered important, while the remaining blocks are categorized as unimportant. This sparsity factor serves as a cut-off point, to control the number of blocks which will be processed by the disparity refinement, also guaranteeing a controllable computation budget for inference. This factor can differ from the threshold for binarization of ground truth targets, that was introduced in the training section, however best performance is achieved closer to the threshold value.

The practical inference process is presented in Figure 3.11. Guided by the importance estimation decisions, U_1 blocks are extracted from U tensor and are stacked in the batch dimension, which contain the important blocks identified through the importance estimation process. These blocks are then processed through the refinement pipeline to calculate the disparity residual

U_1' which is finally added as a residual to U, forming the final U_{ref} disparity map, containing blocks of both refined and unrefined disparity.

3.4 Experimental Results

3.4.1 Implementation Details

For evaluating the proposed sparse approach the StereoNet-8x-Single model is used, which is the same from the 3.2 Section. It has one full resolution refinement block after extracting a low resolution disparity estimate. The network is trained for 11 epochs in the *sceneflow-finalpass* dataset (Mayer et al., 2016) and achieves a performance of 2.24 EPE for low-resolution disparity estimation, using 54.42 GFLOPs and 1.59 EPE after the disparity refinement requiring 58.83 additional GFLOPs on the test set of sceneflow dataset. For experiments with the KITTI dataset, we further finetune the network for 300 more epochs on either KITTI 2012 (Menze et al., 2015) or KITTI 2015 (Menze et al., 2018), with learning rate equal to 0.001 which halves after 200 epochs. Typically, the whole training part of KITTI datasets is used for training and the test set is submitted and evaluated in the KITTI server. In our case, multiple network configurations are produced according to varying sparsity. Thus, multiple submissions to the server would be required, which is discouraged and is against the policy of the KITTI benchmark. Alternatively, as the dataset provider suggests for such cases, a subset of 160 image pairs from the training set are used for training, while the rest of them (34 for KITTI 2012, and 40 for KITTI 2015) are used for testing. This trained StereoNet serves as a performance reference point for our proposed sparse networks.

The sparse training process starts by training the only mask estimation sub-network on the sceneflow dataset, following the described methodology for 7 epochs, using Adam optimizer, along with 0.001 learning rate and batch size of 64. The block size is set to 60×60 for all the experiments. Next, given the trained mask estimation network, an additional 3 training epochs are performed for accuracy restoration, where only the weights of the refinement part

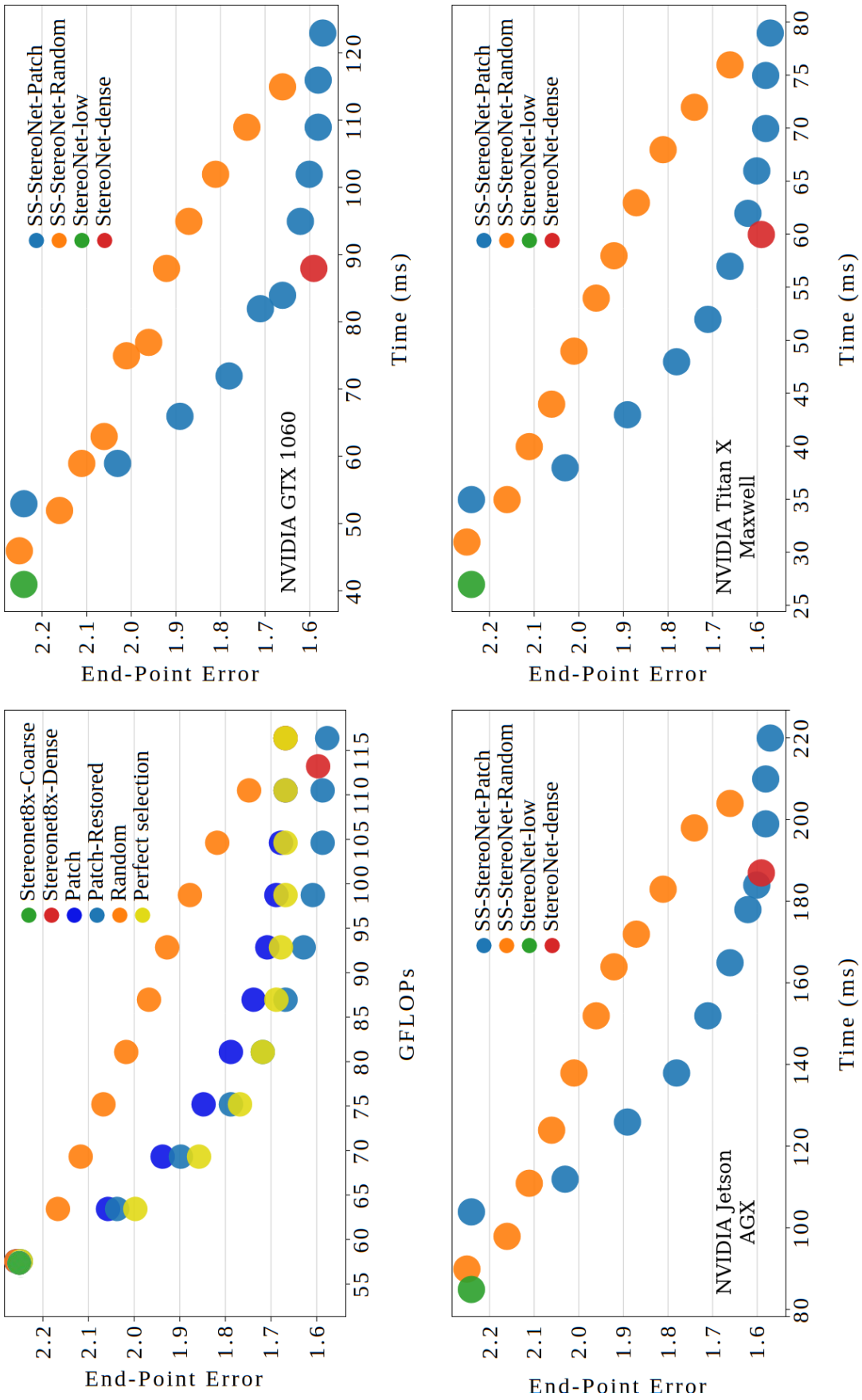


Figure 3.12: Figure presenting the results of our optimized method in sceneflow dataset. The top-left graph depicts the SS-StereoNet sparse operation states with respect to the theoretical FLOPS. The actual execution time of those operation states is presented, by implementing them on the following devices: NVIDIA GTX 1060 (top-right), NVIDIA Jetson AGX under MAXN profile (bottom-left) and NVIDIA Titan X Maxwell (bottom-right).

Sparsity	Architecture	FLOPs (G)	KITTI2012		KITTI2015	
			Out-All	Avg-All	Out-All	Avg-All
100%	StereoNet8x-low	50.00	6.67	1.29	4.35	1.14
0%	StereoNet8x-dense	104.04	4.99	1.05	2.56	0.78
50%	Perfect mask	-	5.60	1.11	4.52	1.04
	Random mask	77.22	5.90	1.15	4.52	1.04
	SS-StereoNet-Full	78.67	5.64	1.11	4.04	0.99
75%	Perfect mask	-	6.10	1.17	4.51	1.04
	Random mask	70.45	6.25	1.19	4.90	1.08
	SS-StereoNet-Full	71.90	6.08	1.17	4.57	1.04

Table 3.4: Evaluation on KITTI 2012 and KITTI 2015 datasets. The percentage of pixels with error bigger than 3 is reported, along with the overall EPE.

of StereoNet are updated. We use the Pytorch framework to implement our networks, while time benchmarks are performed with a Nvidia Jetson AGX embedded GPU device.

3.4.2 Quantitative Results

Internal Evaluation.

We perform an evaluation of the proposed SS-StereoNet on the sceneflow (Mayer et al., 2016), KITTI2012 (Menze et al., 2015) and KITTI2015 (Menze et al., 2018) datasets, by comparing it to its ground truth mask selection and dense/coarse counterparts. Samples from KITTI datasets contain images, which include a large region of irrelevant information with respect to driving, such as the sky or distant locations. As a result, they are well suited for the proposed optimization since, as shown in Table 3.4, finding such locations and omitting the computations has minimal influence on accuracy while reducing FLOPs.

Our SS-StereoNet clearly takes advantage of better area selection than random mask, almost reaching the ideal target (Figure 3.12). Moreover, with the additional accuracy restoration step, it outperforms the ideal target, pushing the sparse performance closer to StereoNet-8x-dense implementation. Our SS-StereoNet with 50% sparsity performs close to the dense version, with the minimal loss of 4.4% accuracy, while requiring 23% less FLOPs.

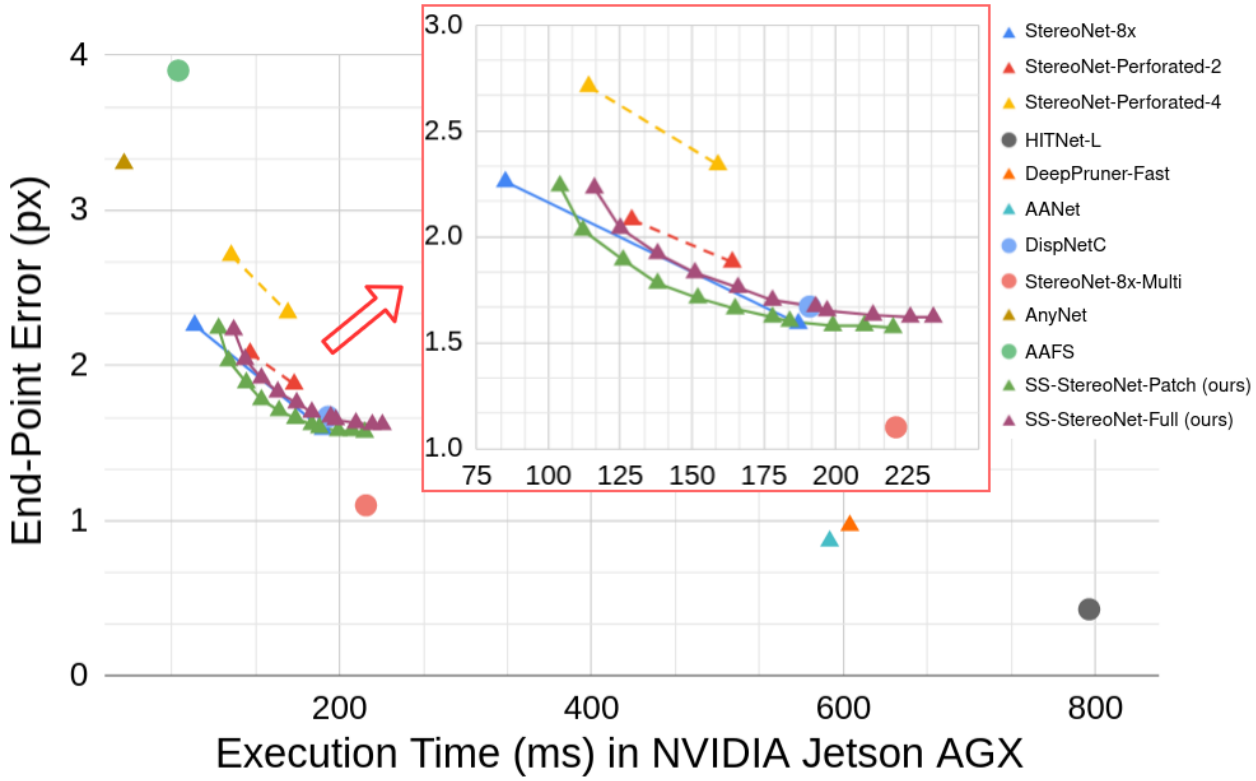


Figure 3.13: Evaluation on Sceneflow-finalpass dataset. Codes of cited methods are used from their public repositories. Execution times in milliseconds for a batch with size equal to 1 are measured on a Jetson AGX under MAXN power profile.

External Evaluation.

Compared to the other optimization method, ours outperforms basic Perforated CNN optimization (Figurnov et al., 2016) in both original and restored version, as seen in Figure 3.13, achieving a more efficient application of spatial sparsity. Regarding the stereo matching literature, there exist a set of methods which focus on improving accuracy, whereas others provide focus on achieving low latency. The original StereoNet is already well placed together with the best performing real-time methods. Therefore, our approach creates a competitive pareto front for stereo matching accuracy, which can be adjusted according to the desired FLOPs. Our SS-StereoNet achieves similar performance to DispNetC (Mayer et al., 2016), albeit with fewer parameters and faster execution time. Essentially, we provide an approach to bridge the gap between lightweight and ultra-lightweight stereo matching methods, under a controlled application of spatial sparsity.

Method	Params (M)	FLOPs (G)	EPE (px)	Time (ms)
PSMNet (Chang and Chen, 2018)	5.25	620.0	1.09	Out of Memory
DeepPruner-Best (Duggal et al., 2019)	7.39	368.0	0.86	2043
HITNet XL (Duggal et al., 2019)	2.07	187.0	0.34	1679
HITNet L (Duggal et al., 2019)	0.97	78.0	0.43	795
DeepPruner-Fast (Duggal et al., 2019)	7.47	135.0	0.97	605
AANet (Xu and Zhang, 2020)	3.93	115.0	0.87	589
StereoNet-8x-Multi (Khamis et al., 2018)	0.40	131.2	1.10	221
DispNetC (Mayer et al., 2016)	38.00	75.0	1.67	191
StereoNet-8x-dense	0.40	113.2	1.59	187
StereoNet-8x-Perf-2($L_{5,6}$) (Figurnov et al., 2016)	0.40	98.6	1.88	164
StereoNet-8x-Perf-2($L_{0\text{-}6}$) (Figurnov et al., 2016)	0.40	69.9	2.08	129
StereoNet-8x-low	0.28	54.4	2.26	85
AAFS (Chang et al., 2020)	0.02	0.7	3.90	72
AnyNet Y. Wang et al. (2019)	0.04	1.4	3.30	29
SS-StereoNet-Full (50%)	1.02	86.8	1.70	178
SS-StereoNet-Full (70%)	1.02	75.0	1.78	151
SS-StereoNet-Patch (50%)	0.44	87.0	**1.66**	165
SS-StereoNet-Patch (70%)	0.44	75.2	1.78	138

Table 3.5: Evaluation on Sceneflow-finalpass dataset. Codes of cited methods are used from their public repositories. Sparsity in our approaches is denoted with the sparsity percentage in parentheses. Execution times are measured on a Jetson AGX under MAXN power profile.

3.4.3 Qualitative Results

To visually assess the the impact and effectiveness of the mask estimation network, a comparison between disparity maps after coarse, dense and our sparse executions with 50% and 70% sparsity is performed in Figure 3.14. High-frequency regions such as the wheel of the motorcycle are selected for refinement under both sparsity levels, while the "flatter" region with the bottom block item is refined at 50% and skipped at 70% sparsity. Thus, we conclude that the mask estimation network under the proposed technique, learns a ranking of the importance of image regions with respect to the potential benefit in accuracy in their refinement. The quality of the output in both of the highlighted patches is deemed supierior to the original full resolution StereoNet, probably due to the adjustment of refinement layers to the high-resolution areas only (accuracy restoration), rather than a more generic fits-all approach of the original network. Moreover, a visual comparison of our SS-StereoNet and AnyNet (Y. Wang et al., 2019) is presented in Figure 3.15. Both methods employ an aggressive downsampling approach followed by full resolution refinement, however the refinement stage in the latter is more lightweight, hence the more blurry estimates in object borders, compared to the proposed SS-StereoNet.

This demonstrates the necessity of an effective disparity refinement scheme to produce high-fidelity disparity maps and the importance of doing that selectively in important regions only, to save computations.

3.4.4 Discussion

Patch vs Full Architecture

This work introduces two architectures for the Mask Estimation Network, namely *Full* and *Patch*. The former incorporates larger context resulting in improved mask estimation on data with large textureless areas (KITTI), whereas the latter requires high frequency information on a large number of extracted patches. Therefore, it is deemed more suitable for data with crisp details, like the sceneflow dataset, as compared in Figure 3.16. Because the *SS-StereoNet-Full* architecture works on the entire image, it has a broader receptive field than *SS-StereoNet-Patch*, allowing it to estimate smoother masks even when the input has few high frequency regions. *SS-StereoNet-Patch*, on the other hand, is limited to each block's local receptive field and also lacks positional encoding, necessitating the presence of numerous high frequency regions in the image to estimate a viable mask. When that is the case (sceneflow dataset), *SS-StereoNet-Patch* outperforms *SS-StereoNet-Full*, since it can capture the intricate details of each patch more accurately.

In conclusion, when comparing the two Mask Estimation Network architectures (*Full* and *Patch*), it is evident that each has its own strengths and weaknesses based on the nature of the input data and the properties of target masks. The key difference lies in the lack of positional information of *Patch* architecture, which is by default present when processing the full image along with an increased receptive field. Therefore, *Full* architecture is more effective at data with less samples and more textureless regions where position is key information for estimating the sparsity mask. On the other hand, *Patch* architecture which resembles Transformer embeddings, can produce better block importance predictions when more data are available and the position information is not important, as is the case with the synthetically

generated sceneflow dataset where objects are randomly allocated spatially.

Latency Investigation

To investigate practical gains, we also benchmark the sparse operation points of SS-StereoNet and their original architectures on three devices with different characteristics, shown in Figure 3.12. Across all devices, a linear decline of execution time is observed, as sparsity is increased. Actual practical latency gains, compared to the StereoNet-dense implementation, are achieved at around 50% sparsity for the desktop GPUs, while 20% sparsity is enough in Jetson AGX device, which is close to the ideal implementation. The reason is that the embedded device is kept more busy during the execution, compared to the more powerful desktop devices and its runtime is less affected by data transformation overheads.

3.5 Conclusions

This chapter presents an adaptation of the StereoNet network to introduce spatial sparsity to its architecture enabling a **performance to computational cost trade-off**, by skipping image regions, with reduced contribution to final output. They key takeaways from this work are:

- Our method creates a pareto front of possible operation points, **controlled by a sparsity factor**, which can be translated to the available computational resources for disparity estimation of a specific device and execution speed requirements.
- This method can be applied on top of a trained CNN to create a more computationally efficient version. Since it doesn't disrupt the network flow globally, it can also be **used in conjunction with other optimizations** (e.g. channel pruning).
- The practical speedup effect is more prevalent when applying sparsity at any part of a CNN that uses convolutions operating at **high resolution**, where computation redundancy is more pronounced.

However, the proposed approach comes with limitations and uncovers some important considerations when working at this space.

- First of all, the **non-differentiability** of the mask estimation module does not allow end-to-end training, therefore a two-step training process is proposed. Incorporating a differentiable training method for the mask estimation with a corresponding sparsity loss target would result in a smoother and less cumbersome training process.

- When departing from dense to the sparse data domain the corresponding **computation overheads for data conversion** are introduced. When focusing on theoretical efficiency (FLOPs) these overheads are often discounted, however they significantly affect the actual execution time on most devices, with the effect being greater at those which are mostly optimized for dense execution (GPUs). Therefore, in order to achieve actual execution time gains, the design of such data transformation operations from dense-to-sparse and sparse-to-dense data formats calls for highly optimized libraries, tailored to the target platforms.

- Finally, the block format introduces **border artifacts** from convolution padding, which limit the receptive field of refinement, especially in the current case where the receptive field of refinement is 34 pixels. Through accuracy restoration the network weights for each block are adapted to the local information rather than incorporating wider image content.

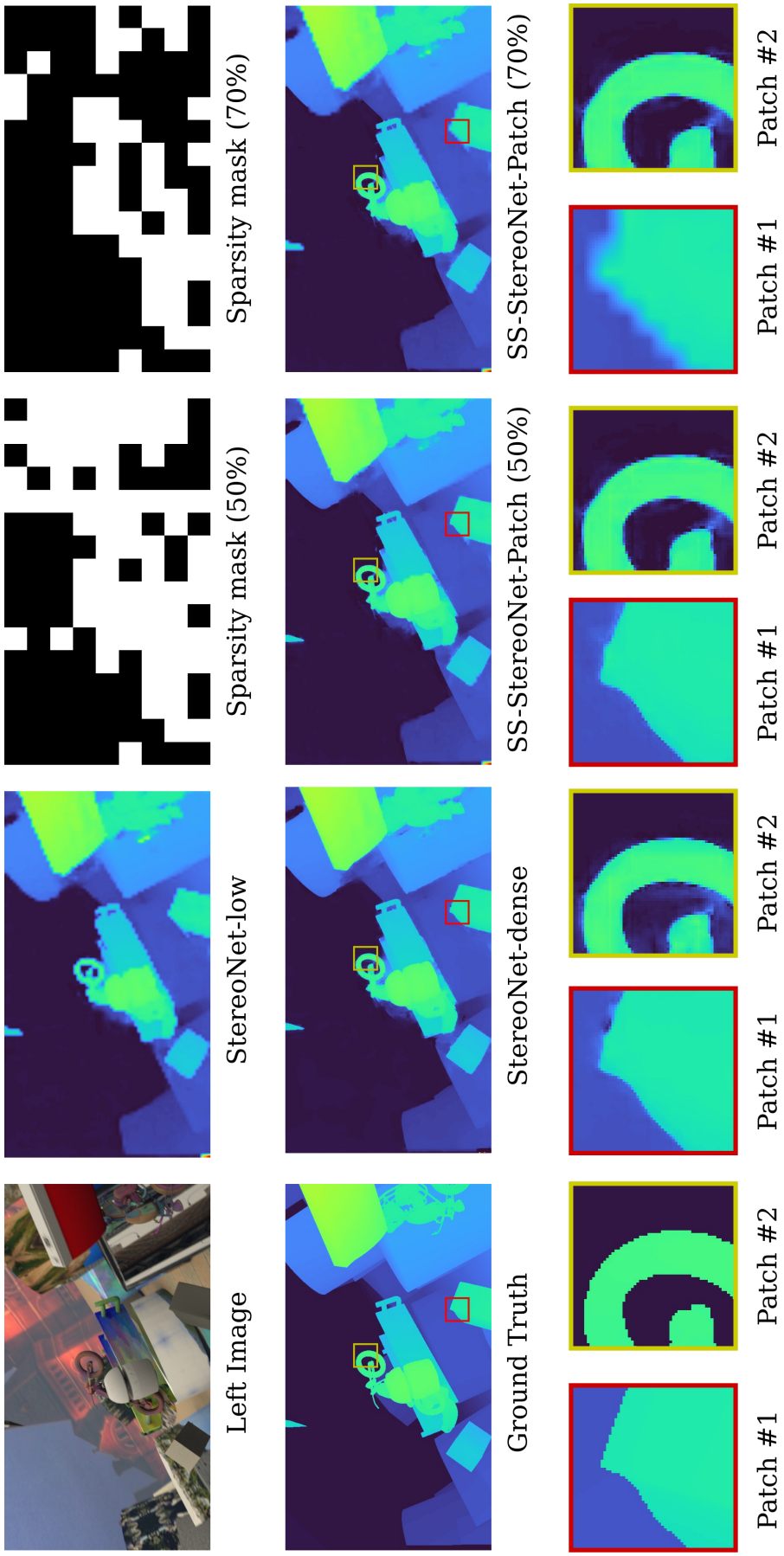


Figure 3.14: Qualitative evaluation of a sample from sceneflow dataset. Sparsity of SS-StereoNet is denoted in parentheses. Third row contains disparity estimate of selected patch samples from the disparity map above them. Note how Patch #2 is refined even at 70% sparsity, whereas refinement of Patch #1 is skipped at higher sparsity.

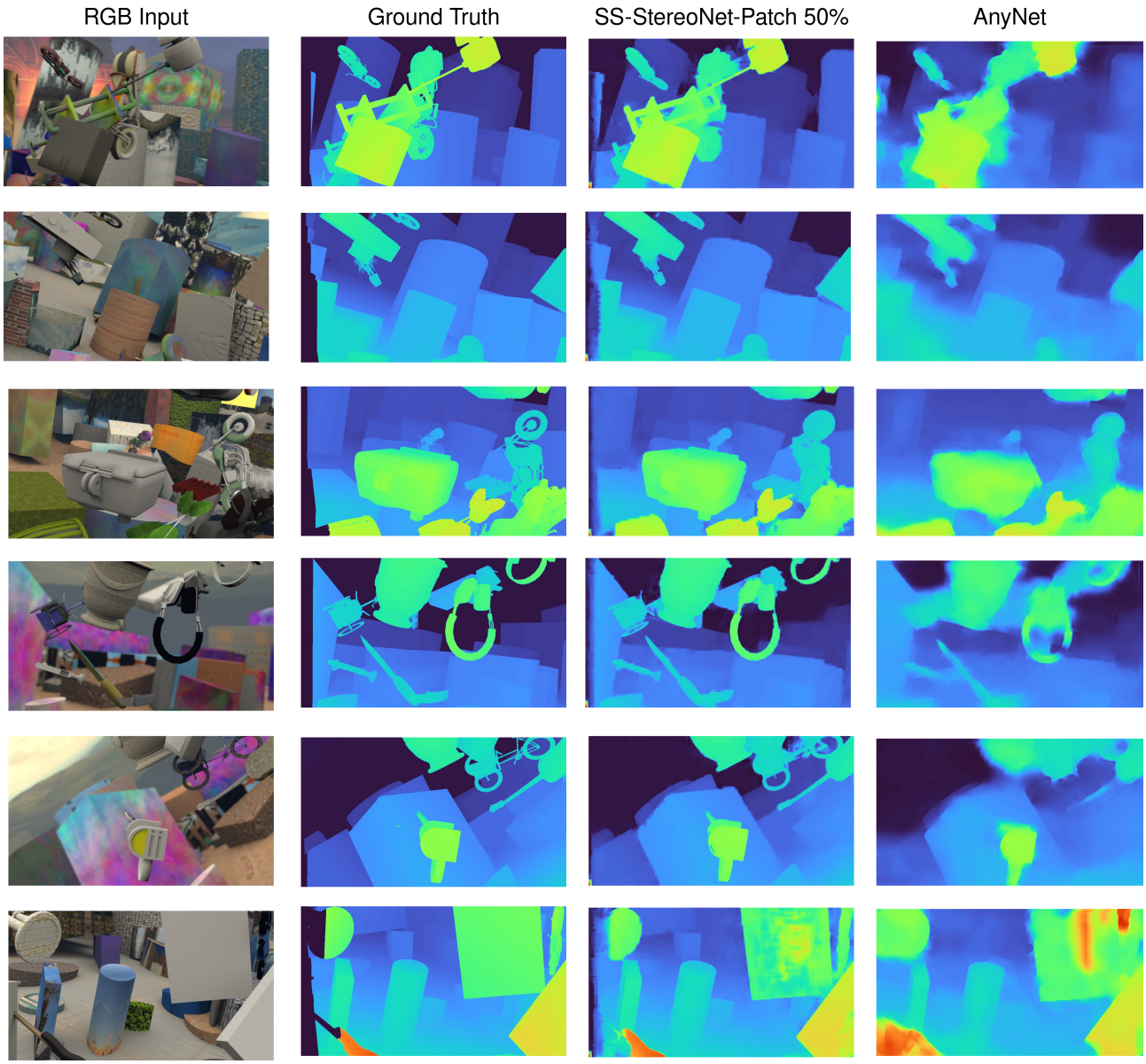


Figure 3.15: Qualitative evaluation of sceneflow dataset samples. Columns from left to right include: RGB input, Ground Truth, SS-StereoNet-Patch at 50% sparsity and disparity estimate from AnyNet Y. Wang et al., 2019 method.

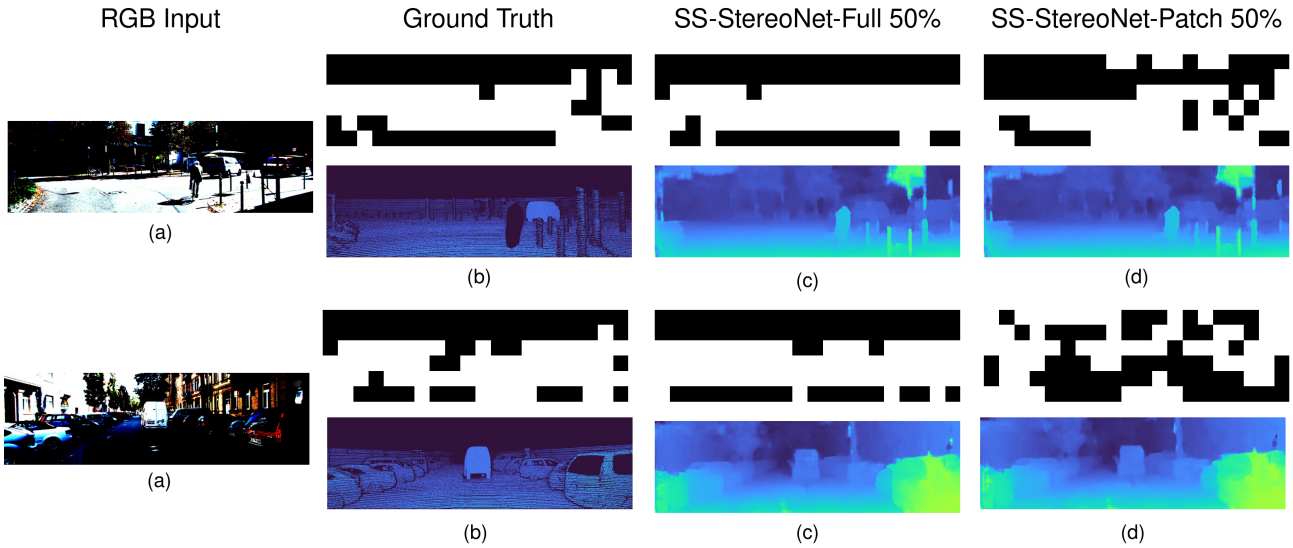


Figure 3.16: Evaluation on KITTI 2015 samples. Columns contain: input image (a), ground truth (b), SS-StereoNet-Full with 50% sparsity (c), SS-StereoNet-Patch with 50% sparsity (d). Top rows of each sample contain sparsity masks, while bottom rows contain disparity estimates.

Chapter 4

Latency-aware spatially adaptive optimization for semantic segmentation

4.1 Introduction

This chapter addresses the second research question: *How can we address computational aspects of data-driven spatially adaptive methods focusing on semantic segmentation CNNs?* Part of the work presented in this chapter is based on the conference paper: *Latency Driven Spatially Sparse Optimization for Multi-Branch CNNs for Semantic Segmentation*, which was published and presented in the Large Language and Vision Models for Autonomous Driving (LLVM-AD) workshop of 2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) [P2].

4.1.1 Semantic Segmentation

Semantic Segmentation, at its core, is a computer vision task aimed at deciphering and attributing semantic meaning to every pixel within an image. It plays a pivotal role in a wide range of applications, spanning from medical image analysis (e.g., tumor boundary extraction and measurement of tissue volumes), to autonomous vehicles (e.g., navigable surface and



(a)



(b)

Figure 4.1: The semantic segmentation (a) of a scene (b) from Cityscapes Cordts et al. (2016) dataset

pedestrian detection), video surveillance, and augmented reality to name a few. Unlike simpler image classification tasks that assign a single label to the entire image or object detection tasks that identify and localize specific objects, semantic segmentation takes the process further by providing a granular understanding of an image at the pixel level. The primary objective of semantic segmentation is to segment an image into regions and assign each pixel within those regions a label corresponding to a specific semantic class. This delineation allows for a detailed understanding of the different objects, boundaries, and structures present in the visual scene. The output of a semantic segmentation model is a dense pixel-wise prediction map, where each pixel is assigned a label representing its semantic class (Figure 4.1).

To perform the semantic segmentation of an image, an algorithm needs to discern intricate patterns in visual data to accurately assign semantic labels to individual pixels. To achieve

that, semantic segmentation algorithms typically follow three basic steps:

- Feature extraction
- Information aggregation
- Semantic map calculation

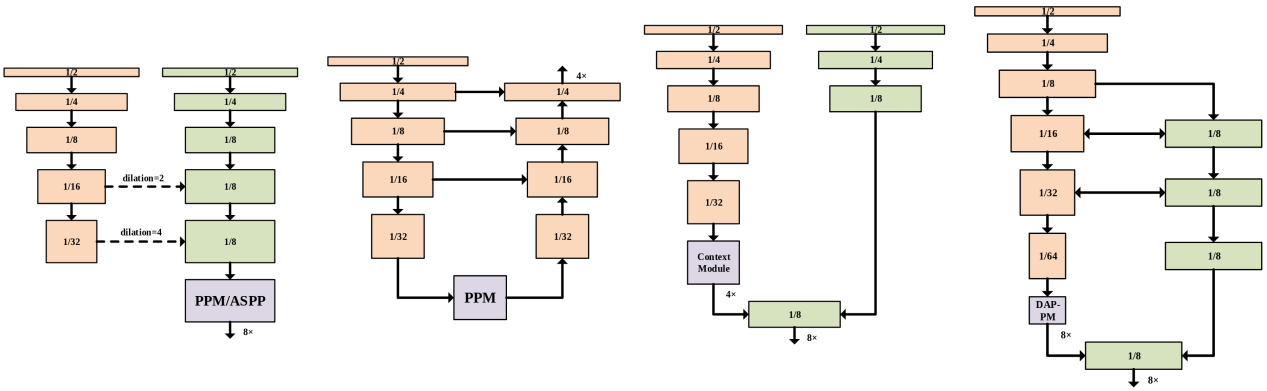
In this section, we p resent the current state of the art in semantic segmentation by exploring advancements in deep learning architectures and innovative methodologies that have pushed the boundaries of semantic segmentation. Moreover, a quantitative comparison table of these approaches and the relevant datasets will be presented.

4.1.2 Prior and Current State-of-the-Art

Prior Art

A multitude of image segmentation techniques have emerged in the literature, ranging from early methodologies like thresholding (Lei et al., 2023), histogram-based grouping and region-growing (Nock & Nielsen, 2004), k-means clustering (Dhanachandra et al., 2015), and watershed methods (Kass et al., 2004), to more sophisticated approaches such as graph cuts (Boykov et al., 2001) and Markov random fields (Plath et al., 2009). However, in recent years, the emergence of deep learning models has brought a new era of image segmentation methodologies, showcasing significant performance advancements. These models often outperform traditional techniques, setting improved scores in benchmarks for accuracy rates in various evaluations, signifying a paradigm shift in the field.

Among the pioneering approaches, the earliest deep learning methods for semantic segmentation laid the foundation for subsequent advancements in this domain. The seminal work of Long et al. (2015) was the first work which introduced Fully Convolutional Networks (FCNs), marking a significant advancement in deep learning-based semantic image segmentation. To accommodate images of varying sizes, the authors adapted existing CNN architectures like VGG16



(a) Dilated (b) Encoder-Decoder (c) Multi-Branch (Isolated) (d) Multi-Branch (Interlinked)

Figure 4.2: Popular backbone architectures for semantic segmentation. Figure reprinted from Hong et al. (2021).

(Simonyan & Zisserman, 2015) and (Szegedy et al., 2015), eliminating all fully-connected layers to produce spatial segmentation maps rather than classification scores, marking also one of the first reported approaches of classification backbone finetuned for downstream tasks. Employing skip connections, feature maps from the model's final layers are up-sampled and merged with earlier layer feature maps, effectively integrating semantic insights from deep and shallow layers, producing accurate and detailed segmentations. Since then, a common practice for semantic segmentation is to utilize architectures that are well established in the classification task as encoders, such as VGG (Simonyan & Zisserman, 2015), ResNet (He et al., 2016), MobileNetV2 (Sandler et al., 2018), ShuffleNetV2 (Ma et al., 2018), EfficientNet (Tan & Le, 2019), and add an upsampling module to recover fine details.

The effectiveness of most semantic segmentation algorithms depends on capturing long-range dependencies between picture pixels and contextual interactions, while preserving spatial information. He et al. (2015) introduced a popular technique to expand the receptive field of a network by fusing features from different strides, called Spatial Pyramid Pooling (SPP). This technique combined with backbones pretrained on ImageNet (J. Deng et al., 2009) and a decoder, formulate a versatile segmentation architecture (Orsic et al., 2019). The use of dilated or atrous convolutions is another popular technique, which allowed for an expanded receptive field without increasing the number of parameters or computational complexity significantly, was proposed by (F. Yu & Koltun, 2016). This pioneering work sparked interest in the commu-

nity and led to the emergence of Deeplab approaches (Figure 4.2a), which marked a significant milestone in semantic segmentation. Based on the initial Deeplab model (L.-C. Chen et al., 2015), (L.-C. Chen, Papandreou, Kokkinos, et al., 2018) proposed the introduction of atrous convolutions with different dilation rates in the spatial pyramid pooling (SPP), creating an Atrous SPP module (ASPP), able to capture multi-scale context, significantly improving segmentation performance. Deeplabv3 (L.-C. Chen et al., 2017) proposed a refined and effective use of atrous convolutions which progressively improved the architecture's ability to capture multi-scale context without the need for additional post-processing techniques like CRFs.

Current Art

The aforementioned approaches have focused on pushing the boundaries of performance; however, they tend to entail higher computational cost due to their more complex architectures and operations and the increased resolution of feature maps throughout the network. Additionally, some of these methods often exhibit increased inference times, further compounded by the absence of optimized implementations for dilated convolutions in widely used hardware, contributing to increased execution times during inference and training. Therefore, as the demand for deploying these advanced segmentation models in real-world scenarios surged, the necessity for methods that can achieve comparable performance with reduced computational and memory costs became paramount. One prevalent strategy to reduce computational costs involves employing an encoder-decoder architecture (Figure 4.2b), exemplified by L.-C. Chen, Zhu, et al. (2018). This technique involves iteratively downsampling the input to reduce complexity, with a decoder part focuses on recovering fine details lost in the downsampling process. SegNet, as proposed by Badrinarayanan et al. (2017), adopts a similar encoder-decoder structure but incorporates skip connections and transposed convolutions for efficient upsampling. Furthermore, approaches like ERFNet (Romera et al., 2018) take a unique path by leveraging residual connections and factorized convolutions, prioritizing efficiency without compromising accuracy. Meanwhile, GUNet, introduced in Mazzini (2018), introduces a guided upsampling module that integrates information from multi-scale input images, enhancing contextual awareness, while DFANet (H. Li et al., 2019), employs a sub-network and sub-stage cascade atop of a

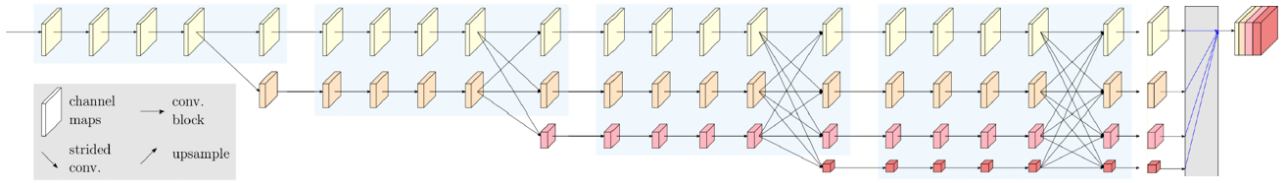


Figure 4.3: HRNet (J. Wang et al., 2021) architecture for semantic segmentation. Figure rerpinted from J. Wang et al. (2021).

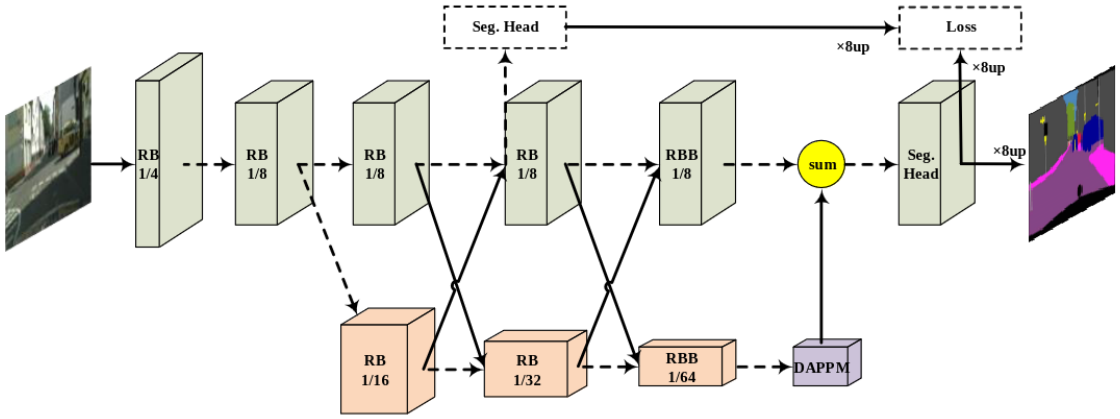


Figure 4.4: DDRNet (Hong et al., 2021) architecture for semantic segmentation. Figure reprinted from Hong et al. (2021).

lightweight backbone, efficiently aggregating discriminative features for improved segmentation performance. Drawing from the essence of efficient encoder-decoder architectures, the quest for lightweight yet accurate semantic segmentation models led to the adoption of depthwise convolutions (Chollet, 2017). Methods relying on backbones which use depthwise convolutions such as Hyperseg (Nirkin et al., 2021) which uses an EfficientNet-B1 (Tan & Le, 2019) backbone and RegSeg (Gao, 2023) using a ResNeXt (S. Xie et al., 2017) backbone have demonstrated impressive efficiency in terms of FLOPs and parameter count.

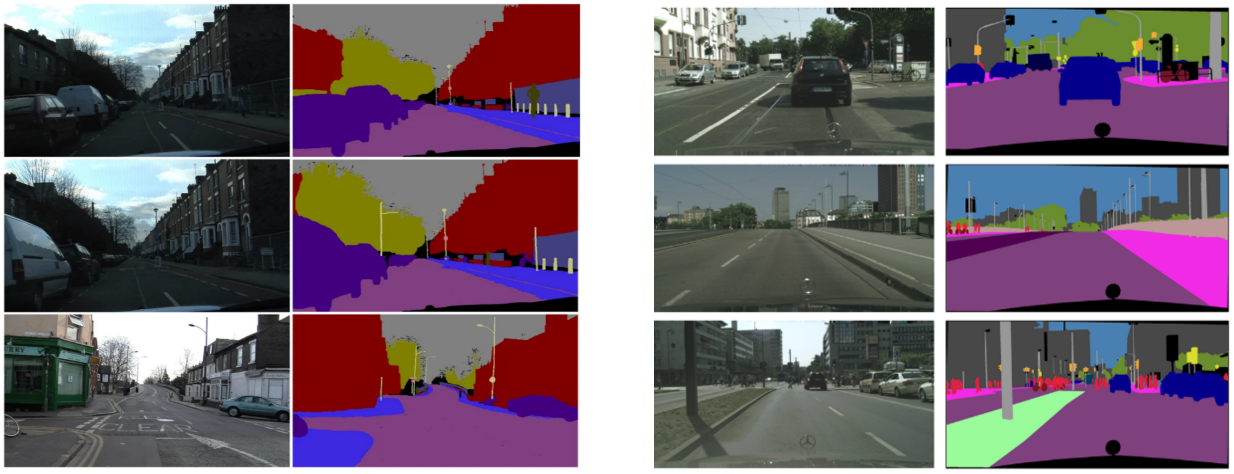
Information lost by downsampling is impossible to recover by upsampling, which introduces an upper bound on the performance of encoder-decoder methods. To address that, architectures with multiple paths (**multi-branch**) have been proposed in the literature, aiming to maintain high resolution representations during execution. BiSeNetV1 (C. Yu et al., 2018) and BiSeNetV2 (C. Yu et al., 2020) introduce a deep path with strided convolutions to increase receptive field, with a parallel shallower path to obtain high-resolution information. The two

paths are separate and a fusion module combines the outputs of the two paths in the final stage of the network. Similarly, CABiNet (Saksena, 2020) uses a dual-branch approach and introduces a low-cost, compact asymmetric position and local attention block. While those approaches demonstrated increased fidelity by using high-resolution branches, they rely on isolated branches which are not connected to the backbone until the final aggregation (Figure 4.2c). In an attempt to provide high-frequency cues to the backbone, more recent approaches introduced connections between the branches, establishing a consistent information flow between them during execution (Figure 4.2d). HRNet (J. Wang et al., 2021) architecture maintains high-resolution representations throughout the entire process by connecting parallel convolution streams operating at different resolutions and repeatedly exchanging information across resolutions. This approach enhances the quality of the representation, resulting in a semantically rich and spatially precise output that improves the overall performance of the network, while the complexity is curtailed by employing thinner branches on high resolution. Moreover, this approach demonstrates increased inference speeds compared to competing method at similar accuracies, because of its much thinner high-resolution feature maps throughout the network. Although high-resolution branches of HRNet are much thinner, they can be greatly enhanced by parallel low-resolution branches and repeated multi-scale fusion. The recently proposed DDRNets (Hong et al., 2021), further push the efficiency of multi-branch approaches by introducing a family of efficient backbones, carefully designed for semantic segmentation. They include a high and a low resolution branch, which communicate information using bilateral connections, while a Deep Aggregation Pyramid Pooling Module (DAPPM) is applied on the output of the low-resolution branch in order to extract multi-scale and global context information. Overall, the latest multi-branch architectures boast increased efficiency against the single branch counterparts since they can accommodate lighter high and low resolution paths, which are fused at multiple stages establishing information between multiple scales. A quantitative comparison of the discussed methods will follow in the upcoming evaluation section.



(a) ADE20K (b) PASCAL VOC 2012

Figure 4.5: Samples from general purpose semantic segmentation datasets (a) ADE20K (Zhou et al., 2019), (Zhou et al., 2017) and (b) PASCAL VOC 2012 (Everingham et al., 2015). Samples are organized in pairs of RGB image and ground truth segmentation mask.



(a) CamVid (b) Cityscapes

Figure 4.6: Samples from urban driving semantic segmentation datasets (a) CamVid (Brostow et al., 2009) and (b) Cityscapes (Cordts et al., 2016). Samples are organized in pairs of RGB image and ground truth segmentation mask.

Datasets

With the advent of deep learning technologies for the semantic segmentation task the need for densely annotated data has emerged to train and evaluate models effectively. Several datasets have been introduced that enabled research in various traditional computer and robotic vision fields ranging from generic scene understanding (PASCAL VOC, ADE20K) to others tailored for specific applications such as autonomous driving (Cityscapes, CamVid). Indicative samples of the discusses datasets can be found in Figures 4.5 and 4.6.

PASCAL VOC 2012 (Everingham et al., 2015) contains images of varying resolutions, up to 500×500, representing 21 classes (including a class for the background). This set originally contained 1,464 training, 1,449 validation, and 1,456 test images. Its training set was later extended by others to a total of 10,582 images.

ADE20K (Zhou et al., 2019), (Zhou et al., 2017) is a dataset offering diverse, densely annotated images across a broad spectrum of 365 indoor and outdoor scenes. The images in ADE20K encompass various resolutions (with median resolution of 300×200 pixels) and contain detailed annotations for over 150 object categories. This dataset provides $25,574$ images for training and $2,000$ images for validation purposes, fostering research in fine-grained semantic segmentation. ADE20K's comprehensive coverage of intricate scenes and numerous object categories makes it a crucial resource for advancing the understanding of complex environments in computer vision applications.

Cityscapes (Cordts et al., 2016) is one of the most popular urban street scene segmentation datasets, with pixel-level annotations for diverse objects of 19 classes, relevant to urban driving, such as roads, sidewalk, pedestrians etc. It consists of 2975 training, 500 validation and 1525 test samples, with 1920×1080 resolution, making this challenging for real-time semantic segmentation.

CamVid (Brostow et al., 2009), a urban street scene dataset akin to Cityscapes, comprises 960x720 resolution images and provides 367 annotated training images, 101 for validation, and 233 for testing.

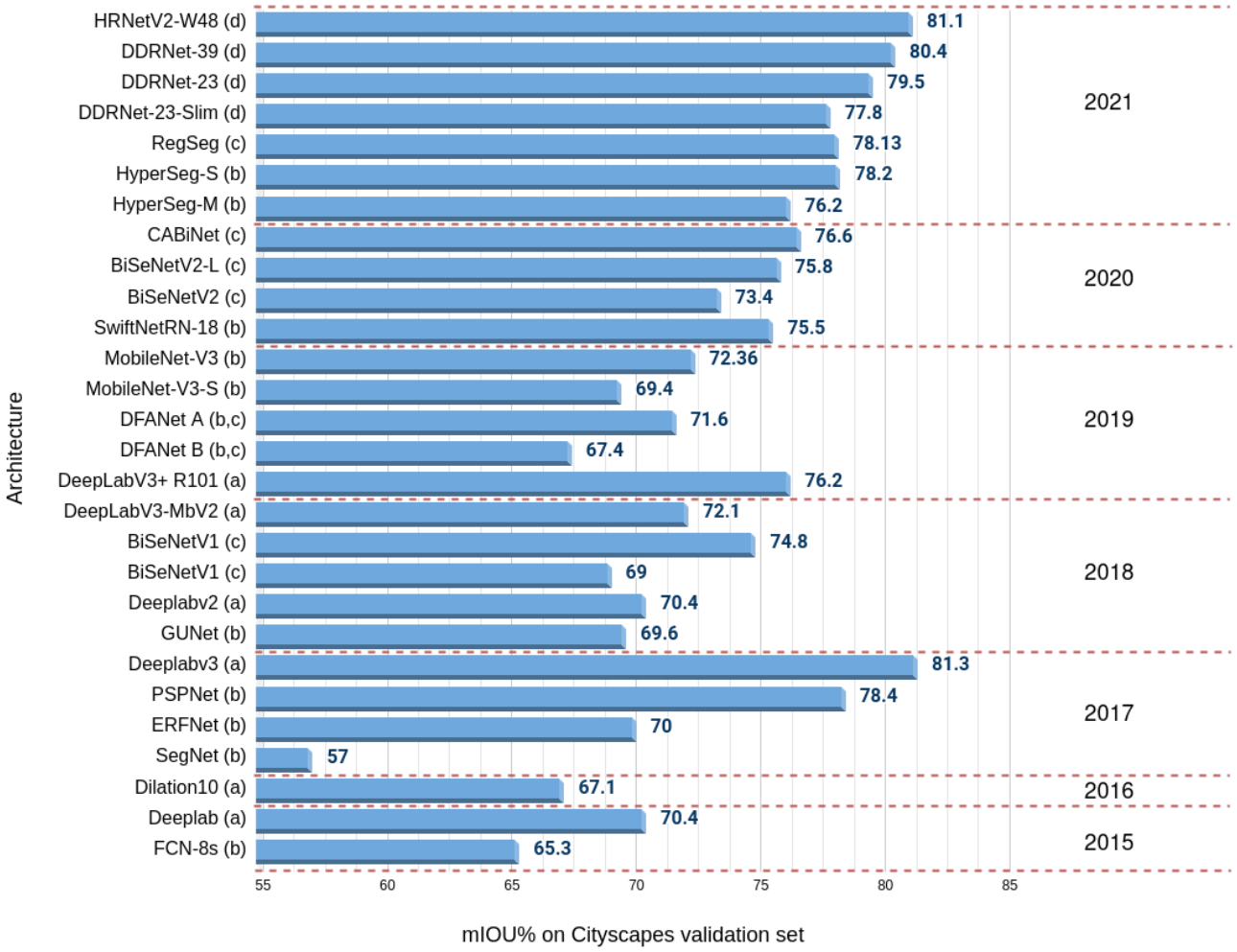


Figure 4.7: Evolution of state-of-the-art in semantic segmentation in mIOU% on Cityscapes (Cordts et al., 2016) validation dataset, sorted from newer (top) to older (bottom). Letter in parenthesis denotes the architecture category they use according to Figure 4.2.

Evaluation

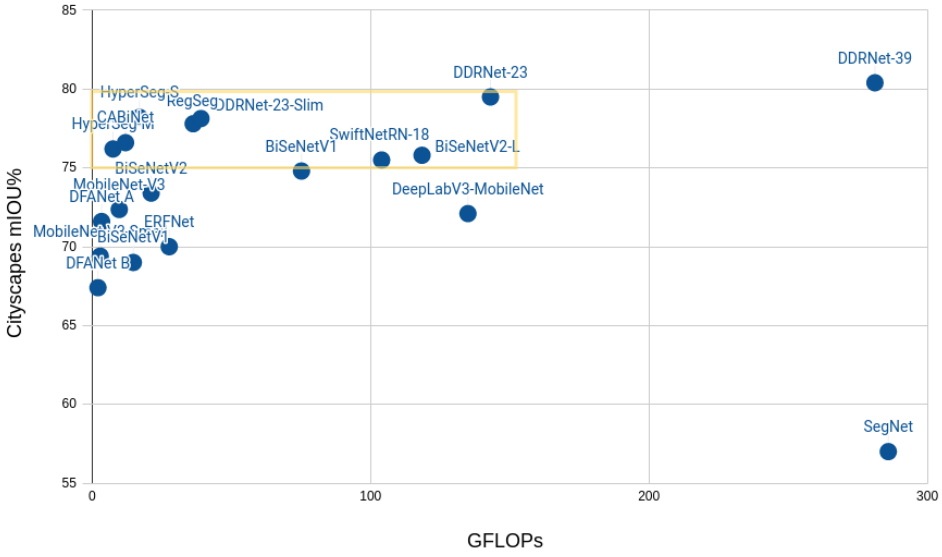
To evaluate semantic segmentation methods researchers typically rely on mIOU (mean Intersection over Union) metric; a measure that quantifies the accuracy of pixel-level predictions compared to ground truth annotations. The mIOU computes the intersection over union for each class and then takes the average across all classes, offering a comprehensive understanding of segmentation performance. Mathematically, mIOU is calculated as:

$$\text{mIOU} = \frac{1}{N} \sum_{i=1}^{N} \frac{TP_i}{TP_i + FP_i + FN_i} \tag{4.1}$$

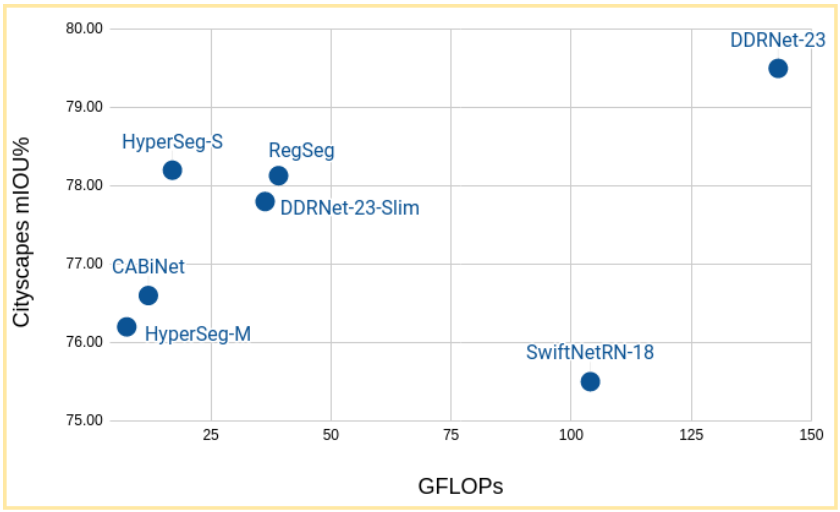
where TP_i represents the true positives (correctly predicted pixels for class i), FP_i stands for false positives (incorrectly predicted pixels for class i), FN_i denotes false negatives (missed pixels for class i), and N is the total number of classes. The higher the mIOU score, which ranges from 0 to 1 and often converted to a percentage, the better the segmentation model's performance across different classes in the dataset. The previously introduced Cityscapes (Cordts et al., 2016) dataset serves as a prominent benchmark for evaluating semantic segmentation methods due to its diverse collection of high-resolution urban street scenes captured in various weather and lighting conditions. Its rich annotations, covering 19 classes including pedestrians, vehicles, and road markings, make it an ideal choice for assessing the robustness and accuracy of segmentation models across complex real-world scenarios, therefore most recent semantic segmentation methods report mIOU scores on it.

In pure accuracy terms, deep learning methods have improved greatly since their emergence for semantic segmentation around 2015 with FCN (Fully Convolutional Network) initially measuring 65.3% mIOU on the Cityscapes dataset (Figure 4.7). Since then, accuracy has experienced a steady ascent over subsequent years, reaching mIOU scores over 80% by 2022. This rise can be attributed to innovative architectural advancements like spatial pooling pyramids (SPPs), atrous convolutions and multi-branch designs, enabling more refined feature extraction and contextual understanding, leading to a gradual improvement in segmentation accuracy.

While accuracy is an important indicator on the general performance of algorithms, it often

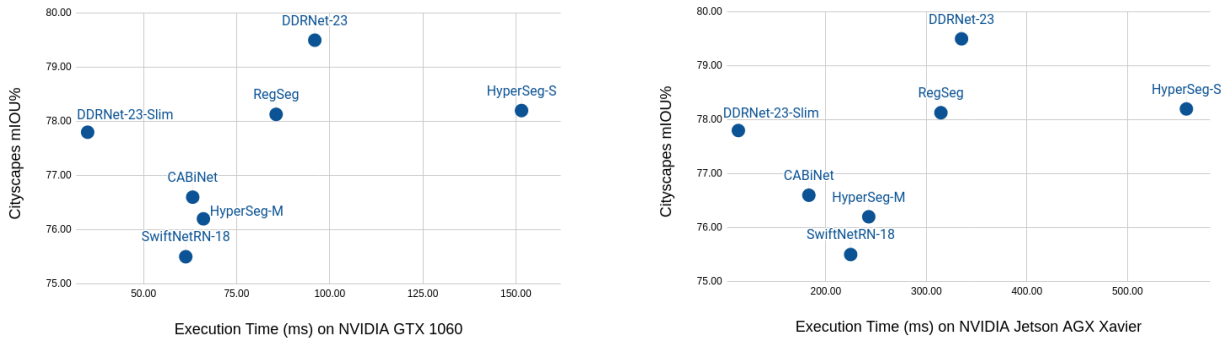


(a)



(b)

Figure 4.8: The state-of-the art landscape for semantic segmentation matching methods in terms of mIOU% vs normalized computational complexity on Cityscapes (Cordts et al., 2016) validation dataset as of 2022. Top (a) image includes all methods, whereas bottom (b) is limited to methods requiring less than 150 GFLOPs and mIOU% over 75%.



(a) NVIDIA GTX 1060 (b) NVIDIA Jetson AGX Xavier

Figure 4.9: The mIOU% on Cityscapes (Cordts et al., 2016) validation dataset over the actual latency on two devices: (a) an NVIDIA GTX 1060 desktop GPU and (b) an NVIDIA Jetson AGX Xavier embedded GPU.

comes with increased computational or memory demands, which can prohibit or limit their deployment on edge devices. Thus, it becomes valuable to examine each method for the trade-off between the achieved accuracy and the required computational cost or actual latency. The achieved performance measured with mIOU% metric compared to the required FLOPs is presented in Figure 4.8. As seen in the Figure 4.8a, there is a denser concentration of methods closer to the top left of the graph, translating to methods with greater accuracy and small computational budget, which maximizes efficiency. Focusing on most efficient and recent state-of-the-art approaches from Figure 4.8b, CABiNet (Saksena, 2020), HyperSeg (Nirkin et al., 2021), RegSeg (Gao, 2023) and DDRNet (Hong et al., 2021) define a pareto front of efficient semantic segmentation, with accuracy ranging from 76% to 79.5%, while GFLOPs range from 10 to 150.

As discussed in an earlier section, it's important to note that FLOPs serve as a proxy metric for latency and may not always directly correlate with actual latency experienced in a real system. Therefore, estimating the latency, rather than FLOPs alone, is essential as it provides a more direct and conclusive measure of execution time and responsiveness in deployment scenarios, ensuring a more comprehensive and practical assessment of computational efficiency. Figure 4.9 illustrates the relationship between accuracy and latency for two specific GPU devices: an NVIDIA GTX1060 3GB and an NVIDIA Jetson NX, employing the PyTorch (Paszke et al., 2019) deep learning library, and uncovers important conclusions about the FLOPs to latency correspondence. The FLOPs of HyperSeg, RegSeg, and CABiNet are similar to those of

DDRNet-23-Slim, while they are approximately one-third of the FLOPs of DDRNet-23. However, in terms of execution time in GPU devices, DDRNet-23-Slim is faster than HyperSeg, RegSeg, and CABiNet, whereas DDRNet-23 takes about 1.5 times longer to execute compared to these models. The reason is that methods such as HyperSeg, RegSeg, and CABiNet rely on depthwise separable and atrous convolutions, which are able to achieve similar performance with a fraction of parameters of regular convolutions, however they modify the structure and compactness of the data, resulting in less optimal device utilization than their regular counterparts, eventually exhibiting increased execution time in GPU devices.

4.1.3 Contribution

The previous chapter discusses the introduction of spatial sparsity to the refinement part of StereoNet, presenting a working paradigm of an optimization by exploiting spatial sparsity, while also showcasing the benefits and highlighting the limitations of such approaches. Given the similarities of semantic segmentation with the stereo matching task and the widespread adoption of CNNs for solving such tasks, there is increased motivation to pursue exploring spatially sparse network designs for different tasks and networks. Departing from previous work, our goal is to enhance our approach in three key areas: a) introducing spatially adaptive behavior to multiple parts of the network, b) achieving single-step end-to-end training, and c) prioritizing the consideration of latency rather than relying on proxy metrics such as FLOPs.

Therefore, we propose a framework which equips a semantic segmentation CNN with the ability to change the spatial distribution of computations during inference, by selectively activating processing modules, conditioned on the input. In a data-driven approach, the optimization process can consider either FLOPs or the actual latency of the network, to optimize a CNN for efficiency, introducing spatial sparsity to the network. Within the scope of this work, our emphasis is placed on GPU devices. To facilitate latency gains, a block-formatting strategy combined with a sparse data transformation method is utilized during inference.

4.2 Motivation

The primary drive behind this work is the pursuit of a spatially adaptive optimization approach, with a central focus on pushing the efficiency of methods situated at the pareto front of semantic segmentation. Meanwhile, it is also important to directly consider actual latency, rather than relying on proxy metrics.

4.2.1 Background Motivation

As discussed in the previously, multi-branch networks DDRNet (Hong et al., 2021), HRNet (J. Wang et al., 2021) have emerged as the state of the art in semantic segmentation, taking into account the trade-off between performance and computational cost or latency. However, there is a pressing need to generate models that can further push the performance-accuracy trade-off, as this has significant implications for various applications, such as real-time deployment and resource-constrained scenarios. Towards this end, we apply the sparse optimization methodology on this family of networks, ensuring more challenging but impactful contributions. Such networks employ one or more high resolution branches, which are executed parallel to the backbone to provide fine details which are lost due to downsampling. This strategy is akin to the disparity refinement process seen in StereoNet CNN (presented in the previous chapter), where the aim is to recover fine details lost during cost volume downsampling. Given that evidence, we assume that high-resolution branches could focus mostly on high-resolution details leading to the existence of redundancy in their computations.

Previous work on spatially adaptive optimization either demonstrates subpar results when applied to multi-branch networks (Verelst & Tuytelaars, 2023), or focuses on one specific layer type which does not translate to recent state-of-the-art approaches (Verelst & Tuytelaars, 2020). Moreover, previous work on sparsity optimization targets more “vanilla” networks (Verelst & Tuytelaars, 2023) which tend to be over-parameterized and redundancy is much more prevalent compared to the more optimized recent multi-branch approaches. In light of these limitations, our work aims to address these challenges by targeting state-of-the-art semantic segmentation

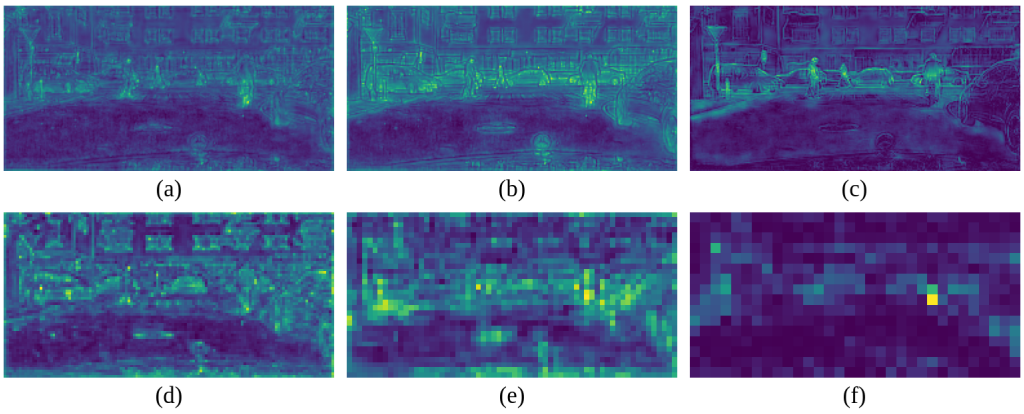


Figure 4.10: Figure demonstrating the summed feature maps along channel dimension of DDRNet23-slim (Hong et al., 2021) after residual layers of the high resolution branch (top) and the low resolution branch (bottom), from early (left) to latter stages (right).

networks.

The ultimate objective of utilizing sparsity is to minimize latency, which is a critical factor in real-time applications. Since directly measuring latency can be challenging, researchers often rely on the number of floating-point operations (FLOPs) as a proxy, which is not always correlated to latency, since the latter is influenced by factors such as the hardware and software implementation. Many dynamic methods achieve practical speedup of execution time (Verelst & Tuytelaars, 2020), (Verelst & Tuytelaars, 2023), (L. Wang et al., 2021), (Zampokas et al., 2022) but it mostly comes as a byproduct rather than being included explicitly in the optimization process.

4.2.2 Hands-on Motivation

In a typical CNN a full image is passed through the network and the same computations are performed for all image regions. By examining Figure 4.10, which contains the feature maps of DDRNet23-slim (Hong et al., 2021), a variation between the activation of various image areas after each layer is observed, especially in the high resolution branch. As execution progresses, the network tends to focus mostly on intricate high-frequency image details which are more challenging to classify. This suggests importance and impact of performing computations differs across different image areas. Therefore, the default uniform allocation of computations between

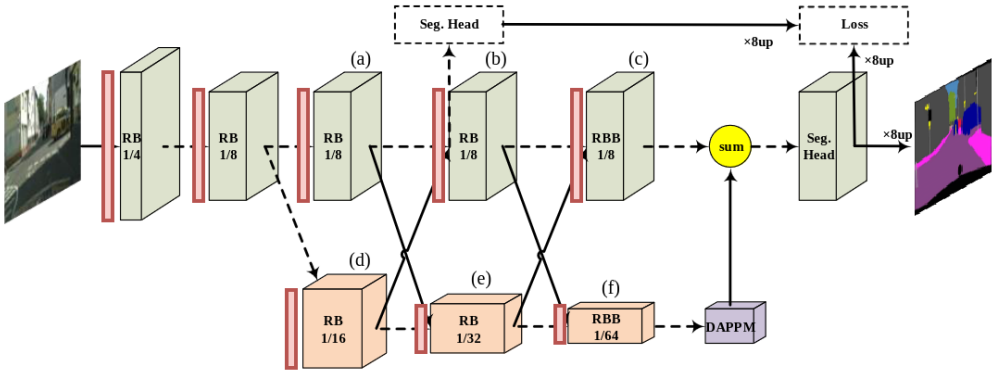


Figure 4.11: Introduction of random mask modules (red blocks) in DDRNet (Hong et al., 2021) architecture (Hong et al., 2021). Figure reprinted from (Hong et al., 2021) and augmented with additional information. (a-f) letters denote layers in correspondence with feature maps of Figure 4.10.

all layers and image regions (dense execution) could be sub-optimal in terms of compute resource allocation, compared to a different distribution tailored on the specific network, which could achieve increased efficiency.

An important aspect of spatially sparse designs is the ability to control the performance to computation ratio, to operate within the desired range. In the following analysis, we attempt to explore the potential computation redundancy of the network and examine the trade-off of performance by introducing spatial sparsity to the network. We employ a pretrained DDRNet23-slim (Hong et al., 2021) trained on the Cityscapes (Cordts et al., 2016) dataset as the baseline network. A mask module is introduced before every residual block, as seen in Figure 4.12. This module determines which areas of the input should be processed or skipped, in a random manner. Spatial sparsity is introduced into its residual blocks by densely processing the layers and selectively nullifying the residual impact in specific regions using the mask information. We systematically introduce sparsity into the network by progressively skipping a linear percentage of image pixels, ranging from 10% to 100%, resulting in multiple operation points with varying FLOPs, and perform this experiment 100 times. Referring to Figure 4.11, we observe that model scaling can be attained through the regulation of computational budget via spatial sparsity, thus yielding multiple operational points.

To explore the impact of sparsity placement on performance, we conduct two experiments.

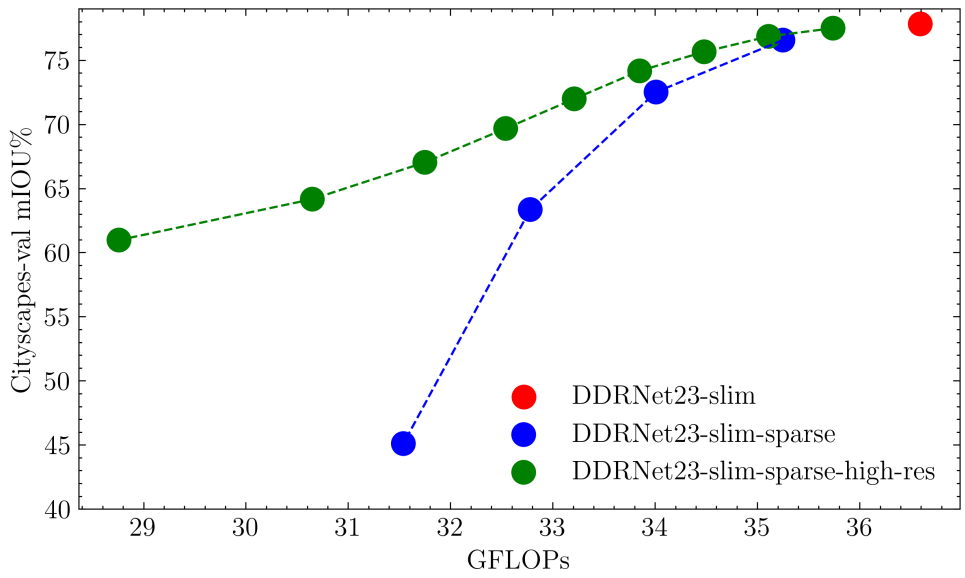


Figure 4.12: Performance of DDRNet23-slim (Hong et al., 2021) by introducing sparsity in a random manner in the both branches (green) and only in high resolution branch (blue).

First, we introduce sparsity to all residual blocks throughout the network and in the second experiment, we limit sparsity introduction to only the residual blocks within the high resolution branch, resulting in the blue and green lines of Figure 4.11 correspondingly. We observe that introducing sparsity in the second branch only, results in more optimal performance curve rather than the whole network, where performance degrades more rapidly. This indicates that the second branch could potentially include more redundant computations and therefore is a more suitable candidate to introduce sparsity. Finally, this performance curve is extracted using a random selection for the activation of binary masks, however a more sophisticated sparsity pattern of the activation could lead to improved performance, at the cost of the binary mask estimation overhead.

4.3 Methodology

Considering the conclusions from the previous section, we formulate a methodology that aims to increase the efficiency of CNNs for semantic segmentation by introducing spatial sparsity to multiple parts of the network as a way to reduce redundancy of computations. Additionally, instead of focusing on optimizing for theoretical computations (FLOPs), we introduce the

actual latency of the sparse implementation into the optimization process. The optimization framework is inspired by (Verelst & Tuytelaars, 2020), extending it to accommodate higher resolution data efficiently and introducing the latency optimization capability. In the following sections, the key components of the optimization framework are described.

4.3.1 Problem Formulation

To address the optimization of computational resources in semantic segmentation models, we formulate the problem of increasing efficiency of either FLOPs or latency as an optimization task, seeking to strike a balance between performance and resource utilization. To formally define the problem let's consider x as the base CNN model designed for semantic segmentation, which incorporates a number of residual blocks x_r. Three key functions are employed to assess the model's performance: a) $C(x)$ measures the computational complexity in terms of FLOPs, b) $L(x)$ quantifies the latency associated with model execution and c) $A(x)$ evaluates the model's overall performance using a task relevant metric (mIOU% in the semantic segmentation problem). To enhance efficiency, the residual blocks x_r are transformed into spatially adaptive residual blocks x_r^s capable of selectively skipping computations in less important regions. The importance of each region is determined using a set of importance estimation techniques x_m. The optimization process revolves around targeting specific computational budgets, either in terms of FLOPs, denoted as f_b, or latency, represented by l_b, which indicate the desired fraction of the corresponding quantity when using x_r^s instead of x_r. For FLOPs optimization and a provided budget f_b, the objective is to identify a model that maximizes $A(x)$ while satisfying the condition:

$$\begin{aligned} &Maximize \quad A(x) \\ &C(x_m) + C(x_r^s) \approx f_b C(x_r) \end{aligned} \tag{4.2}$$

Similarly, given a latency budget l_b the goal of latency optimization is to identify model maximizes $A(x)$ while satisfying the latency condition:

$$\begin{aligned} &Maximize \quad A(x) \\ &L(x_m) + L(x_r^s) \approx l_b L(x_r) \end{aligned} \tag{4.3}$$

In conclusion the problem of increasing efficiency in either FLOPs or latency, is formulated as an optimization problem aimed at finding an optimal configuration of the model parameters that maximizes performance metrics while adhering to the specified budget constraints.

4.3.2 Sparse Architecture

The DDRNet (Hong et al., 2021) and HRNet (J. Wang et al., 2021) multi-branch networks for semantic segmentation share very similar architectures (Figures 4.4, 4.3). They consist of two (DDRNets) or four (HRNets) parallel branches, which operate at different resolutions and exchange information between them at every stage, facilitating a consistent coarse to fine information flow. Higher resolution branches are generally lighter (fewer channels) than low resolution ones, to keep the complexity low. Their main processing modules are Residual Blocks and Bottleneck Residual Blocks. Given the constant interaction between layers, grouping them into one individual component like the refinement part of StereoNet is not feasible. In contrast, each Residual Block is considered an individual spatially adaptive component with the ability to follow its own allocation of computations by introducing sparsity. This processing is spatially sparse, meaning that computation is performed on specific regions rather than processing the entire input uniformly, assigning zero to residuals of inactive regions. To achieve this sparsity, a mask estimation module is assigned to each residual layer, responsible for determining which areas should be processed and which should be skipped.

Given the multiple instances of mask estimation module, the network from previous chapter is updated in favor of a more lightweight approach in terms of computational cost. To estimate the mask we choose a lightweight module, comprising of a 2D convolution with input channels

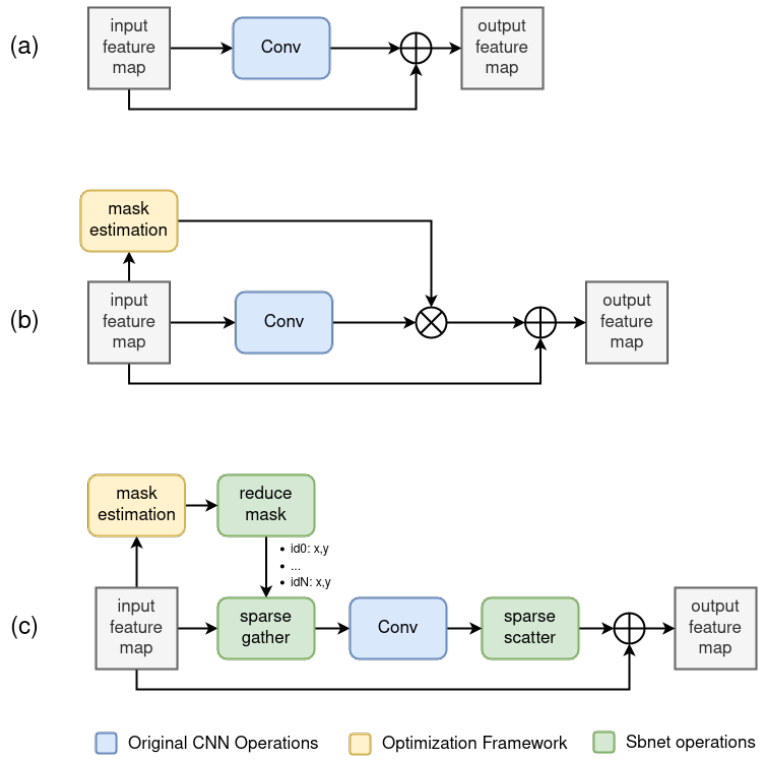


Figure 4.13: The adaptation of (a) a vanilla residual block for sparse execution using (b) mock sparsity implementation and (c) practical implementation using Ren et al. (2018) method.

equal to those of the input feature map and output channels equal to 1, similar to Verelst and Tuytelaars (2020). We further add Batch Normalization and Average Pooling modules to control the granularity of the output mask, by grouping image regions in square blocks. Departing from pixel level (finest) to coarser granularity provides a dual benefit. First, latency of the sparse execution is reduced, since a local structure is enforced in data, making the workload more friendly for GPU execution. Secondly, the coarser granularity of mask also enforces a regularization during training, since the search space is reduced, leading to improved accuracy. The FLOP overhead of the module is $1/C_{\text{in}}$, where C_{in} is the number of channels of the input feature map.

4.3.3 Sparse Training Framework

The Gumbel-Softmax Trick

Training mask decisions involves challenges due to the non-differentiable nature of selecting top-k activations with an argmax function. To enable end-to-end training without requiring an intermediate proxy dataset, the Gumbel-Softmax trick method (Jang et al., 2016) is employed. It uses the Gumbel distribution to incorporate categorical/discrete distributions (Coles, 2001), (Gumbel, 1958) into algorithms that use neural networks and still allow for optimization via backpropagation.

Selecting top-k activations represents a stochastic process defined by a discrete distribution, and it is not possible to compute a smooth gradient for either a stochastic or discrete process. The reparameterization technique (Jang et al., 2016), (Maddison et al., 2016), (Jang E, 2020), transforms the stochastic sampling process to a linear combination of a deterministic and a stochastic element. Specifically, the samples from categorical distribution z are defined as the sum of its mean μ, adding some stochastic noise σ creating a continuous distribution as,

$$z \sim \mathcal{N}(0,1) \rightarrow z = \mu + \sigma\epsilon, \tag{4.4}$$

with $\epsilon \sim \mathcal{N}(0,1)$. This allows to compute the gradient of the mean and the variance of the continuous distribution, and bypassing the stochastic node completely.

The reparameterization trick facilitates differentiability or the gradient flow during backward pass, but assumes a continuous distribution as input. In order to allow sampling from a categorical distribution, the deterministic and stochastic parts of the sampling process must be defined. The deterministic part is computed by taking the log probabilities of the logits, whereas the the stochastic part is represented by adding a noise. The selection of the noise function plays a critical role to the establish the desired behavior of the sampling process. For this reason, the Gumbel distribution is selected, characterized by its ability to model extreme values, since it is typically used to model the distribution of the maximums for a number of

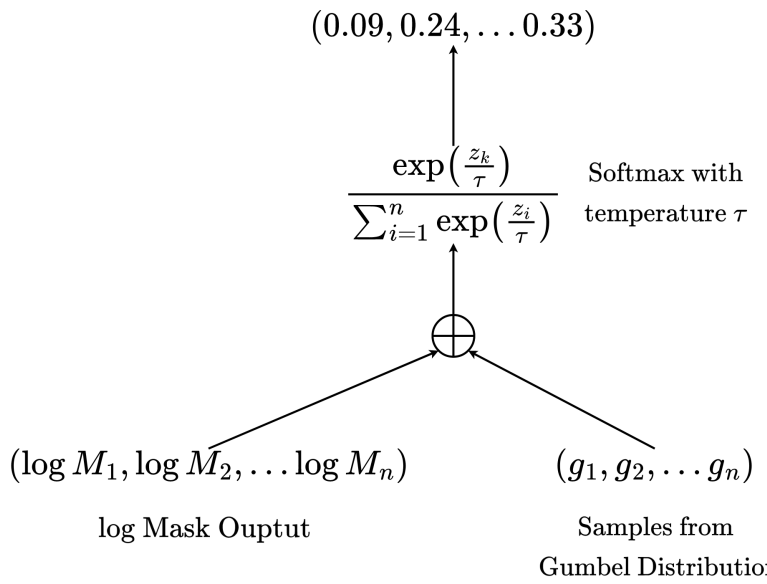


Figure 4.14: Figure demonstrating the Gumbel-Softmax trick that resembles discrete sampling.

samples pulled from other distributions (Gumbel, 1958),(Coles, 2001).

So far, we presented the methodology to convert the discrete distribution to a continuous distribution with the desired properties, but sampling still relies on the non-differentiable argmax function. To solve that, the differentiable softmax function was proposed as a replacement to argmax, as seen in Figure 4.14. In our setting, given $M_i \in \mathbb{R}^{N\times M}$ as the continuous output of mask estimation module for sparse layer i, the application of Gumbel-softmax is described by

$$z_i = \frac{\exp((\log(M_i) + g_i)/\tau)}{\sum_{j=1}^{k}\exp((\log(M_j) + g_j)/\tau)}, \tag{4.5}$$

with τ the temperature of Gumbel distribution where noise samples g_i are drawn from. During inference, locations where $M_i > 0$ are activated. A temperature τ is also used in the softmax function, which controls the "softness" or the level of exploration versus exploitation within the softmax function. Larger τ leads to a more uniform distribution of the expected classes, while with a smaller τ, the expected value of a Gumbel-Softmax random variable approaches the expected value of a categorical random variable with the same logits, as discussed by (Jang et al., 2016) and (Maddison et al., 2016). A typical training process, involves starting with a larger τ value encouraging the probabilities to be more evenly spread out, reducing the tendency for a single activation to dominate, essentially allowing for exploration of multiple activation during

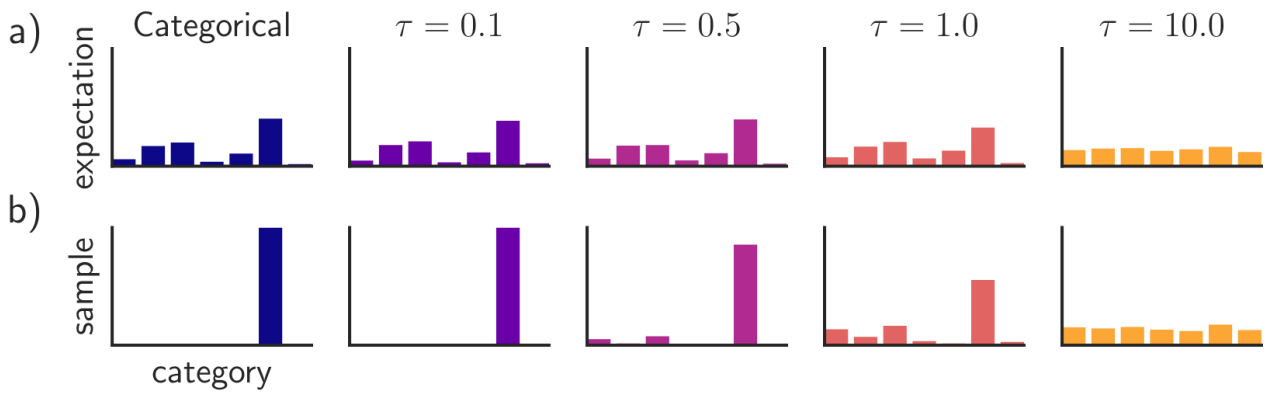


Figure 4.15: The impact of softmax temperature τ in the output expectation. Figure reprinted from Jang et al. (2016).

the training. As training progresses, temperature is typically decreased, producing sharper distributions, making it more peaky and favoring the selection of the most probable locations.

Supervision and Loss

Since all components are fully differentiable, training is performed in an end-to-end manner, similar to Verelst and Tuytelaars (2020). A computational budget hyperparameter f_b is provided as input to the training process, representing the fraction of the total computational budget of sparse layers. During training, the network tries to skip computations to reach the desired target budget, while achieving the best possible performance in the semantic segmentation task.

For a 2D convolutional layer i, the number of Floating Point Operations (FLOPs) is given by:

$$F_{\mathrm{i}} = 2 \times C_{\mathrm{in}} \times k_{\mathrm{x}} \times k_{\mathrm{y}} \times C_{\mathrm{out}} \times H \times W, \tag{4.6}$$

where $C_{\mathrm{in}}, C_{\mathrm{out}}$ are the number of channels for input and output feature maps respectively, $k_{\mathrm{x}}, k_{\mathrm{y}}$ is the convolutional kernel and H, W refers to input feature map resolution. Therefore, the sparse FLOPs for the same layer $\mathrm{F}_{\mathrm{i,sp}}$ are calculated as: Therefore, the number of FLOPs $F_{\mathrm{i,sp}}(d)$ required when the same layer is executed in a spatially adaptive manner, specifically

by performing operations solely on active regions is given by:

$$F_{\mathrm{i,sp}}(d) = d \times F_{\mathrm{i}}, \tag{4.7}$$

with d denoting the fraction of active to the total number of active image pixels produced by the mask estimation module. The discrepancy between the target computational ratio f_b and the ratio utilized by a network comprising B residual blocks is subsequently minimized through the following loss function:

$$J^F_{sp,total} = \left(\frac{\sum_i^B F_{\mathrm{i,sp}}}{\sum_i^B F_{\mathrm{i}}} - f_{\mathrm{b}} \right)^2, \tag{4.8}$$

Throughout training, the required FLOPs are averaged across the batch size, granting the network flexibility in determining their allocation across both residual blocks and spatial locations. However, inadequate initialization might lead the network to exclusively execute all positions within a block or none at all. This situation leads the mask units to a local minimum state, which is almost impossible to escape, as minimizing the sparsity loss becomes considerably easier than addressing the task loss. To this end, a constraint is enforced on each layer, by penalizing values away from the target budget of the network. This is formulated as an additional loss term:

$$J^F_{sp,layer} = \sum_i^M \left(\frac{F_{\mathrm{i,sp}}}{F_{\mathrm{i}}} - f_{\mathrm{b}} \right)^2. \tag{4.9}$$

This additional loss term can provide an initialization target for each layer, however limits their freedom to allocate computations between layers and explore the full space. Therefore, the equation of 4.9 is translated to a lower and upper bound, which are annealed through the training process from $\mathrm{b_t}$ to 0 and 1 respectively, gradually relaxing the boundaries. This ensures each layer has a proper initialization of sparsity to avoid extreme values and gradually gains the desired flexibility on number of operations to execute. Boundaries are enforced using the following equations, where p is changing from first to last epoch using cosine annealing from 1 to 0:

$$J^F_{sp,low} = \frac{1}{M}\sum_i^M max\left(0, p \cdot f_{\mathrm{b}} - \frac{F_{\mathrm{i,sp}}}{F_{\mathrm{i}}}\right)^2, \tag{4.10}$$

$$J^F_{sp,high} = \frac{1}{M}\sum_i^M max\left(0, \frac{F_{\mathrm{i,sp}}}{F_{\mathrm{i}}} - (1 - p(1 - f_{\mathrm{b}}))\right)^2. \tag{4.11}$$

Finally, Cross Entropy Loss to supervise the semantic segmentation task is included, leading to a final loss of the form:

$$J = J_{\mathrm{task}} + \alpha(J^F_{sp,total} + J^F_{sp,low} + J^F_{sp,high}) \tag{4.12}$$

with α as a hyperparameter to compensate for the difference in order of magnitude between task and sparsity loss and is set to 10 in our experiments.

4.3.4 Latency Aware Optimization

The methodology presented thus far has enabled optimization for a specified FLOPS target, a critical aspect in designing efficient neural networks. However, it's important to note that achieving optimization for FLOPS does not inherently lead to improvements in latency, which is equally vital in real-time applications. Therefore, while the current methodology caters to FLOPS optimization, the focus now shifts towards optimizing for a latency target. The subsequent steps necessary to achieve optimization specifically tailored to latency will be presented in the following sections.

Sparse Implementation

Given a mask decision, a straightforward approach to apply sparsity during inference on a residual block, involves implementing a mock sparse convolution (depicted in Figure 4.13-b). This method entails executing the workload densely and subsequently multiplying the output with the binary activation mask to negate the residual effect on the desired locations. When priori-

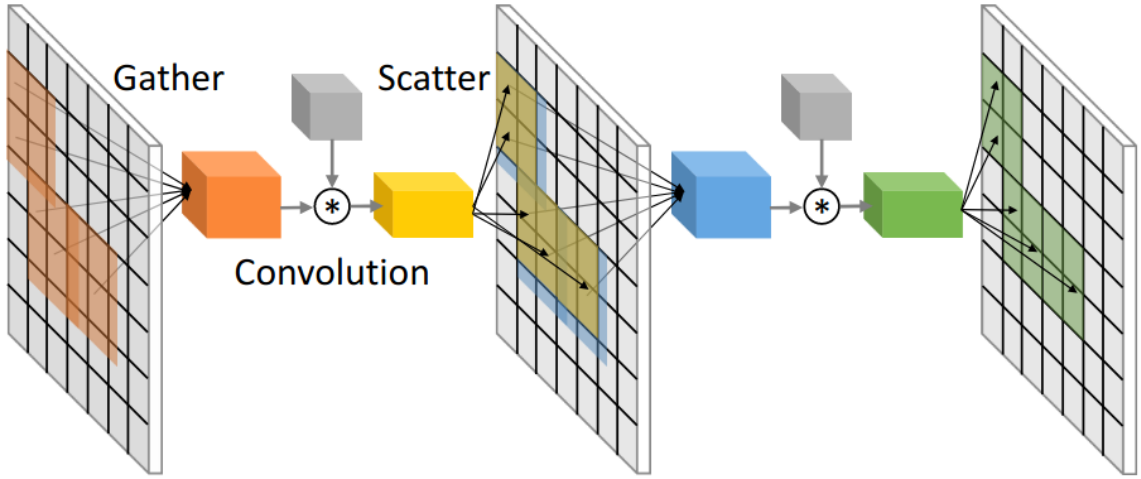


Figure 4.16: The sparse execution proposed by SBNet (Ren et al., 2018) denoting the basic operations. Figure reprinted from Ren et al. (2018).

tizing FLOPS optimization, the utilization of such mock implementation suffices for inference purposes. However, this implementation, while functional, lacks optimization and falls short in achieving lower latency than the extensively optimized dense convolution implementations, found in modern deep-learning libraries. Consequently, it is critical to establish an optimized sparse implementation, capable of surpassing the performance standards set by dense implementations at a given sparsity level, thereby ensuring tangible gains over their highly optimized counterparts.

To apply a sparse implementation, it is essential to define; a) a selection rule that determines which elements of the data are considered active, and b) a set of computations that constitute a target workload. Given these, a sparse implementation typically involves three steps:

- Based on this selection rule (such as a binary mask), the **mask reduction** step generates indices or locations for active processing.
- The **sparse gather** operation transforms the data from dense to sparse by extracting activated blocks and stacking them in the batch dimension, resulting in a tensor containing only the areas to be processed.
- The execution of the sparse workload only for the active areas.

- The **sparse scatter** operation handles the transformation from sparse back to the dense domain. After processing of the active areas has taken place, they are placed back to the original dense tensor, guided by the extracted grid indices.

This process involves multiple data transformation and access steps, which require data accessing and copying operations. Such operation introduce overheads which have been an important hurdle for the realization of practical gains using sparse network designs.

In the previous chapter, gather and scatter operations of SS-StereoNet were implemented by using Pytorch fold and unfold functions, which essentially require similar FLOPs and data accesses as a convolution operation. While this is a significant overhead, SS-Stereonet still managed to achieve latency speedup over the sparse variants for two reasons. First, the impact of the overheads was mitigated by (a) the fact that there was only a single instance of fold and unfold called for a single inference and (b) by the large sparse workload which consisted of 5 residual blocks. However, in the current setup, there is a requirement for multiple instances of sparse gather and scatter and the sparse workloads include single resblocks or residual bottleneck blocks, rendering the previous approach not a good fit for achieving practical speedup in sparse execution.

Background

To achieve practical speedup from dense convolutions it is important to come up with ways to mitigate the impact of those overheads. Submanifold Sparse Convolutional Networks (Graham & van der Maaten, 2017) fundamentally define sparsity as the presence of data occupying a lower effective dimensionality within its encompassing space. This key insight drove their proposition of a valid sparse convolution operation, an approach that selectively applies the convolutional kernel solely on the active, relevant regions of the data. Their pioneering work centered on establishing a sparse implementation framework, reducing FLOPs by targeting computations exclusively at the essential data points. Despite the significant reduction in FLOPs, this sparse implementation technique did not lead to significant improvements in latency.

The optimization of CPUs and GPUs primarily caters to dense calculations, necessitating structured and localized data for efficient processing. However, many input sources exhibit a form of

structured sparsity, wherein certain regions, like background pixels, tend to be surrounded by similar elements. Based on that, Ren et al. (2018) presented an algorithm for handling block-sparse to dense and dense to sparse data transformations, targeting nVIDIA GPU devices. Using a mask input, the reduce mask module performs a combined max pooling (or average pooling) operation localized to each block an generates a list of indices for blocks where either max or average pooling value exceeds specified tolerance, corresponding to blocks that will be processed (active). Sparse gather and scatter operations handle the dense to sparse and sparse to dense data transformations, by performing copy operations using the active block indices. Both gather and scatter are data copy operations, therefore their execution times depend on the amount of data that they process, similar to the actual sparse workload. Thus, practical latency gains in sparse execution follow a non-linear increase as sparsity is increasing. The three sparse operations are implemented in CUDA, to make use of optimized functions and offer reduced latency for GPU devices. Figure 4.16 demonstrates the introduction of SBNet data transformations to realize spatially sparse execution.

Adopted Method

We adopt this method to apply spatial sparsity to CNNs in the proposed optimization framework, by grouping image regions in blocks. However, Ren et al. (2018) was designed on CNNs using 3D Lidar data, which are inherently more sparse than 2D images and operate at much higher sparsities without loss of important information. Additionally, the requirement of our method to allow dynamic behavior on a per-layer basis, introduces the overhead of the mask estimation along with data transformations, further reducing the window for latency gains.

Latency Profiling - Sparse Benchmark Bank

When using FLOPs as an optimization objective, a linear function (Equation 4.7) is employed to map computational cost (FLOPs) of a single layer to activation density d. In the latency domain, this is regarded as an ideal implementation, assuming a direct 1-1 relationship between FLOPs and latency without accounting for the overheads of computation in the sparse space, resulting in the $L_{i,sp,ideal}(d)$ line in Figure 4.17. However, as discussed this ideal function is

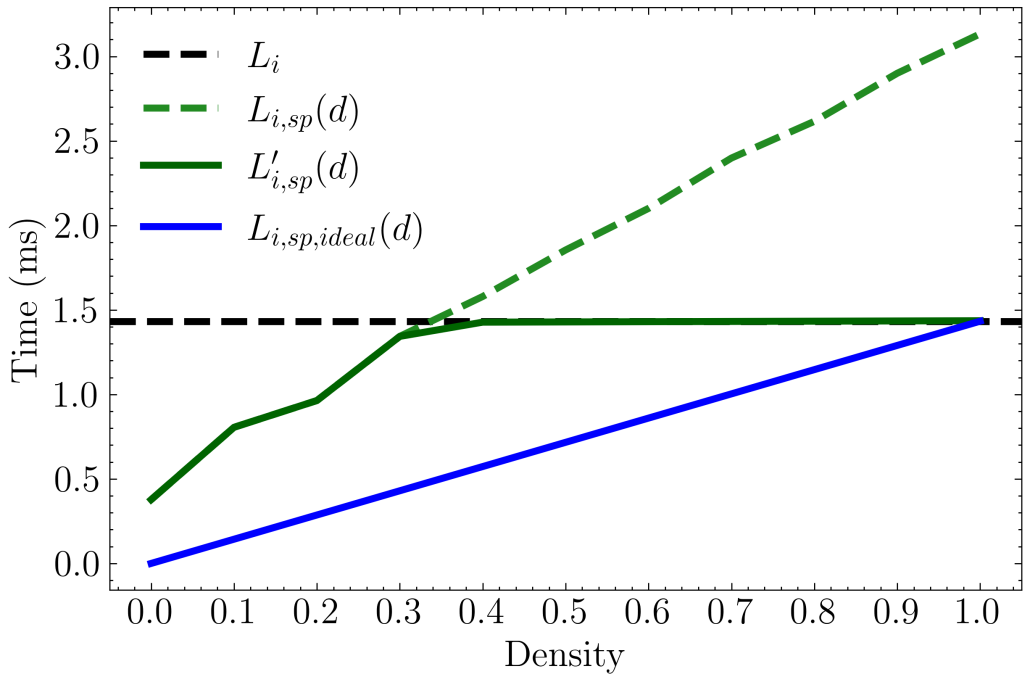


Figure 4.17: Latency functions for the first residual layer of DDRNet23-slim used for training.

rarely the case when targeting real devices, due to the overheads introduced by the hardware and software implementation when dealing with unstructured sparsity. These overheads include the operations to copy data from a dense format to a different memory location (sparse format) and from a sparse format back to a dense one. Since CNN execution is assumed to be compute bound, such memory transfers are not captured by measuring FLOPs, therefore a discrepancy between FLOPs and actual execution latency is observed on real devices.

Departing from FLOPs to practical latency optimization, a function which models the relation of density to actual latency needs to be used, considering the layer, blocksize, device and software implementation. Such function encapsulates all the required transformation and computations to achieve the spatially sparse execution of N active regions of a convolutional workload as,

$$L_{i,sp}(d) = t_{mask}(d) + t_{reduce}(d) + N(t_{gather}(d) + t_{conv}(d) + t_{scatter}(d)), \tag{4.13}$$

with each $t(d)$ denoting the execution times required for each operation under a specific density d in milliseconds.

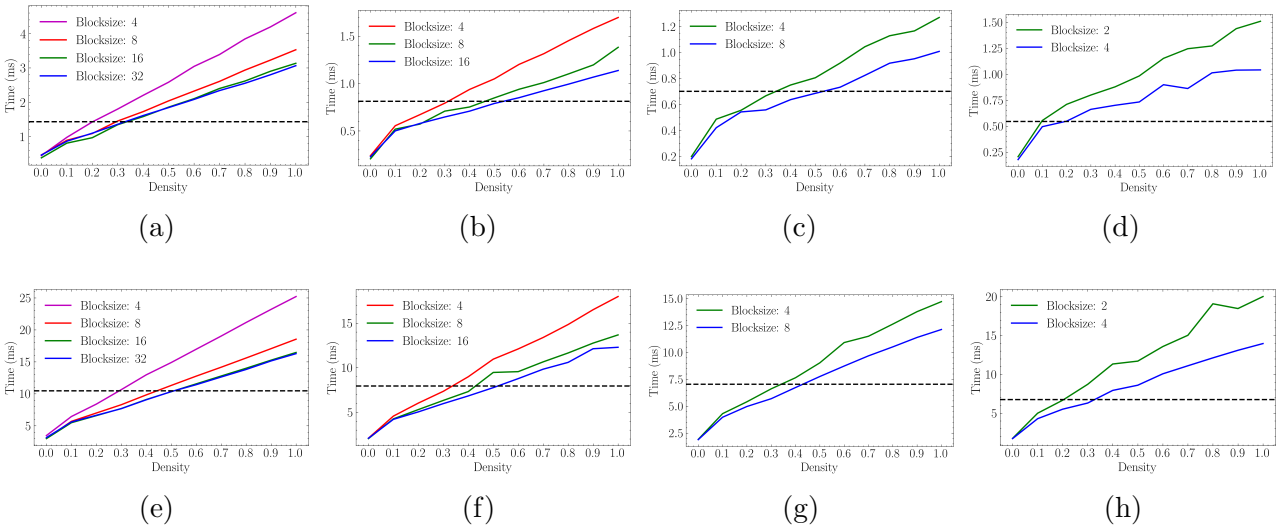


Figure 4.18: Figure presenting the sparse profile for different layers (rows) of DDRNet23-slim (Hong et al., 2021) in two devices: NVIDIA RTX 3060ti (a-d) and NVIDIA JetsonNX (e-h) under various blocksizes (colored lines). Dashed line denotes the dense execution time for the corresponding layer.

To construct this function for a given network and target device, a series of benchmarks are proposed to be conducted for each individual layer of the CNN, using the sparse implementation introduced in the previous section. To account for the mask estimation overhead, we further include the mask estimation computations in the benchmarks, resulting in a realistic latency profile of the CNN layers. By noting the execution times for multiple density levels from 0% to 100% in steps of 10% and interpolating in between, the function $L_{i,sp}$ is created. The latency of the corresponding layer under the same configuration, L_i is also calculated. Latency profiles for multiple layers of DDRNet23-slim (Hong et al., 2021) network for two devices can be seen in Figure 4.18. We can evaluate the effectiveness of sparse implementation across various layers and devices by noting the density of the intersection point of sparse timings with the dense time. NVIDIA Jetson NX exhibits generally superior performance across most layers, allowing latency gains at slightly higher densities compared to the NVIDIA RTX 3060ti, however layers from the high-resolution branch (depicted in Figure 4.18 c,g) demonstrate better performance on the latter.

Latency Aware Training

To introduce latency to the optimization process, the timing function $L_{i,sp}(d)$ is used to map density of each layer to actual latency, replacing the FLOPs function $F_{i,sp}(d)$, which maps density to the number of FLOPs. We observe that latency gains using the sparse implementation are achieved only when active element density is low, whereas when dealing with higher densities, the execution times become considerably slower compared to t_{dense}. To address this, a mock sparse implementation (Figure 4.13-b) can be employed at these density levels, executing the layer in a dense manner and subsequently applying the sparsity mask to negate the convolution effect for inactive regions. Although this implementation incurs a slight overhead for mask estimation, it remains faster than the sparse implementation at higher density levels. This introduces a two-fold benefit: an upper bound to the latency of each layer and the ability to explore higher densities for critical layers, without significant latency costs.

In order to introduce this concept to training, we formulate a two-part function to map density to latency as,

$$L'_{i,sp}(d) = \begin{cases} L_{i,sp}(d) & d < d_{thresh} \\ \\ L_i(1 + s(d - d_{thresh})) & d \geq d_{thresh} \end{cases} \tag{4.14}$$

where d_{thresh} is the density threshold where beyond that, the sparse execution time becomes slower than the dense or $L_{i,sp}(d) > L_i$. A slight slope is also added to the second part to allow convergence during training, set to $s = 0.01$. The resulting $L'_{i,sp}(d)$ which is used during training, is demonstrated in (Figure 4.17). This enables the optimization process to selectively promote sparsity in layers capable of latency gains, while preventing sparsification in layers that wouldn't meet the latency threshold of L_i.

To facilitate the consideration of latency during the optimization process, some key adaptations are performed on the FLOPs optimization framework of the previous section. The target FLOPs budget f_b is replaced by a latency budget $l_b \in [0, 1]$ denoting the fraction of latency that the dynamic network should attain relative to the cumulative latency across all layers during dense

execution. Thus, equation 4.8 is modified to accommodate latency instead of FLOPs as

$$J^{L}_{sp,total} = \left(\frac{\sum_{i}^{B} L'_{\mathrm{i,sp}}}{\sum_{i}^{B} L_{\mathrm{i}}} - l_{\mathrm{b}} \right)^{2}, \tag{4.15}$$

and by modifying 4.12, the total loss is given by:

$$J = J_{\mathrm{task}} + \alpha J^{L}_{sp,network} \tag{4.16}$$

with J_{task} being the Cross Entropy Loss for semantic segmentation and α is a hyperparameter to control the difference in order of magnitude between task and sparsity loss. In FLOPs optimization, boundary terms are used to constraint sparsity rations early to avoid extreme states where the mask units never recover from Verelst and Tuytelaars (2020). However, for latency optimization that is not necessarily an issue, since operating in extreme sparsity states are a prerequisite for latency gains, as demonstrated in Figures 4.18, where high sparsity (or low density) states demonstrate speedups over dense implementation. Moreover, high density states should not be penalized more medium density states since they result in similar execution times. This is demonstrated in Figure 4.17 where the dense implementation is used densities over the breaking point (around 40%). Therefore, the boundary terms of Equations 4.10 and 4.11 proposed by Verelst and Tuytelaars (2020) are omitted for latency optimization.

4.4 Experimental Results

4.4.1 Target Networks

Both the FLOPs and latency optimization methodologies are applied on multi-branch semantic segmentation architectures, DDRNet23-slim, DDRNet23 (Hong et al., 2021) and HRNet (J. Wang et al., 2021), by converting the regular residual blocks and bottleneck blocks to dynamic sparse blocks, as described in 4.3.2. For each network, two optimization variations are considered according to the inclusion of branches for dynamic execution; (a) all branches and

Optimization Architecture	GFLOPs			Percentage of Sparse to Base Network FLOPs
	Lower Bound	Upper Bound (Base Network)	Sparse	
DDRNet23-slim	14.09	36.59	22.50	61.49%
DDRNet23-slim hb	25.87	36.59	10.72	29.30%
DDRNet23	50.40	143.60	93.20	64.90%
DDRNet23 hb	97.80	143.60	45.80	31.89%
HRNetV2-48	109.00	747.00	638.00	85.41%
HRNetV2-48 hb	301.74	747.00	445.26	59.61%

Table 4.1: FLOP bounds for spatially adaptive optimization for DDRNet23-slim, DDRNet23 (Hong et al., 2021) and HRNet (J. Wang et al., 2021) networks, for both full and high resolution branch compression.

Hyperparameter	**Value**
Input Resolution	1024×1024 for DDRNets, 1024×512 for HRNetV2-W48
Batch size	8 for DDRNets, 4 for HRNetV2-W48
Optimizer	Adam
Epochs	150
Learning rate (Backbone)	0.0001, divided by 10 in 60%,90% of training
Learning rate (Mask)	0.02, divided by 10 in 60%,90% of training
Gumbel Temp	5 annealed to 1 until 80% of training
Gumbel Noise	Disabled after 80% of training

Table 4.2: Hyperparameters used for training the sparse networks on Cityscapes (Cordts et al., 2016) dataset.

(b) high-resolution branch only, denoted as "hb". The initial stem for feature extraction from RGB input and the final segmentation heads are not included for consideration for dynamic execution and remain at 100% density, given the importance of having at least some minimum feature representation for all image areas and providing a prediction for all pixels. This leads to a specific lower bound on FLOPs for each method, leaving a corresponding number of sparse FLOPs for compression that is presented in Table 4.1.

4.4.2 Train Setting

We use pretrained checkpoints for DDRNet23-slim, DDRNet23 (Hong et al., 2021) and HRNetV2-W48 (J. Wang et al., 2021) on Cityscapes, and further train for 150 epochs under the sparse optimization framework. DDRNet models are trained with an input resolution of 1024×1024, a batch size of 8, whereas 1024x512 and batch size of 4 is used for HRNetV2 instead. We train

all the networks using Adam optimizer, with the initial learning rate for the network weights of 1e−5 and 2e−3 for the mask estimation weights, and is reduced by 10 times at epoch 60, and 90 and 120. This ensures that CNN weights remain close to the pretrained (dense) ones, while allowing the uninitialized mask estimation to explore more aggressively, but also avoid reaching states with total skipping of layers which the network can't recover from. Gumbel temperature is initially set to 5 and is gradually annealed to 1, until 80% of training, and then Gumbel noise is disabled to allow the network adapt to actual inference conditions. The preprocessing pipeline follows Hong et al. (2021) including argumentation of random cropping images, random scaling in the range of 0.5 to 2.0, and random horizontal flipping. For the comparison with state-of-the-art compression methods, we use the publicly available code for Segblocks to train the multi-branch models using the framework. As for DynConv (Verelst & Tuytelaars, 2020) we simply set $blocksize = 1$ and include the boundary sparsity loss terms using the FLOPs optimization objective. We compare our results with two spatially adaptive state-of-the-art compression methods of Segblocks (Verelst & Tuytelaars, 2023) and DynConv (Verelst & Tuytelaars, 2020). We use the publicly available code for Segblocks to train the multi-branch models for multiple τ values using the framework. Compression with DynConv (Verelst & Tuytelaars, 2020) is performed by using our framework and making certain modifications. First, a FLOPs optimization objective is used, the boundary loss terms are included in the loss function and the blocksize is set to 1.

4.4.3 Quantitative Results

Results of the FLOPs optimization are presented in Table 4.3, where f_b denotes the target budget for sparse layers and "hb" denotes compression of high resolution branch only. Introducing sparsity masks separately for each layer while skipping processing in inactive regions shows improved efficiency in exploiting spatial sparsity in multi-branch architectures. On the other hand, Segblocks (Verelst & Tuytelaars, 2023) effectiveness in compressing single-branch networks like SwiftNet (Orsic et al., 2019), as reported in the paper (Verelst & Tuytelaars, 2023), does not directly transfer to the multi-branch domain (Figure 4.19). Moreover, grouping

Δ FLOPs %	Method	Δ Latency %	Δ mIOU%
	DDRNet23-slim (Hong et al., 2021)		
-30%	DynConv (Verelst & Tuytelaars, 2020) ($\tau = 0.4$)	10.35	-2.67
	Segblocks (Verelst & Tuytelaars, 2023) ($\tau = 0.6$)	-6.86	-1.34
	Ours $f_b = 0.5$	11.31	**-0.59**
	Ours $l_b = 0.8$	**-7.85**	-1.29
-20%	DynConv (Verelst & Tuytelaars, 2020) ($\tau = 0.65$)	10.35	-0.73
	Segblocks (Verelst & Tuytelaars, 2023) ($\tau = 0.7$)	-4.13	-0.85
	Ours hb $f_b = 0.3$	-0.37	**-0.24**
	Ours hb $l_b = 0.8$	**-11.1**	-0.33
	DDRNet23 (Hong et al., 2021)		
-30%	DynConv (Verelst & Tuytelaars, 2020) ($\tau = 0.4$)	7.39	-0.92
	Segblocks (Verelst & Tuytelaars, 2023) ($\tau = 0.6$)	11.15	-1.37
	Ours $f_b = 0.5$	5.52	**-0.21**
	Ours $l_b = 0.8$	**-7.31**	-0.90
-20%	DynConv (Verelst & Tuytelaars, 2020) ($\tau = 0.65$)	7.39	-0.52
	Segblocks (Verelst & Tuytelaars, 2023) ($\tau = 0.7$)	-7.92	-0.89
	Ours hb $f_b = 0.3$	-6.72	**-0.21**
	Ours hb $l_b = 0.6$	**-12.73**	-0.31
	HRNetV2-W48 (J. Wang et al., 2021)		
-40%	DynConv (Verelst & Tuytelaars, 2020) ($\tau = 0.6$)	18.49	-2.51
	Segblocks (Verelst & Tuytelaars, 2023) ($\tau = 0.4$)	**-30.71**	-6.27
	Ours $f_b = 0.5$	-6.64	**-0.80**
	Ours $l_b = 0.8$	-11.48	-1.09
-30%	DynConv (Verelst & Tuytelaars, 2020) ($\tau = 0.7$)	18.49	-1.41
	Segblocks (Verelst & Tuytelaars, 2023) ($\tau = 0.6$)	**-16.00**	-3.96
	Ours hb $f_b = 0.5$	-2.73	**-0.27**
	Ours hb $l_b = 0.8$	-12.54	**-0.27**

Table 4.3: Table presenting the result of optimization of DDRNet23-slim, DDRNet23 and HRNetV2-48 multibranch architectures, evaluated on the validation set of Cityscapes dataset, under various FLOPs reduction regimes.

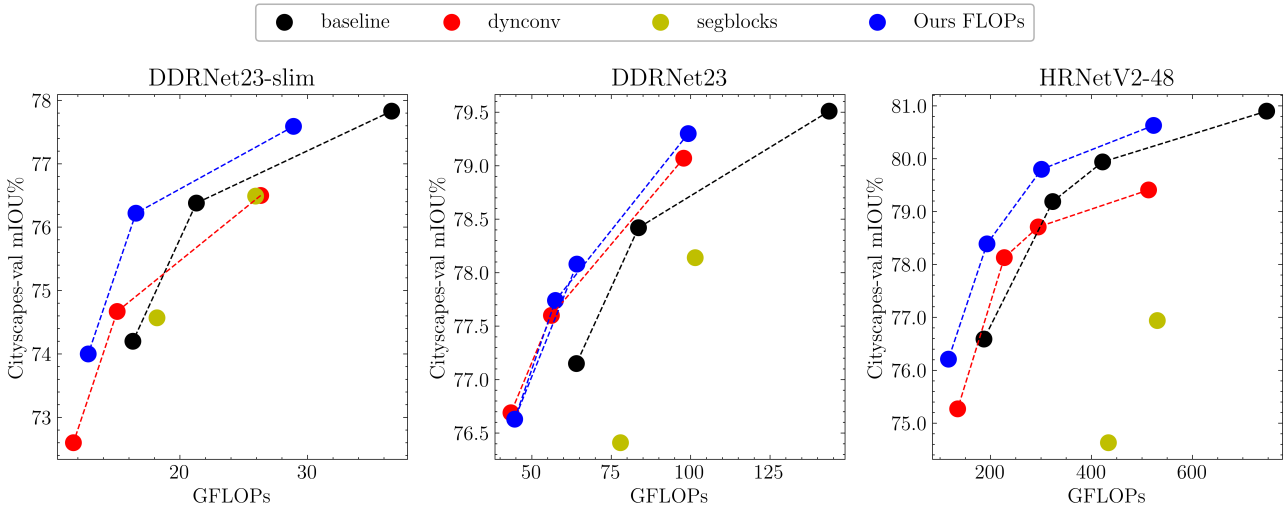


Figure 4.19: FLOPs optimization of DDRNet23-slim (Hong et al., 2021), DDRNet23 (Hong et al., 2021) and HRNetV2-48 (J. Wang et al., 2021) multibranch architectures compared to SoA compression methods using spatial sparsity. Multiple connected points are generated by evaluating the full resolution models at different scales.

image areas in blocks rather than individual pixels like DynConv (Verelst & Tuytelaars, 2020), is beneficial in workloads which include high resolution images such as semantic segmentation.

In all cases, optimising using FLOPs target results in a solid reduction of computations, however not always directly translating to latency. To demonstrate that, results of optimization by considering the actual latency of networks are presented in Table 4.3. We denote l_b as the target latency budget of the sparse layers for our optimization framework. Latency aware optimization achieves improved execution times of DDRNets compared to latency agnostic optimization and improved efficiency when compared to other compression methods at similar compression rates. While Segblocks (Verelst & Tuytelaars, 2023) achieves higher reduction in latency of HRNetV2-48 (J. Wang et al., 2021), accuracy deteriorates much more than our approach. Moreover, applying the optimization only on the high resolution branches rather than the full network demonstrates increased efficiency, highlighting the existence of more redundant computations in it. In general, 20% to 40% of total FLOPs can be reduced with minor impact on accuracy, depending on the sensitivity of the network.

To explore how our latency optimization framework can generalize across other GPU devices, we also evaluate it on an NVIDIA Jetson NX embedded GPU, by using the individual benchmark functions for this device such as those of Figure 4.18 (e-h) and perform the final benchmarks on the device. The results obtained are similar (Table 4.4), confirming the successful transfer and effectiveness of the latency optimization technique. Moreover, we observe that optimizing for latency across the entire network proved most effective for the RTX 3060ti device, whereas optimization of high-resolution layers only, showcased superior performance for the Jetson NX device. This variance in performance highlights the crucial role played by the device-tailored optimization through using actual device-specific latency profiles.

4.5 Ablation Studies

The section dedicated to ablation studies within the optimization framework delves into analyzing the individual components, hyperparameters and design choices and measure their impact

Method	Δ Latency %	Δ mIOU%
NVIDIA RTX 3060ti		
$l_b = 0.8$	-7.31	-0.90
hb $l_b = 0.8$	-10.20	**-0.22**
hb $l_b = 0.6$	**-12.73**	-0.31
NVIDIA Jetson NX		
$l_b = 0.8$	-7.50	-0.47
hb $l_b = 0.8$	-5.25	-0.34
hb $l_b = 0.6$	-10.07	-0.26

Table 4.4: Latency optimization results for DDRNet23 (Hong et al., 2021) network performed on two different GPU devices: a desktop NVIDIA RTX 3060ti and an embedded NVIDIA Jetson NX.

on the overall performance.

4.5.1 Block Size

The choice of block size plays an important role in the current optimization framework for two reasons. Firstly, it introduces structure to the data, which is essential to mitigate the overheads for sparse execution in GPU devices, allowing space for the realization of latency gains on GPU devices using Sbnet sparse operations. Secondly, grouping image pixels in a region seems to have a beneficial regularization effect in training as seen in Table 4.5. The 1×1 2D convolution operations used by mask estimation layers might be too isolated to reason about the potential upside of processing each pixel, whereas combining them with a spatial operation such as pooling, results in more effective reasoning about the region.

Block size	mIOU%	Latency (ms)
1	76.06	112.11
8	76.12	41.90
32	76.85	20.79
64	**77.00**	20.65
128	76.89	**20.59**

Table 4.5: DDRNet23-slim optimized with target budget $f_b = 0.50$ with varying blocksizes in full image resolution. Latency measured on a NVIDIA RTX 3060ti.

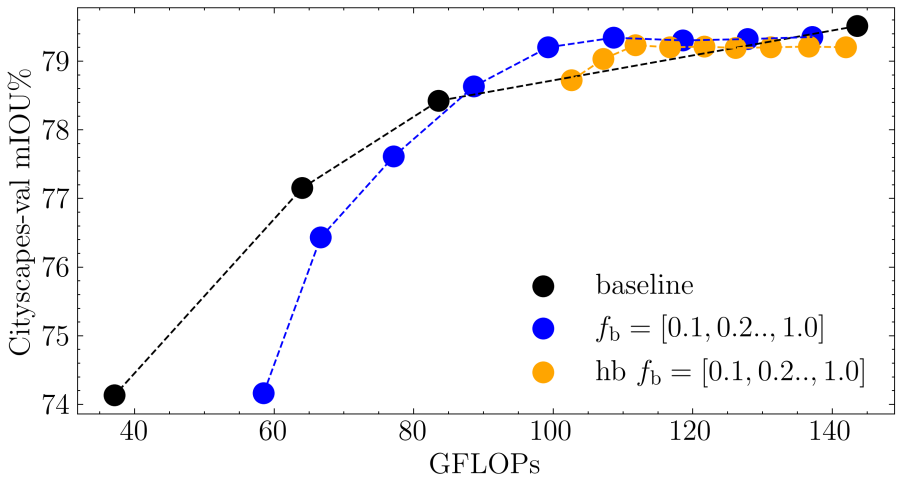


Figure 4.20: Performance of sparse DDRNet23 (Hong et al., 2021) under various target budgets $fb \in [0.1, 0.2.., 1]$

4.5.2 Target budget

We conduct an ablation study on the target budget of the optimization process, to investigate the relationship between network sparsity, model complexity, and predictive capability. The analysis is also helpful to get insights for the most efficient trade-offs between computational efficiency and performance. In the DDRNet-23 case (Figure 4.20), extreme sparsities in the full network optimization lead to sub-optimal states, while the most efficient compression is achieved when targeting budgets within the range of 0.5 to 0.7. On the other hand when optimizing the high-resolution branch only, the effective target budgets fall within the interval of 0.2 to 0.4.

no boundary loss term	timing slope	mIOU%
✗	✗	79.78
✗	✓	79.99
✓	✗	80.51
✓	✓	80.63

Table 4.6: The impact of removing the initial boundary loss term and addition of the slope in the timing function, for latency optimization of the high resolution branch of HRNetV2-48 using $l_b = 0.8$.

4.5.3 Slope and loss boundary terms

This section explores the impact of design choices of the training process when moving from FLOPs to latency optimization (Table 4.6). Omitting the boundary loss term allows the optimization to explore the full space from the start, therefore reaching to more extreme sparsities/densities when necessary. The addition of slope in the timing function $L'_{i,sp}(d)$ favors the mask estimation to push higher densities when the density threshold for practical gains is surpassed, but shows a smaller impact in practice.

4.5.4 Qualitative Results

The evaluation of dense and sparse methods stands is a useful benchmark for pixel-wise segmentation task performance. Figures 4.22 and 4.23 present a comparative analysis showcasing disparities and similarities between the estimated segmentation maps of pretrained dense networks and our optimized sparse ones. It is observed that the quality attained through sparse methods closely parallels that of dense methods in the largest part of the image. Upon meticulous examination, it becomes evident that while sparse methods excel in accurately recovering the main and critical structures, in some instances some minor details, such as poles, may be overlooked. This nuanced difference shows a small difference between the methodologies, yet does not detract significantly from the overall fidelity of the reconstructed structures, providing a favorable trade-off between performance and latency.

A visual comparison between FLOPs and latency optimization is demonstrated in Figure 4.24. Both methods, albeit yielding comparable overall performance, exhibit some basic differences. Notably, FLOPs optimization can allow higher mask granularity which does not directly translate into an observable or tangible enhancement in overall performance compared to latency optimization, which requires increased grouping of pixels. This showcases the importance of introducing the desired metric for optimization, steering the training process to generate a very similar output which exhibits the desired behavior.

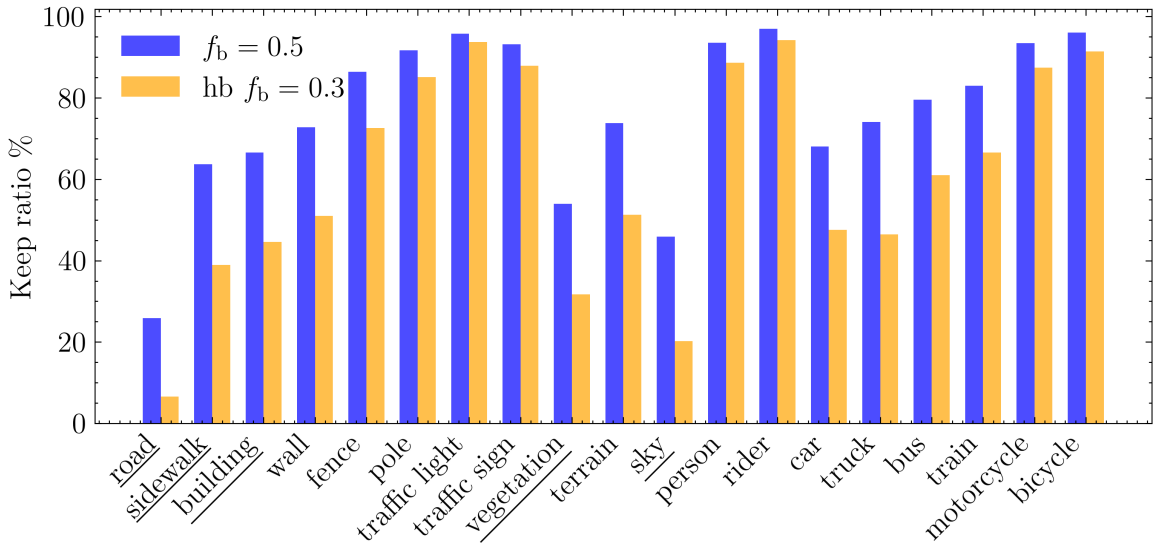


Figure 4.21: Percentages of active pixels for each class of sparse DDRNet23 Hong et al., 2021 network. Pixels of *coarser* classes (underlined) are more frequently masked than those from *finer* classes.

4.6 Discussion

4.6.1 Class Sparsity Analysis

As we explore the connection between sparsity and multiple classes, it becomes evident that the impact of sparsity is far from uniform, as seen in Figure 4.21. Some classes exhibit high sparsity ratios (low keep rate%), indicating that only a small portion of the available features contribute meaningfully to their representation. This phenomenon can be attributed to inherent characteristics of certain classes that render them naturally less cluttered or complex. On the other hand, certain classes demonstrate lower sparsity ratios (high keep rate%), where a higher portion of their features are actively engaged in their representation. This analysis proves the both the existence of redundancy of computations within the networks, which is exploited in order to reduce the computational budget, and the necessity of a masking strategy to select the optimal regions to skip.

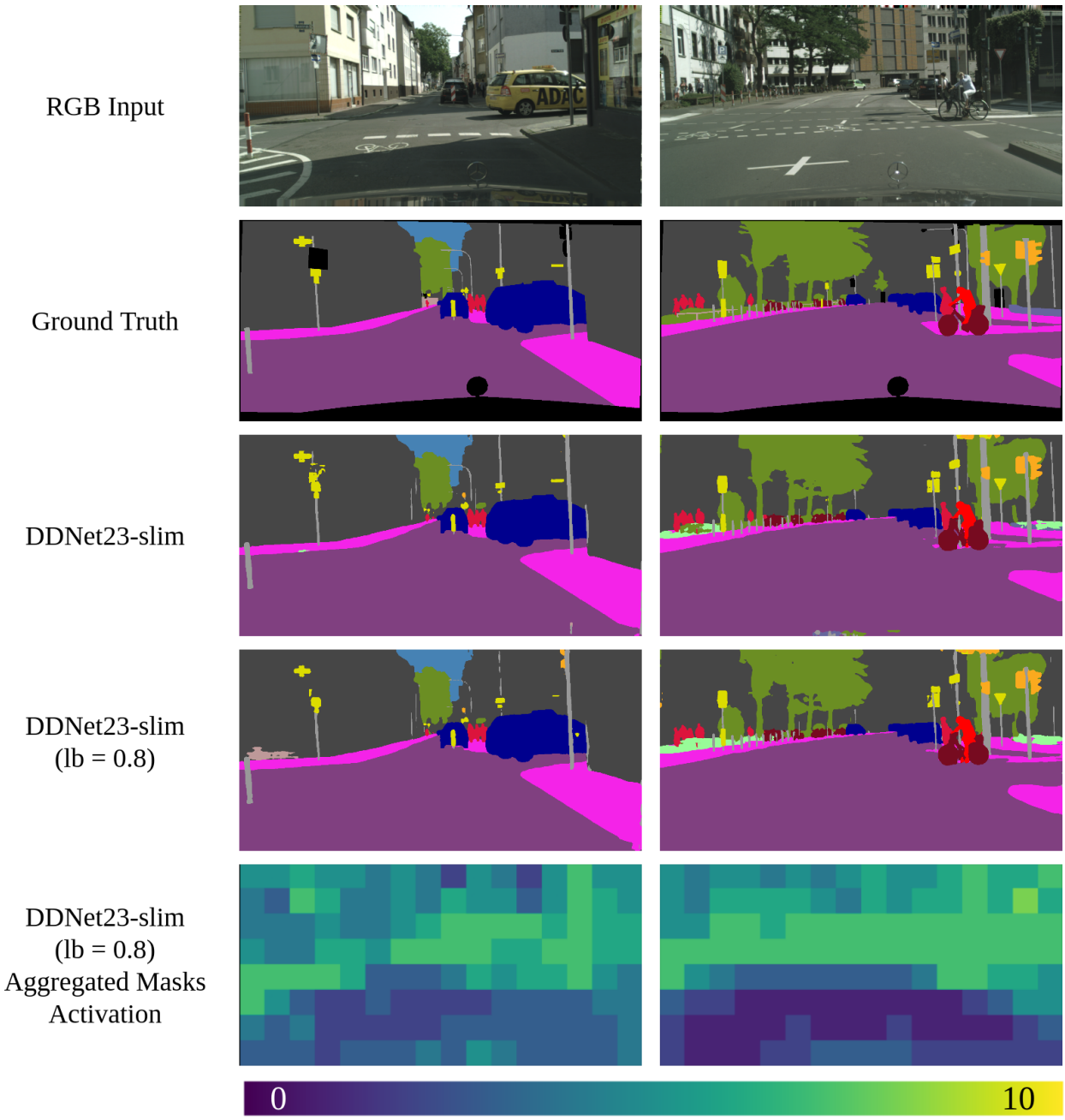


Figure 4.22: Qualitative evaluation for latency optimized DDRNet23-slim for $lb = 0.80$ compared to their dense counterpart. Rows from top to bottom contain RGB input, ground truth, corresponding network dense output, sparse output, summed cost maps and scale colorbar.

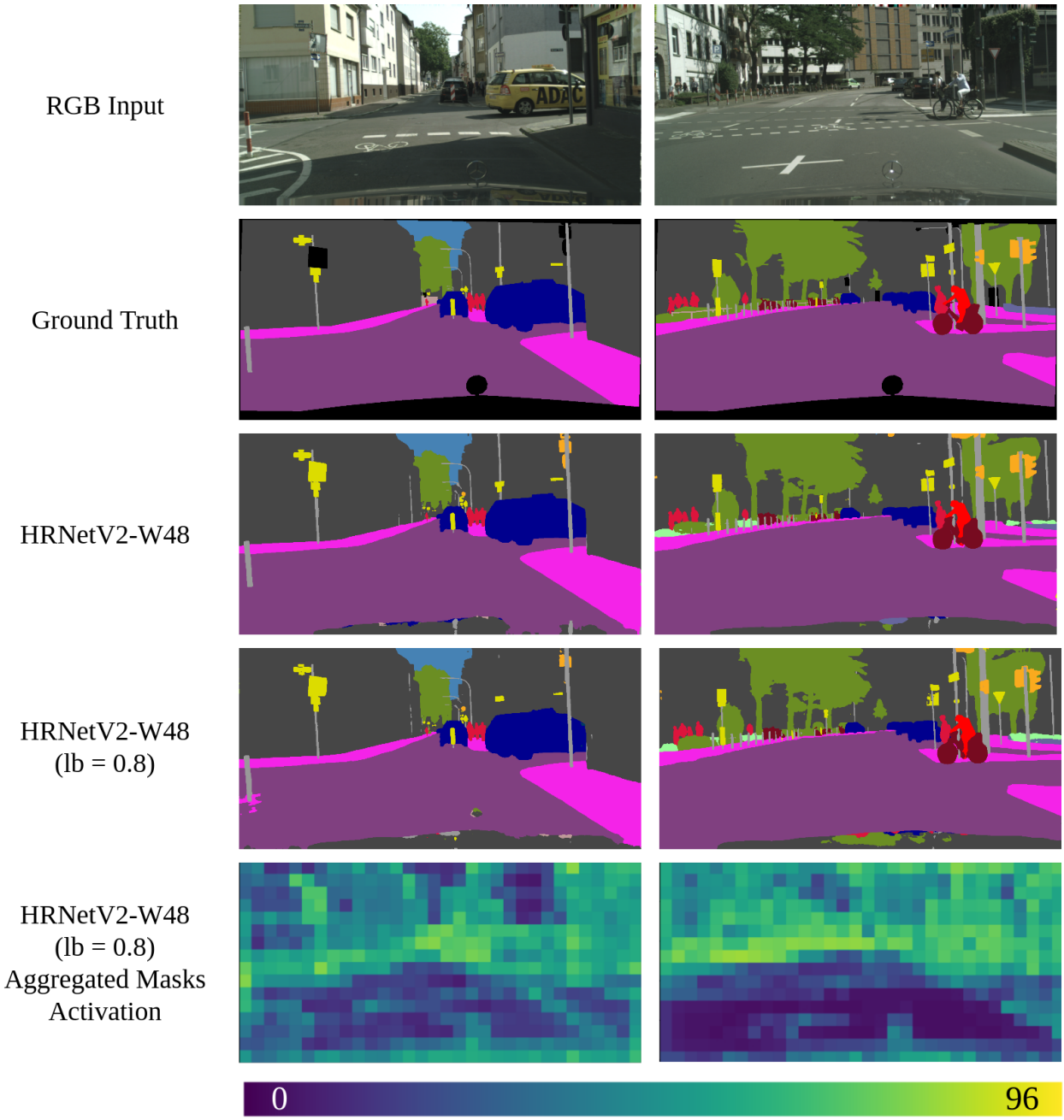
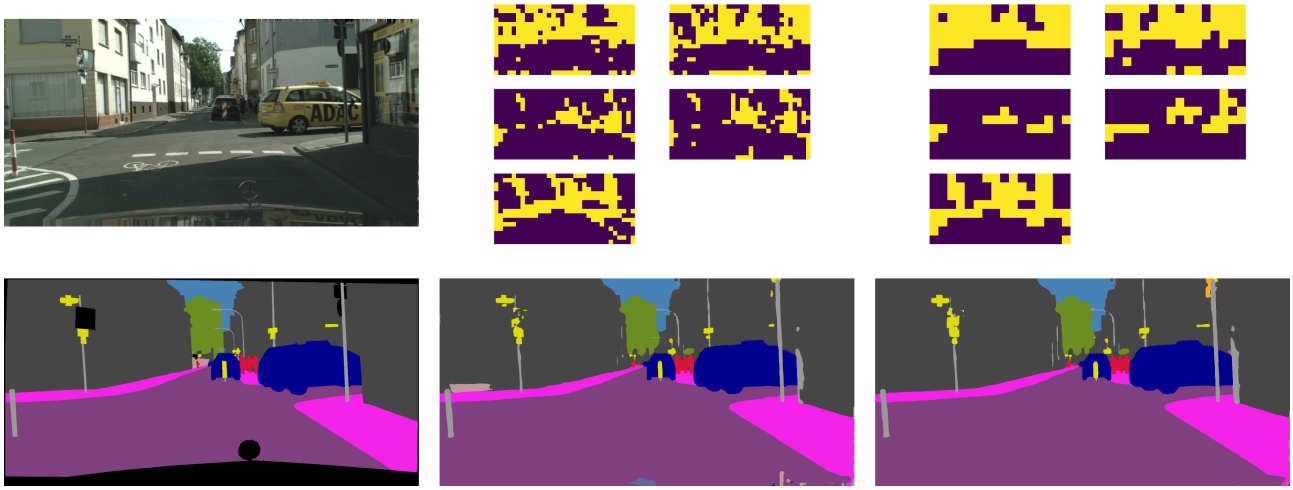
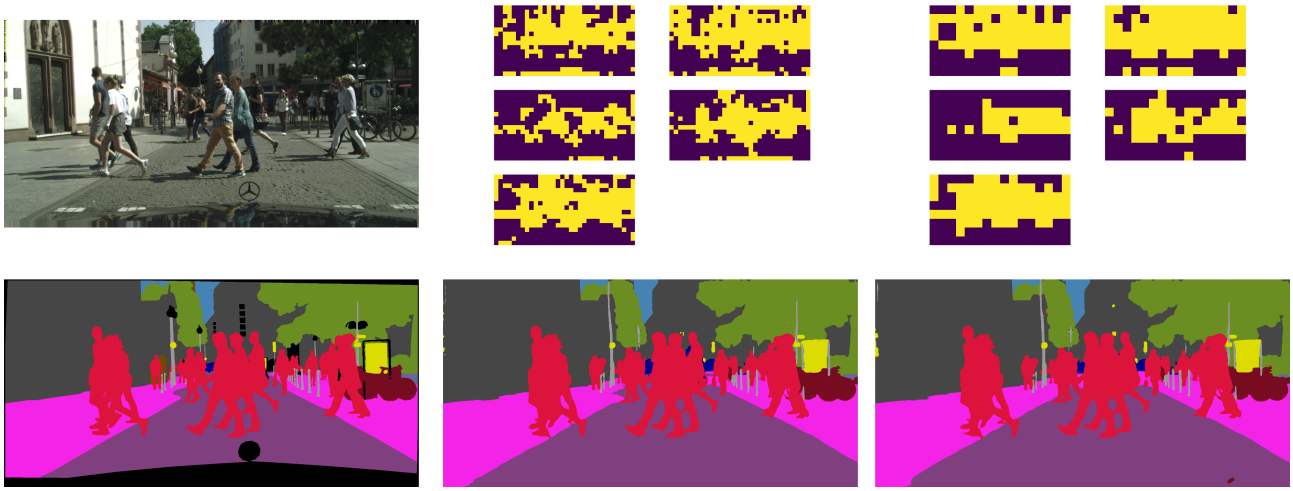


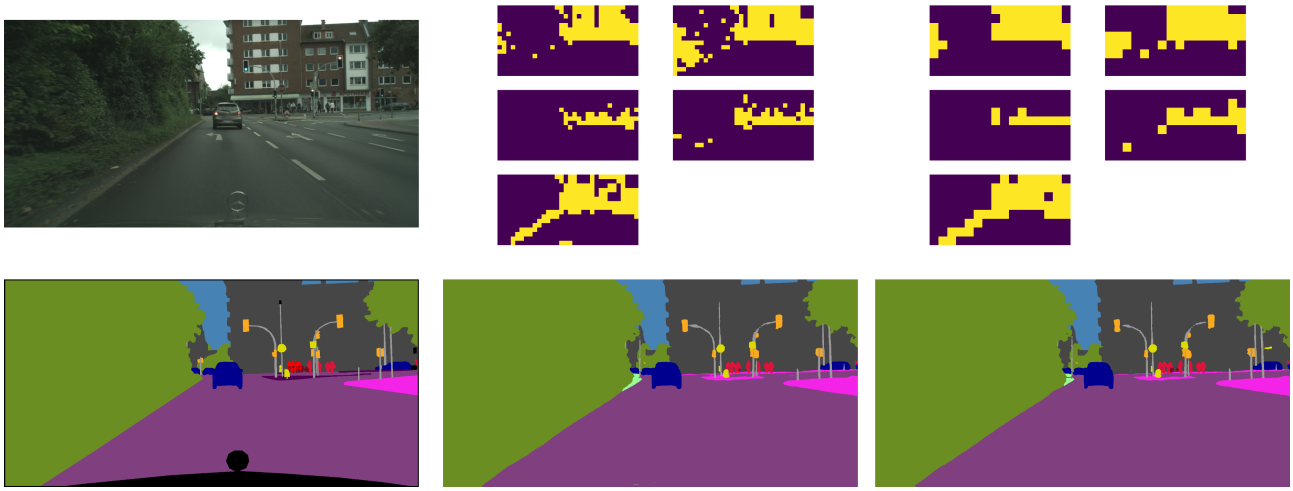
Figure 4.23: Qualitative evaluation for latency optimized HRNetV2-W48 (b) for $lb = 0.80$ compared to their dense counterpart. Rows from top to bottom contain RGB input, ground truth, corresponding network dense output, sparse output, summed cost maps and scale colorbar.



(a)



(b)

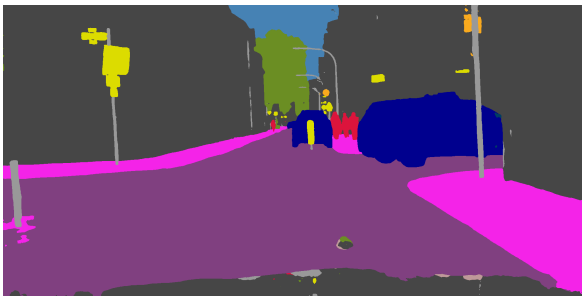


(c)

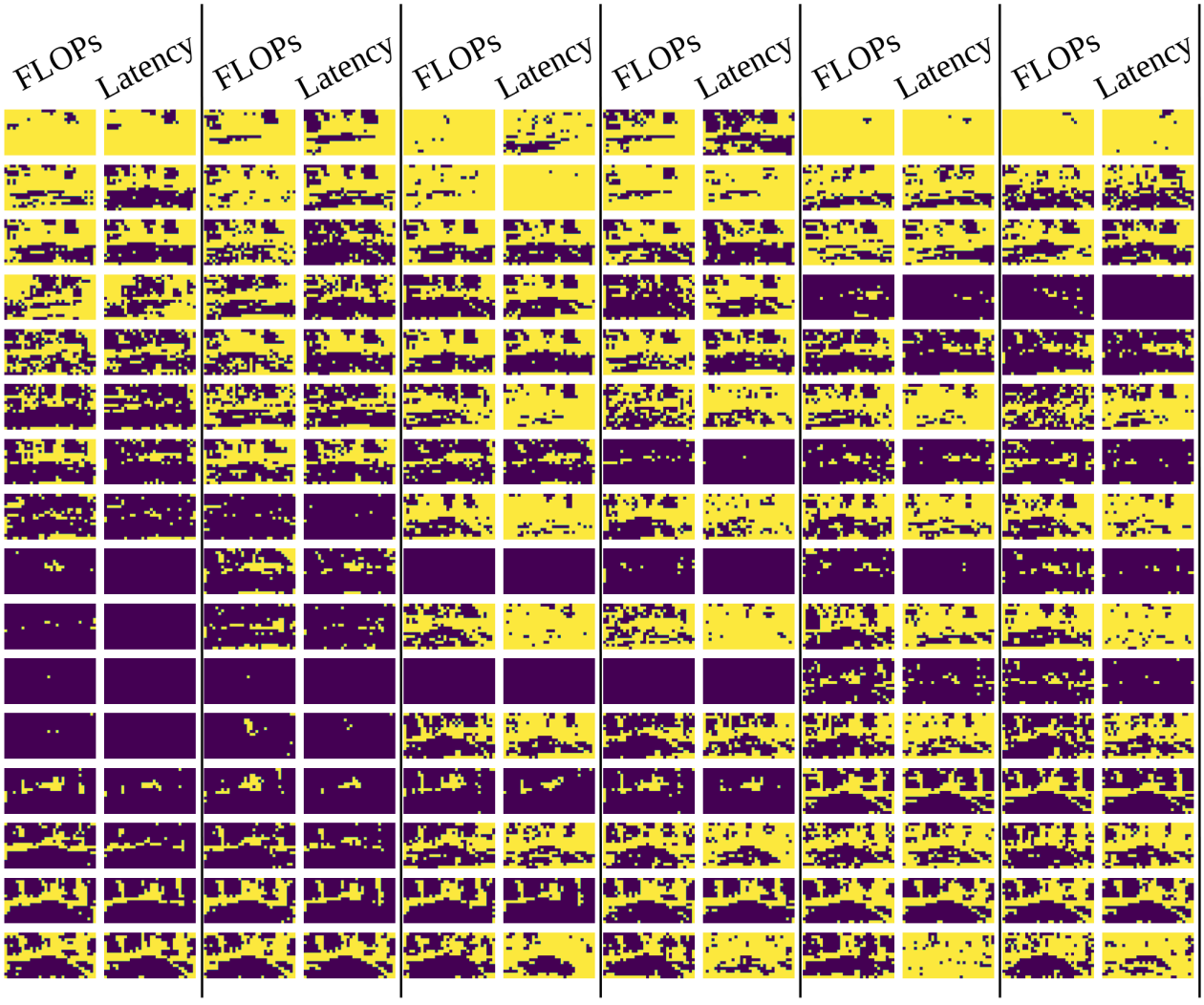
Figure 4.24: Cityscapes samples (left column) with output from DDRNet23-slim hb optimization under 80% FLOPs (middle) and 80% latency (right) targets. Each sample contains the RGB image (top left), annotated ground truth (bottom left), and middle and right colums show the per-layer estimated binary masks (top) and output segmentation map (bottom).



(a) HRNetV2-W48 $fb = 0.50$



(b) HRNetV2-W48 $lb = 0.50$



(c)

Figure 4.25: Sparsity mask inspection for flops (HRNetV2-W48 fb=0.50) and latency (HRNetV2-W48 lb=0.80) optimization of the same model. Figures (a) and (b) contain the semantic segmentation map outputs, and (c) demonstrates the per-layer sparsity mask of each network.

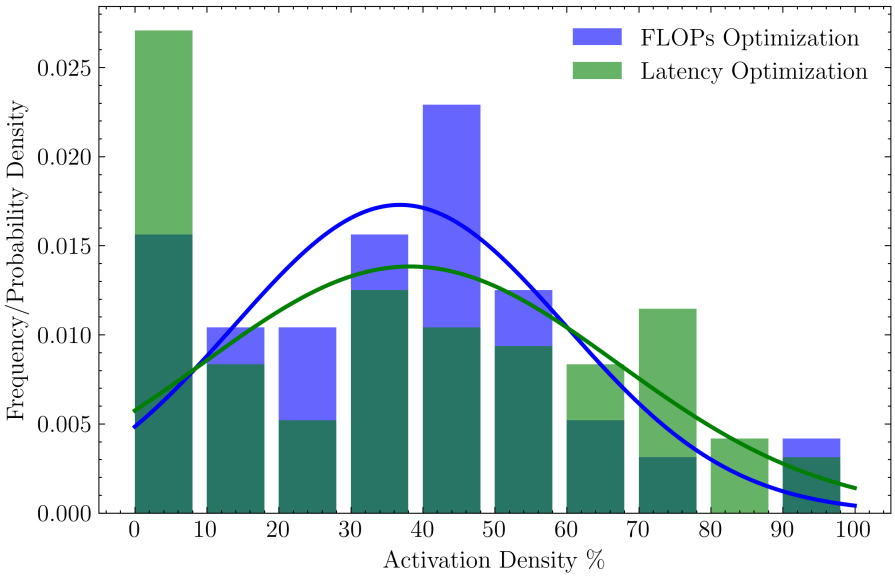


Figure 4.26: Histogram with actiation rates for sparse layers of HRNetV2-48 optimized for FLOPs ($f_b = 0.5$) and for latency ($l_b = 0.8$).

4.6.2 Latency vs FLOPs Optimization

Both FLOPs and latency optimization methodologies attempt to minimize a target budget by skipping computations in spatial locations. However, there is an important difference in the way computations are distributed between layers. When optimising for FLOPs, the activation rates follow a distribution closer to mean activation which is usually the target budget, whereas when optimizing with a latency objective, activation rates tend to spread to more extreme values in both directions (Figure 4.26). A detailed laper-per-layer comparison between activations of FLOPs and latency optimized HRNetV2-W48 with $lb = 0.80$, is shown in Figure 4.25, showcasing the more extreme value distribution of latency optimization. The reason is that latency gains for a layer can be achieved below a certain density threshold and are maximized closer to zero density. On the other hand, if this threshold is exceeded, the dense implementation followed by binary masking can be used instead, which requires equal time as if the activation rate was 100%. Therefore, in latency optimization, activation rates closer to extremes are generally more optimal, since either reduced latency or increased density can be achieved, a behavior introduced in training by employing the leaky timing function.

	DynConv (Verelst & Tuytelaars, 2020)	SegBlocks (Verelst & Tuytelaars, 2023)	Ours
Sparse Decisions	N	1	N
Computation Scheme	Skip/Sparse	Low Resolution	Skip/Sparse
Mask Granularity	Pixel	Block	Block
Training Target	Computational Budget	Computational Budget	Computational or Latency Budget
Pareto optimal FLOPs Efficiency	✓	✓	✓
Pareto optimal Latency Efficiency	✗	✗	✓

Table 4.7: Feature table comparing the key elements of the relevant spatially adaptive methods with our approach on dual branch CNNs for semantic segmentation.

4.6.3 Comparison to SegBlocks

Given the relevance of the proposed work with Segblocks (Verelst & Tuytelaars, 2023), we discuss some interesting conclusions. Segblocks takes advantage of the over-parameterized nature of single branch architectures for semantic segmentation, which try to accommodate both the deep features and fine details in all feature maps, to *essentially generate two branches of computation*. However, recent multi-branch architecture have *already accounted for that* by defining multiple paths of computation, and therefore, redundancy is less apparent in those approaches constituting compression a more challenging task. Moreover, multi-branch networks don't seem to benefit from having the two paths of computation of Segblocks in the application of sparsity, since it practically doubles the number of branches, creating potentially inefficient overlaps of computation in some resolutions. Finally, Table 4.7 presents a comparison of the key elements of relevant spatially adaptive methods of SegBocks and DynConv and our proposed approach, specifically for dual-branch CNNs applied to semantic segmentation.

4.7 Conclusions

This chapter presents an optimization methodology that introduces spatial sparsity to semantic segmentation CNNs, by skipping computations on image regions on a per-layer basis, to enable performance to FLOPs and performance to latency trade-offs. The level of sparsity is controlled

by a specified target budget, which is provided as an objective for the optimization process. The key takeaways from this work are:

- Our method can generate spatially adaptive variants of semantic segmentation CNNs, which define a favorable pareto front in both latency and FLOPs space for state-of-the-art multi-branch CNNs for semantic segmentation.
- The introduction of latency to the optimization process is achieved by employing a latency profiling of the network, resulting in a device-tailored optimization.
- Results demonstrate increased performance-latency efficiency when compared to previous hardware agnostic theoretical optimization.
- There are significant overheads to implement sparse execution in modern hardware, which are often discounted when considering only the number of FLOPs, which call for specific design choices to mitigate their impact.

Yet, the suggested approach carries constraints and brings to light pivotal considerations when operating within this domain.

- Latency optimization is dependent on both the existence of a sparse implementation for the target hardware and it's ability to achieve faster execution compared to default dense execution at some sparsity level. The effectiveness of the sparse implementation defines the space for the potential gains under sparsity.
- The current framework supports only residual style blocks, since fulfill the requirement for equal input and output channels, which is required by the sparse scatter operation. This limits the potential network candidates of the optimization framework in it's current form, to those containing such blocks.
- The produced spatially adaptive networks are generated by specifying a target FLOPs or latency budget in training and result in a dynamic behavior reflecting this specific target budget. Therefore, it is not possible to adjust the desired budget in inference without re-training for a different budget.

Chapter 5

Memory-aware Spatially Adaptive Inference for CNNs

This chapter addresses the third research question: *How can dynamic inference for multiple target computational budgets be achieved while focusing on optimizing memory usage?*. Part of the work presented in this chapter is based on the conference paper: *Budget-aware Dynamic Spatially Adaptive Inference*, which was published and presented in the British Machine Vision Conference (BMVC) 2024 [P3].

5.1 Image Classification

Image classification is a fundamental task in the field of computer vision, with the objective of assigning a label or category to an input image, based on its visual content. It has significant importance in a wide range of applications, such as healthcare, autonomous vehicles, agriculture, security, and retail. Image classification also serves as a foundational task for various other computer vision applications. Backbones trained for large image classification datasets are not only effective classifiers, but provide essential context for more complex tasks like object localization and scene understanding. Since such backbones are widely used among the community, core progress in computer vision model performance is greatly dependent on advancements in

image classification. The following section explores the evolution of image classification methods from their early iterations to modern advancements, revealing a diverse array of models characterized by varying performance levels, computational costs, and requirements.

5.1.1 Evolution of Methods

Prior Art

The advent of deep learning methods in the early 2010s signaled a transformative era in the field of computer vision. AlexNet (Krizhevsky et al., 2012) marked the beginning of the deep learning era for image classification. It revolutionized the field with its pioneering architecture, which included fundamental components such as convolutional layers and rectified linear units (ReLU), which remain integral building blocks to state-of-the-art neural networks until today. AlexNet significantly outperformed traditional methodologies by 11% in classification accuracy in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2015) uncovering signs of the capabilities of CNNs for image understanding tasks. Its success not only showcased its superiority over traditional approaches but also sparked widespread adoption of CNNs for image classification tasks, prompting researchers to explore deeper and more complex network architectures. VGG (Simonyan & Zisserman, 2015) followed with a deeper architecture, comprising of a series of convolutional layers with small 3x3 filters followed by max-pooling layers. Despite its computational cost due to the large number of layers, VGG achieved remarkable accuracy on ImageNet (J. Deng et al., 2009) benchmark dataset, outperforming AlexNet by 8% in classification accuracy. Using such deep architecture with up to 19 layers, demonstrated the importance of depth in capturing hierarchical features from images, laying the groundwork for subsequent deeper architectures.

However, while adding more layers theoretically increases the modeling capacity of a network, it does not always practically translate to improved performance. This happens due to vanishing gradient problem, which refers to the decrease in accuracy observed with the addition of more layers to a neural network. ResNets (He et al., 2016) addressed the challenge of training very

deep neural networks by mitigating the vanishing gradient problem, by introducing skip (residual) connections which enabled the direct flow of information from one layer to the next layers. This allowed for the creation of much deeper networks (up to 152 layers) without suffering from vanishing gradients or degradation in performance. Moreover, ResNet architectures are highly modular and scalable, with backbone variants like ResNet-50 and ResNet-101 widely adopted for various computer vision tasks beyond image classification.

As deep learning gained prominence in computer vision, the demand for efficient and scalable image classification models from research and industry grew significantly. Early architectures like AlexNet and VGG, while groundbreaking in their time, posed challenges in terms of computational budget and deployment on resource-constrained devices. This led to the emergence of methods focusing on more optimized deep network designs as a response to these challenges.

ResNeXt (S. Xie et al., 2017) extends the ResNet architecture by introducing grouped convolutions in the residual block. It uses a "split-transform-merge" strategy (branched paths within a single module) similar to an Inception (Szegedy et al., 2015) module, which aggregates a set of transformations. Compared to the original Residual Block, it exposes a new dimension, cardinality (size of set of transformations), as an essential factor in addition to depth and width. This design increases model capacity and enables more diverse feature representations without significantly increasing computational complexity.

Towards Efficiency

MobileNets (A. G. Howard et al., 2017), (Sandler et al., 2018), (A. Howard et al., 2019) are a family of lightweight convolutional neural networks (CNNs) designed for efficient deployment on mobile and embedded devices with limited computational resources. To achieve that, they adapted the original residual block of ResNet to an inverted residual (MBConv) block, by replacing regular convolutions with depthwise separable ones, which decompose standard convolutions into depthwise and pointwise convolutions. This separation significantly reduces computational complexity while preserving representational capacity, making MobileNets well-suited for resource-constrained environments. Towards the same goal, ShuffleNets (X. Zhang et

al., 2018), (Ma et al., 2018), introduce the concept of channel shuffling to enhance information flow and reduce computational cost in convolutional neural networks. The key contribution lies in the integration of channel shuffle operations, which enable feature maps to be split into groups, shuffled, and concatenated to facilitate cross-group information exchange. By leveraging channel shuffling, ShuffleNets achieve efficient parallelization and reduced communication overhead, making them particularly suitable for deployment on resource-constrained devices. Variants like ShuffleNetV2 further improve efficiency by incorporating additional optimizations such as pointwise group convolutions and channel-wise scaling.

Finally, in a more structured and scalable approach towards efficient models, EfficientNet (Tan & Le, 2019) introduced the concept of compound scaling, optimizing model depth, width, and resolution simultaneously. This is justified by the intuition that the larger the input image gets, then the network needs more layers to increase the receptive field and more channels to capture more fine-grained patterns of the image. Therefore, EfficientNet models are systematically scaled up by increasing the number of layers, width of the network, and resolution of input images in a balanced way. Using the inverted residual blocks from Mobilenetv2 as main building blocks, this compound scaling strategy ensures that each dimension contributes to improved model performance without disproportionately increasing computational cost. The convergence of these efforts reflects a shift towards more practical and accessible deep learning solutions, enabling advancements in areas such as real-time inference, edge computing, and deployment in resource-constrained environments.

Current Art

By 2021, CNNs were established as the go-to method for vision tasks due to their ability to effectively capture spatial hierarchies in images. However, Transformers (Vaswani et al., 2017), initially designed for natural language processing tasks, were adapted and applied to image classification with remarkable success (Dosovitskiy et al., 2020). Unlike CNNs, which rely on convolutional layers to extract features hierarchically, Transformers employ self-attention mechanisms to capture global dependencies within the input data. This enables Transformers

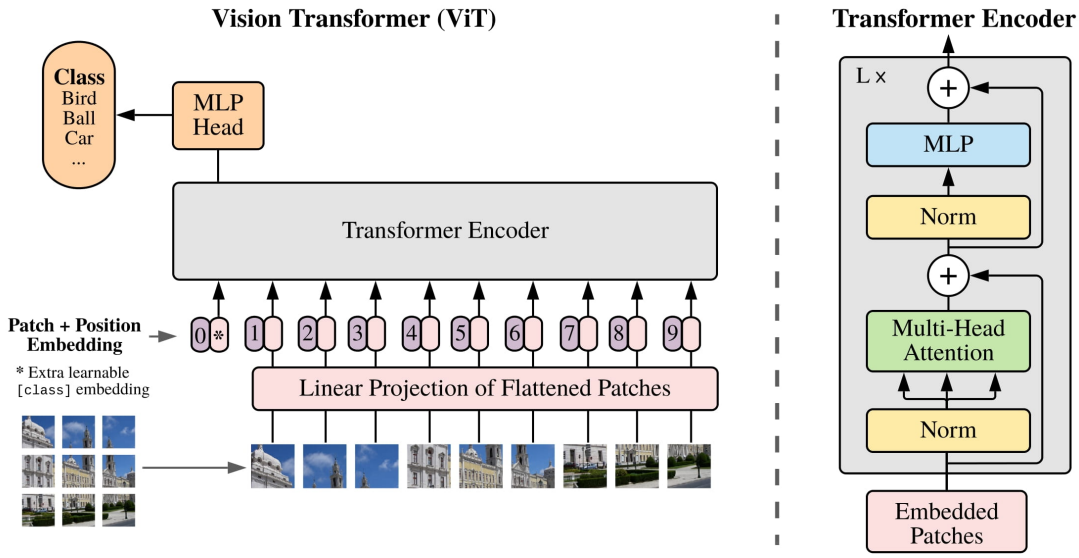


Figure 5.1: Figure with ViT architecture (left) from the original paper Dosovitskiy et al. (2020) along with the architecture of the Multi-Head attention block (right).

to process images holistically, considering all pixels simultaneously rather than sequentially, allowing them to capture both global and local context at all times. Therefore, they can effectively model long-range dependencies in images, leading to superior performance in image understanding tasks. The innovative features of Vision Transfomer (ViT) (Dosovitskiy et al., 2020), including positional encoding, patch splitting, and the utilization of Multi-Head Attention (MHA) blocks, established a foundational framework (Figure 5.1) for subsequent research in the field, since these concepts have become fundamental building blocks in modern works.

The introduction of Transformers in vision tasks marked a significant breakthrough, yet their implementation typically comes with increased computational costs which scales quadratically with model size. The reason is that in self-attention operation, each token in the input sequence attends to every other token in the sequence to compute the attention scores. This process involves calculating pairwise similarities between tokens, which results in a quadratic increase in the number of computations required as the sequence length grows, which is common in vision tasks where images are represented as grids of pixels. This led to the emergence of hybrid methods aiming to strike a balance between computational efficiency and performance. To address that, Swin Transformer Z. Liu et al. (2021) proposes a hierarchical Transformer whose representation is computed with Shifted windows, facilitating a hierarchical feature representation which comes with linear computational complexity with respect to input image size. Alterna-

tively, in an approach to reduce the redundant input tokens, M. Chen et al. (2023) proposed a coarse-to-fine vision transformer (CF-ViT), with two-stage inference. First, the input image is split into a small-length token sequence in an attempt to classify "easy" samples, using smaller computational cost. If not confident enough, patches containing critical information identified and further re-split in to perform inference with the same network but at a finer granularity. In conclusion, Transformer-based architectures managed to surpass CNNs in achieving state-of-the-art performance across various vision tasks, marking a significant shift in the landscape of computer vision research and applications.

However, the emergence of Transformer architectures sparked an on-going debate among researchers between the effectiveness of CNN and Transformers for computer vision. Despite the revolutionary impact of the Transformers, they still face challenges in capturing local spatial information crucial for visual localization tasks and rely on positional encoding information. On the other hand, CNNs inherently include spatial information in their architecture but struggle with long-range dependencies and global context understanding. In an attempt to bridge the two architectures, ConvNeXt (Z. Liu et al., 2022), attempts to harness the strengths of CNN while incorporating Transformer-inspired mechanisms. An initial patchify step, including a 4x4 convolution with stride of 4, produces non-overlapping patches, similar to transformer tokens. To achieve long-range dependencies in a pure convolutional design, inverted residual blocks containing depth-wise separable layers with 7x7 convolution kernels are used, resembling the receptive field of self-attention operation. This leads to a purely convolutional architecture, which implements the key components that propelled the success of Transformers, but retaining the computational efficiency and inherent locality of CNNs.

In conclusion, the evolution of methods for visual understanding in the deep learning era has witnessed significant advancements, from early CNNs like AlexNet to the recent breakthroughs with vision Transformers. Each model brings its unique contributions, addressing challenges in accuracy, efficiency, and scalability. However, with such a diverse landscape, there is no one-size-fits-all approach, since various applications have different requirements and limitations.



(a) MNIST (b) CIFAR 10, CIFAR 100

Figure 5.2: Samples from (a) MNIST L. Deng (2012) and (b) CIFAR 10 and CIFAR 100 Krizhevsky (2012) datasets.

5.1.2 Datasets

Image classification datasets comprise of images and a set of labels, and each image can have one or more labels. While the image-label pair is a requirement for facilitating end-to-end training of models, the most challenging part is to ensure the dataset is bias-free, balanced and has an adequate number of samples. Several datasets have been popular for image classification and can be roughly separated according to their number of samples, in medium scale (a)-(e) and large scale (f)-(g) ones.

(a) **MNIST** (Modified National Institute of Standards and Technology) (L. Deng, 2012) is a dataset of handwritten digits, consisting of 60,000 training images and 10,000 test images, each of which is a grayscale image of size 28x28 pixels. It is commonly used as a benchmark dataset for evaluating machine learning models for image classification tasks. (b) **Fashion-MNIST** (Xiao et al., 2017) is a dataset similar to MNIST with equal number of samples but consists of images of clothing items instead of handwritten digits. (c) **Caltech-256** (Griffin et al., 2007) is a dataset containing 30,607 images across 256 object categories. Each category contains at least 80 images, and the dataset covers a wide range of objects, scenes, and animals. (d) **CIFAR-10** and (e) **CIFAR-100** (Krizhevsky, 2012) are benchmark datasets extensively used in the field of computer vision for image classification tasks. CIFAR-10 consists of 60,000 32x32

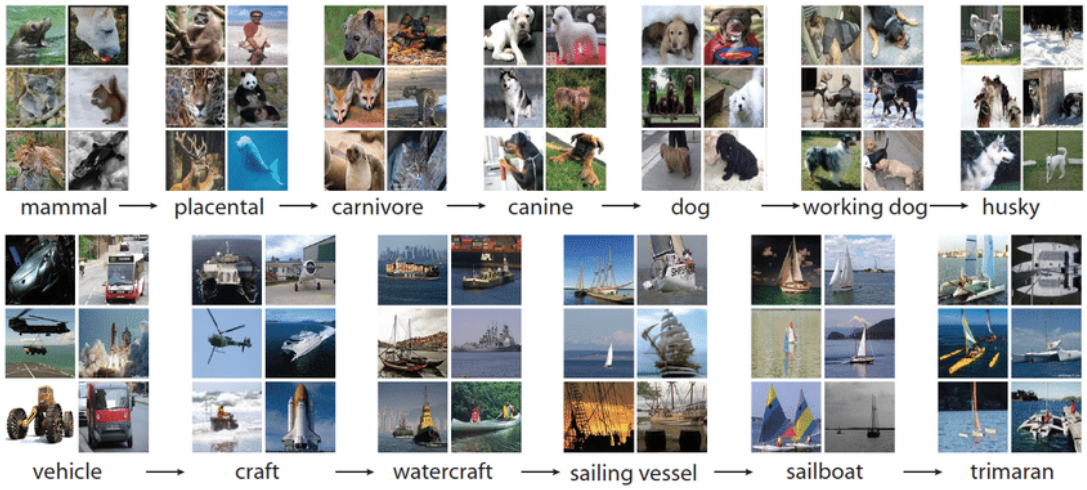


Figure 5.3: Samples from Imagenet-1k (J. Deng et al., 2009)

color images across 10 classes, with 6,000 images per class. The classes cover a wide range of everyday objects such as cars, airplanes, and animals. CIFAR-100, on the other hand, contains 100 classes, each containing 600 images, which are grouped into 20 superclasses, achieving a higher level of complexity by introducing subclasses within broader categories.

(f) **ImageNet-1K** (J. Deng et al., 2009), also known as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) dataset, contains approximately 1.2 million images spanning 1,000 categories, containing 10,000 manually selected synsets. (g) ImageNet-22K (Ridnik et al., 2021) expands on this by including around 14 million images across 21,841 synsets. The images in the ImageNet datasets vary in resolution, as they are sourced from various websites and cameras. However, the dataset's standard practice is to resize images to have a fixed resolution before use in experiments and evaluations. In most cases, the images are resized to a resolution of 224x224, 256x256 or 384x384 pixels, depending on the specific requirements of the task or model being trained. ImageNet has become a standard benchmark dataset in computer vision research, for image classification, therefore it has been instrumental in advancing the field of deep learning, serving as a crucial resource for training and evaluating state-of-the-art models.

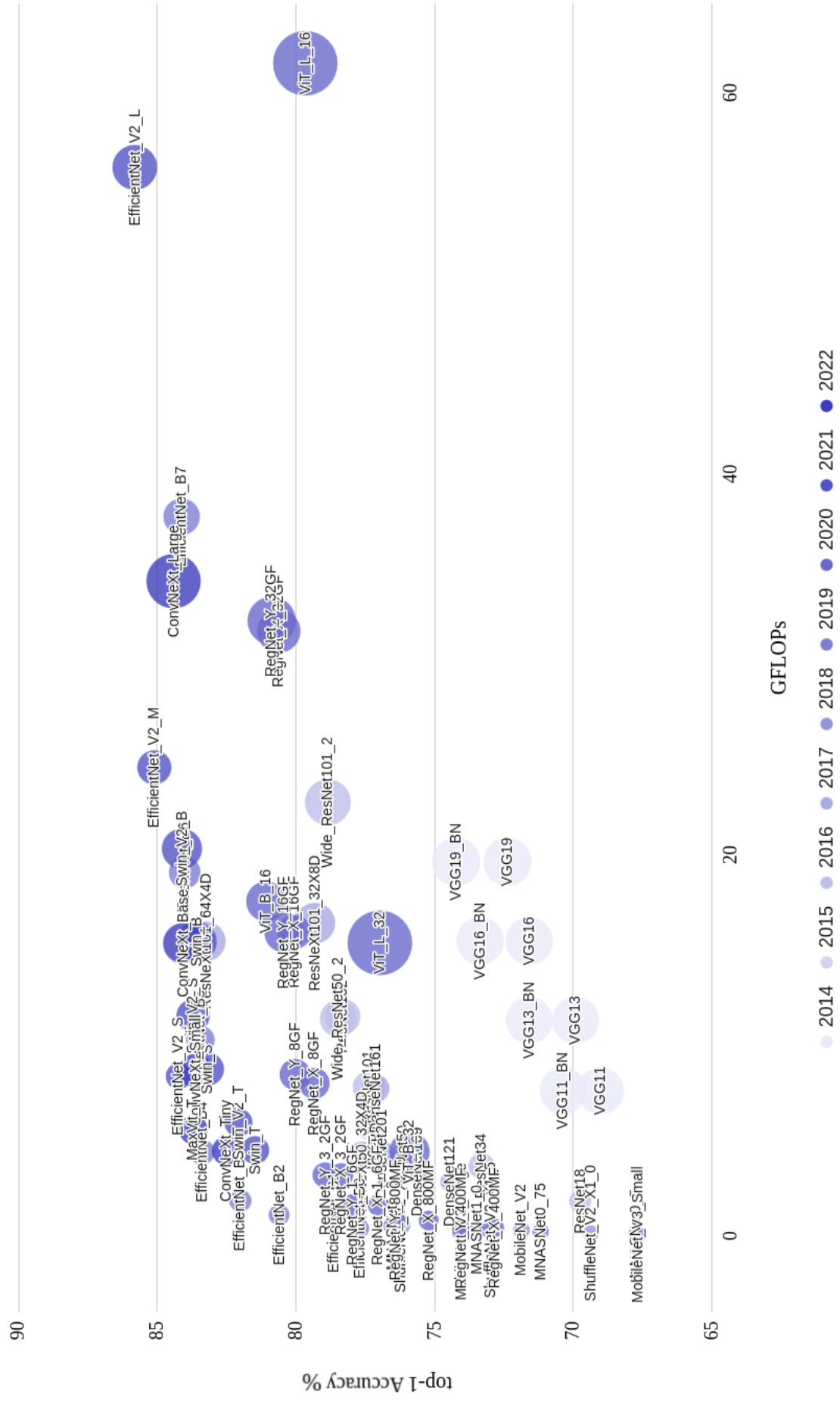


Figure 5.4: top1 accuracy % to GFLOPs visualization of state-of-the-art methods in Imagenet-1k (J. Deng et al., 2009) image classification dataset, from early (lighter colors) to recent (darker colors) deep learning methods. Size of bubble represents the number of parameters of each model.

5.1.3 Evaluation

Evaluation is crucial for assessing the performance of classification models accurately. The most prominent metric used is accuracy, which calculates the proportion of correctly classified images out of the total number of images in the dataset. To provide insights into the model's proficiency in making accurate predictions even when the correct class may not be the highest probability prediction, the top-k% accuracy metric can be defined, by considering the k-top probabilities predicted by the model. For the scope of this work, the top-1% accuracy will be used as a metric of performance in the image classification task.

Figure 5.4 depicts the evolution of state-of-the-art in image classification methods and provides insights into the advancements achieved in this field. Notably, both accuracy and the trade-off between accuracy and computational cost have been significantly pushed in recent years, indicating an improvement of effectiveness and computational efficiency of image classification models. Moreover, while there is a general reduction of model parameters compared to early deep learning models like VGG, the emergence of transformers and their impact on modern CNNs like ConvNext (Z. Liu et al., 2022) led to increased parameter count in recent methods. Finally, the large amount of research effort to enhancing model performance for visual tasks has resulted in a diverse range of models with varying attributes and characteristics, reflecting the multifaceted nature of the problem and the need for tailored solutions to address specific requirements and constraints.

5.2 Memory-aware Dynamic Performance

5.2.1 Introduction

In dynamic systems, the ability to adapt to varying computational costs is important, since computational resources can fluctuate in real-time due to factors like workload demands, power constraints, user interactions or environmental conditions. This adaptability is particularly pronounced in embedded and IoT systems due to their inherent limitations in power and con-

strained resources. Therefore, it is paramount for models to strike a balance between performance and computational cost, necessitating the exploration of trade-offs between model accuracy and the number of FLOPs required for inference. There is currently a plethora of CNN (He et al., 2016), (Tan & Le, 2019), (Tan & Le, 2021),(Z. Liu et al., 2022) and Transfomer (Dosovitskiy et al., 2020), (Z. Liu et al., 2021), (Ryali et al., 2023), (M. Chen et al., 2023) models, which are meticulously trained for a wide range of target FLOP budgets, delivering optimal performance under the available computational resources. Transitioning between the best available models of the accuracy to FLOPs pareto front to meet the available computational resources at any given time, is a promising solution to further enhance adaptability and ensure optimal performance.

However, another critical consideration for real dynamic systems that is often neglected, is the availability of memory to store model parameters. In any type of device, the available storage memory is finite and often limited, making this a universal constraint across all devices, especially in embedded or IoT devices. To mitigate that, proposed solutions involve techniques for executing parts or entire workload on the cloud (cloud offloading) (Kang et al., 2017), (E. Li et al., 2019), although this approach may encounter unpredictable latency and raise data privacy concerns due to reliance on network connections. Moreover, loading and unloading model weights to accommodate changing computational demands (model-swapping) introduces notable delays and could potentially disrupt the inference pipeline, calling for hardware specific solutions (M. Yu et al., 2023), (K. Wang et al., 2024). Therefore, establishing a balance between model accuracy, computational cost and number of parameters allows the deployment of models across different hardware platforms, especially in resource-constrained environments.

Model compression techniques provide solutions for improving the efficiency of deep learning models and provide ways to meet the varying computational resources and enable inference at multiple FLOP regimes. The following section presents a high-level view of the techniques and examine them for this problem, from the performance to memory trade-off perspective. For consistency, let us denote that each model x may include either one (x_0) or multiple model backbones $x_i \in N$, and $P(x_i)$ is the number of parameters required to be stored for each x_i.

A simple yet effective approach to achieve dynamic performance can be established by altering the input image resolution of images fed into the model. **Resolution scaling** is defined by a scaling factor which alters the dimensions of the input image, affecting the computational requirements of the model. This method is achievable since CNNs and Transformers with positional encoding are scale invariant, meaning they can effectively process images at different resolutions without significantly impacting their ability to extract features and make predictions. It introduces zero parameter overhead requiring only the weights of a single model, since inference is performed with that model at all scales.

Quantization (Banner et al., 2019) (L. Deng et al., 2020) is a technique used to reduce the precision of numerical values in neural network models. By converting parameters and activations from higher precision formats, such as 32-bit floating point numbers, to lower precision formats like 8-bit integers, quantization can significantly reduce the memory and computational requirements during inference, while the number of parameters remains the same. However, it introduces rounding errors and reduces the representational capacity of the model, which can eventually lead to a degradation in performance. To mitigate that, techniques such as quantization-aware training (QAT) or Post-training quantization (PTQ) can be employed. In the former the model is trained with quantization in mind, allowing it to adapt to the lower precision format while minimizing loss of accuracy, while in the latter the quantization process is applied to the trained model after it has completed training For now, we consider quantization from a memory perspective. To achieve dynamic performance using quantization, given a full precision model, multiple quantized models need to be generated with varying performance and memory costs, which are generally smaller than the full precision model, since weights at lower precision format require less space to store. Therefore, the total memory cost to store all the models to achieve dynamic performance of a full precision model using quantization is $\sum_{i=1}^{N} P(Q(x_i))$, where $Q()$ is the quantization function of model to achieve computational cost.

Pruning (S. Yu et al., 2022), (Molchanov et al., 2019), (Z. Liu et al., 2019), (Huang & Wang, 2018), (S. Lin et al., 2019), (J. Liu et al., 2024) is another technique commonly used for model optimization, which involves removing certain connections, nodes, or parameters from the network to reduce its size and computational cost while attempting to maintain performance.

Classical pruning techniques involve identifying and removing connections or parameters based on certain fixed criteria, such as their magnitude (magnitude pruning), importance scores (importance pruning), or their contribution to the overall model error (error-based pruning). Using these techniques architecture and parameters of the network are directly modified (zeroed out or removed), requiring no extra parameters beyond the original model. However, in some more modern techniques like structured pruning or sparsity-inducing regularization additional information needs to be stored. This typically involves a pruning mask which indicates which connections or parameters are pruned (set to zero) and which ones are retained. However, this mask is usually binary, where a value of 1 indicates that the corresponding weight is retained, and a value of 0 indicates that the weight is pruned, therefore incurs a smaller memory cost for its parameters. To achieve dynamic inference in N computation budget regimes, N pruning masks need to be stored in addition to the original model, leading to $P(x_0) + \sum_{i=1}^{N} P_i^m(x_0)$ total number of parameters, where $P_i^m(x_0)$ are parameters of the pruning mask.

Low-rank approximation (X. Zhang et al., 2015), (L. Deng et al., 2020) methods are another set of techniques used for model optimization, for reducing the computational and memory requirements of deep learning models. The purpose of low-rank approximation is to approximate model weights with lower-dimensional representations, by decomposing the original weight matrices into a product of lower-rank matrices, which require fewer parameters to represent compared to the original weight matrices. For example, in SVD-based low-rank approximation, you decompose a weight matrix W into three matrices U, Σ, and V, where U and V are orthogonal matrices and Σ is a diagonal matrix containing the singular values. Then, by truncating U, Σ, and V to retain only the most significant singular values and their corresponding columns, results in a lower-rank approximation of W. The rank of approximation can be controlled by the number of singular values that are used to approximate the original weight matrix, introducing a controllable trade-off between performance and computational cost. Regarding required memory, the approximated weights can be stored once for inference at various FLOP regimes, constituting a memory efficient approach.

Dynamic methods encompass a spectrum of techniques that adaptively adjust model parameters and architectures during the inference process, tailoring predictions to the specific

Technique	Total parameters to store for N FLOP regimes
Individual Models	$\sum_{i=1}^{N} P(x_i)$
Resolution Scaling	$P(x_0)$
Quantization	$\sum_{i=1}^{N} P(Q_i(x_0))$
Pruning	from $P(x_0)$ to $P(x_0) + \sum_{i=1}^{N} P_i^m(x_0)$
Low-rank Approximation	$P(LR(x_0)))$
Spatially Adaptive	$\sum_{i=1}^{N} P(x_i) + P^m(x_i)$

Table 5.1: Table summarizing the memory cost to store weights for operation at N FLOP regimes, when using each technique to achieve dynamic performance.

characteristics of the input data, allowing an increased level of flexibility and efficiency in model predictions. **Spatially adaptive** methods, presented at Chapter 2 as one subset of dynamic methods, dynamically alter the processing applied to various image regions, conditioned on the input, showcasing favorable performance to cost trade-offs in vision tasks (Verelst & Tuytelaars, 2020), (Rao et al., 2023), (Zampokas et al., 2024). Starting from a base model, they generate dynamic variants for each specific computation target resulting in a distinct set of model weights for each target budget. Therefore, to achieve inference under N FLOP regimes, the model weights along with those of the importance estimation module need to be stored. The memory cost of achieving inference at multiple computational costs, matches this of the multiple model technique, essentially requiring a full model and the importance estimator weights to be stored, leading to $\sum_{i=1}^{N} P(x_i) + P^m(x_i)$ total number of parameters, where $P(x_i)$ and $P^m(x_i)$ are parameters of the model and importance estimation module, trained for a specific computational budget regime i.

In this section, various techniques which can be used to achieve dynamic inference at multiple FLOP regimes have been explored and examined from a memory perspective, particularly regarding the storage requirements of model parameters. The summary of these techniques, along with their respective number of parameters, can be found in Table 5.1, offering an overview of the presented methods. Notably, this analysis represents a high-level exploration, intended to offer a glimpse into the potential bounds of achieving inference at multiple computational budgets using the discussed strategies. Furthermore, it is worth highlighting that many of these methods are complementary and can be used in conjunction with each other to achieve even more favorable cost and memory-to-performance trade-offs. While delving into the details of

such combined approaches presents an interesting investigation for future studies, it lies beyond the scope of this analysis.

In this chapter, we focus on exploring a spatially adaptive methodology to achieve inference at multiple target computational budgets, while also considering the number of parameters. Such methods have achieved state-of-the-art performance to computational cost trade-offs as presented in previous chapters. Yet, the parameter count overhead to operate at multiple target computational budgets grows rapidly for each supported budget, requiring budget specific backbone and importance estimation parameters. However, we observe that spatially adaptive models tailored for different budgets inherently share the same foundational architecture suggesting a potential degree of redundancy across different computational budgets. This observation hints at the potential to exploit such redundancy to facilitate dynamic inference in scenarios where memory resources are constrained, offering a promising avenue for efficiency optimization.

5.2.2 Limitations

Achieving inference at multiple computation budgets can be achieved by employing a different model that meets the each budget's constraint. Compression methods offer a viable solution to this by enabling the creation of compressed versions of the base model, to accommodate the diverse computational budgets. However, maintaining high accuracy across these budgets necessitates dedicating a considerable number of parameters to each, which can be impractical. Nonetheless, a common drawback persists in many existing methods: the resultant models for each budget remain independent entities. This independence means that despite compression, each budget still requires its own distinct model, potentially limiting the efficiency gains.

In conclusion, spatially adaptive methods produce dynamic variations of models aimed at achieving specific optimization targets (FLOPs or latency), yielding unique sets of model weights, optimized to perform at each target. Therefore, addressing the challenge of accurate inference at multiple computational budgets using existing spatially adaptive methods entails an significant escalation in the number of parameters, which can pose problems, particularly

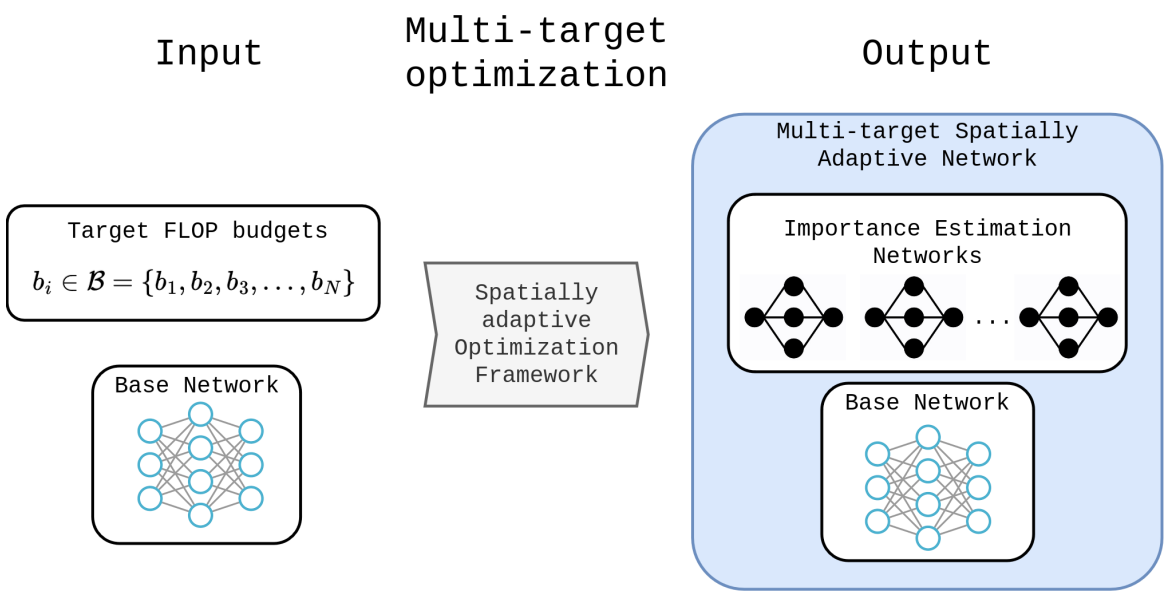


Figure 5.5: Graph of the multi-target spatially adaptive optimization framework. It receives a network and target FLOP budgets and generates a spatially adaptive variant able to operate at the target FLOP budgets with minimal increase in number of parameters.

in resource-constrained scenarios. This necessitates adopting a parameter-efficient approach capable of achieving accurate inference across various computational budgets.

5.2.3 Contributions

This work addresses the problem of performing inference at multiple computation budgets while maintaining a small number of parameters. Towards that goal, dynamic spatially adaptive methods are explored as a solution to produce dynamic variants of a base model to operate at multiple computation budgets. We propose a multi-target training method which produces spatially adaptive variants of a base model, capable of achieving inference under multiple target computational budgets (Figure 5.5). This approach attempts to extend the success of spatially adaptive methods in computational efficiency in the memory domain. They key concept is a weight-sharing architecture which facilitates the desired behavior with negligible increase in parameter count. Moreover, the training process offers controllable granularity in the number of the target computational cost targets, by introducing a trade-off between performance and memory.

Model	Parameter count (M) (% of backbone)		
	Network	Mask Units	Normalization
ResNet-20	0.272	0.003 (1.10%)	0.001 (0.49%)
ResNet-32	0.467	0.005 (1.10%)	0.002 (0.48%)
ResNet-18	11.690	0.004 (0.04%)	0.014 (0.13%)
ResNet-50	25.600	0.088 (0.34%)	0.073 (0.29%)
ResNet-101	44.549	0.265 (0.59%)	0.124 (0.28%)
ConvNeXt-T	28.589	0.066 (0.23%)	0.013 (0.05%)
ConvNeXt-S	50.224	0.135 (0.27%)	0.027 (0.05%)
ConvNeXt-B	88.591	0.181 (0.20%)	0.036 (0.04%)

Table 5.2: The number of parameters needed to store the weights of backbone, mask estimation, and normalization layers, alongside their respective percentages relative to the backbone.

5.3 Motivation

5.3.1 Background Motivation

Devices ranging from smartphones to servers experience varying levels of computational capacity during operation, resulting in available computational resources which can change dynamically during runtime, due to factors like battery life (Ali et al., 2015), network congestion (Feng et al., 2022), user interactions (Bantock et al., 2017), concurrent application demands, or environmental conditions. This variability poses a fundamental problem: how can machine learning models adapt to such dynamic environments to ensure efficient resource utilization while maintaining performance? The need for models capable of dynamically adjusting their computational budgets to match the available resources is evident. In this work we consider the problem of dynamic inference for varying computational budgets, while minimizing memory requirements. Spatially adaptive methods have demonstrated already favorable trade-offs of performance to computational cost by exposing the spatial redundancy of computations, however their deployment and suitability in multi-budget inference remains vastly unexplored. These models typically consist of a backbone and importance estimation layers, which are both optimized for a specific target computational cost budget. Therefore, given the similarity of architectures for any target budget, their parameter redundancy under our problem setting is promising to area to explore.

5.3.2 Hands-on Motivation

To explore how the parameter count of existing spatially adaptive methods scales for inference at multiple computational budgets, this analysis focuses on DynConv (Verelst & Tuytelaars, 2020) method, which belongs in the state-of-the-art pareto front of spatially adaptive methods for model compression. We conduct an analysis of the number of parameter within spatially adaptive architectures of widely used image classification models, focusing particularly on ResNets (He et al., 2016) and ConvNeXts (Z. Liu et al., 2022). To achieve accurate inference at N FLOP targets using DynConv, equal number of backbone and mask estimation layers need to be trained and stored. Investigating this overhead in the parameter count of Table 5.2, it is evident that mask estimation modules account for much fewer parameters than the backbone, and fewer than 0.5% in most cases. This provides motivation to consider using a single "expensive" memory-wise backbone and multiple "cheaper" mask estimation modules. Moreover, it prompts further investigation into the extent to which shared information can be encoded within a single backbone, thus motivating the exploration of a model design capable of targeting multiple computational budgets by sharing a common backbone to minimize memory overhead.

5.4 Methodology

5.4.1 Problem Formulation

The problem of dynamic model performance is considered from a multifaceted approach, by simultaneously considering trade-offs of performance to computational cost and number of parameters. This ensures that the resulting models are not only efficient in terms of computational cost but also compact, making them easier to deploy in resource limited environments. To formally define the problem let's consider x as a base CNN model and functions $C(x)$, $P(x)$ and $A(x)$ which measure floating-point-operations, number of parameters and performance of x, by using a suitable metric depending on the application. Within the context of this work, a bud-

get constraint is used to fix the computational cost, matching the ability of a dynamic model to adapt to the availability of computational resources in a real system. This is achieved by defining a set of N FLOP regimes as ratios to the FLOPs of x as $b_i \in \mathcal{B} = \{b_1, b_2, b_3, \ldots, b_N\}$. For each b_i, a corresponding model x_i is defined, resulting in the following problem of finding the optimal x_i models which:

$$\begin{aligned} Maximize \quad & \sum A(x_i) \\ Minimize \quad & \sum P(x_i) \\ b_i - \epsilon \leq C(x_i) \leq b_i + \epsilon, \quad & i \in \{1, 2, ..., N\} \end{aligned} \tag{5.1}$$

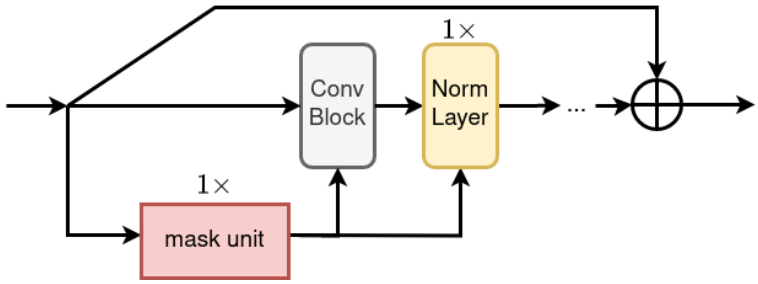
where ϵ represents a tolerance for computational budget, typically set to 3%. In summary, the problem of dynamic inference for multiple computational budgets is formulated as a multi-task problem with each task representing inference at a specific computational budget.

5.4.2 Multi-Budget Training with Sparse Optimization Framework

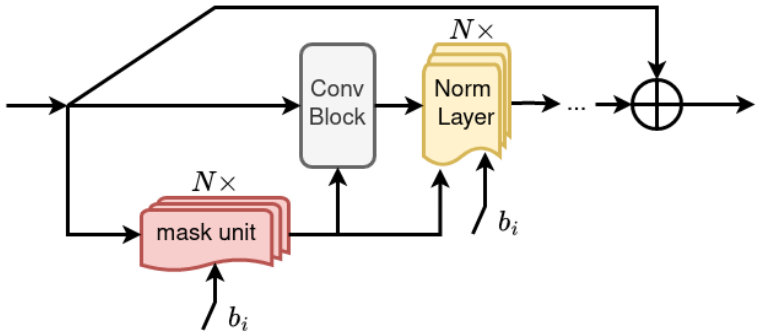
Architecture

The proposed multi-target inference framework is based on DynConv (Verelst & Tuytelaars, 2020), which converts a base CNN to a spatially adaptive variant, providing the ability to skip computations at certain spatial locations to achieve improved performance to FLOP ratio. To enable this behavior, (Verelst & Tuytelaars, 2020) introduce a mask unit responsible for learning which spatial positions should be evaluated. Given a target computation budget, the sparse network is trained to minimize the distance between that and FLOPs required for sparse execution. This results in a set of base model and mask estimation weights for each target computational budget.

Our proposed approach results in a spatially adaptive model that can operate at N computational budgets by a minimal increase in base network parameters. We achieve this by mapping each mask unit to one of N distinct mask units, each corresponding to a particular target budget. Similarly, the normalization layers within each residual block are also mapped



(a) Spatially adaptive residual layer of Verelst and Tuytelaars (2020)



(b) Proposed multi-target spatially adaptive residual layer

Figure 5.6: Spatially adaptive residual layers capable for inference at a single budget (a) or supporting N multiple budgets b_i (b). Referencing b_i, a budget-specific mask unit and normalization layer is selected.

to N instances, each also referring to a different target computation budget. While, having target-specific normalization instances is not a requirement to achieve multi-target inference, in our method, computations are skipped in a varying rate between different execution budgets. Consequently, scale and data distributions varies between them, underscoring the significance of per-target normalization. The parameter overhead of achieving the multi-target behavior ranges from 0.20% to 1% of backbone number of parameters for each additional computational budget. At execution the target computational budget b_i is also provided to the model as input, as:

$$Y_i = x(image, b_i), \tag{5.2}$$

indicating which set of mask and normalization units will be activated. Figure 5.6 demonstrates the process of selecting the corresponding mask and normalization units for the provided b_i, compared to the single mask and normalization unit approach of Verelst and Tuytelaars (2020).

Training

During the training process the goal is to find the backbone and mask unit parameters to achieve the best performance to computational cost trade-offs for all the target computational budgets in $\mathcal{B}$. To achieve that, the training process operates in an iterative manner, cycling through each target budget in a round-robin fashion.

We adopt the loss function from DynConv (Verelst & Tuytelaars, 2020). Given a network with R total residual blocks, and $C(r)$, $C_{sp}(r)$ the corresponding functions that return the number of FLOPs required for the static and spatially adaptive execution of residual block r, a loss that minimizes the difference between the target budget b_i and fraction of total FLOPs required for spatially adaptive to static execution is caclulated:

$$J^{b_i}_{sp,total} = \left(\frac{\sum_r^R C_{sp}(r)}{\sum_r^R C(r)} - b_i \right)^2 \tag{5.3}$$

To address the challenge of suboptimal initialization that occurs in spatially adaptive method training, an additional constraint is introduced, to maintain a balanced distribution of computations over block positions and batch elements. This constraint keeps the ratio of executed computations between bounds, which are tighter in the start of training and are gradually relaxed, by cosine annealing of parameter p from 1 to 0 in the following loss functions:

$$J^{b_i}_{sp,low} = \frac{1}{R} \sum_r^R max(0, p \cdot b_i - \frac{C_{sp}(r)}{C(r)})^2 \tag{5.4}$$

$$J^{b_i}_{sp,high} = \frac{1}{R} \sum_r^R max(0, \frac{C_{sp}(r)}{C(r)} - (1 - p(1 - b_i)))^2 \tag{5.5}$$

To mitigate the potential performance drop by introducing sparsity (Han et al., 2022), (Han et al., 2023) (Rao et al., 2021), (Rao et al., 2023), we use knowledge distillation to guide the spatially adaptive model (student) with the base model (teacher). For this reason, the KL divergence is minimized between the output probabilities of the spatially adaptive model P^{b_i}

for each target budget b_i and the teacher P as:

$$J_{KD}^{b_i} = D_{KL}(P||P^{b_i}) = \sum_k P(k) log \frac{P(k)}{P^{b_i}(k)} \tag{5.6}$$

Where D_{KL} denotes the Kullback-Leibler divergence, and k iterates over each class or output dimension.

By seeking backbone weights to serve inference at multiple targets, we found that it results in an "average" backbone model, which tends to exhibit a bias towards central targets within $\mathcal{B}$. Consequently, the training process may inadvertently prioritize these central targets, underperforming at more extreme sparsity targets. Our proposed solution is to guide the training process by using the potential performance that the spatially adaptive model can achieve. This is achieved by weighting the loss at each target budget with the fraction of the squared distance of the current and the potential performance to the total of these difference across all budgets in $\mathcal{B}$ as:

$$w^{b_i} = \frac{(A_m^{(b_i)} - A_s^{(b_i)})^2}{\sum_b^{\mathcal{B}} (A_m^{(b)} - A_s^{(b)})^2}, \quad b_i \in \mathcal{B} \tag{5.7}$$

where $A_m^{(b_i)}$ and $A_s^{(b_i)}$ represent the top-1% accuracies of the proposed multi-target model and the guide model correspondingly. Accuracy of models generated to operate at a single target budget using the models trained with DynConv (Verelst & Tuytelaars, 2020) method as guide.

The final loss for each b_i is composed by putting the individual components together and has the form of:

$$J = \sum_{b_i}^{\mathcal{B}} w^{b_i} (J_{task}^{b_i} + J_{sp,total}^{b_i} + J_{sp,low}^{b_i} + J_{sp,high}^{b_i} + J_{KD}^{b_i}) \tag{5.8}$$

with J_{task} denoting the task relevant loss function or Cross Entropy loss for image classification.

5.5 Experimental Results

The proposed multi-target spatially adaptive optimization framework is evaluated across a range of models and computer vision tasks. The evaluation follows a consistent setting; re-

porting both trade-off between task performance (top-1 accuracy % for classification and Mean Intersection over Union or mIOU% for semantic segmentation) and FLOPs, along with the number of parameters required to perform inference at the multiple FLOP budgets.

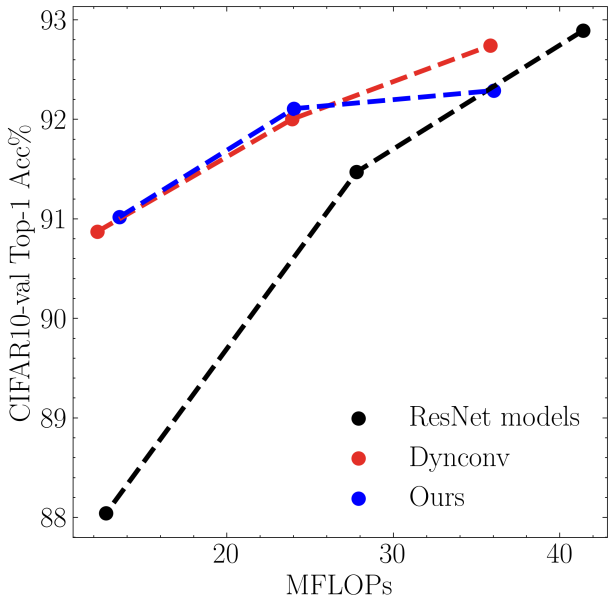
5.5.1 Image Classification

The proposed multi-target spatially adaptive optimization framework is applied on a set of architectures and two datasets for image classification; ResNet (He et al., 2016) for CIFAR10 (Krizhevsky, 2012) and both ResNet and ConvNeXt (Z. Liu et al., 2022) for Imagenet-1k (J. Deng et al., 2009). The goal is to evaluate the methodology on both classical models with more available baselines for comparison (ResNet (He et al., 2016)) and also explore the trade-offs on state-of-the-art CNNs (ConvNeXt (Z. Liu et al., 2022)) for vision tasks.

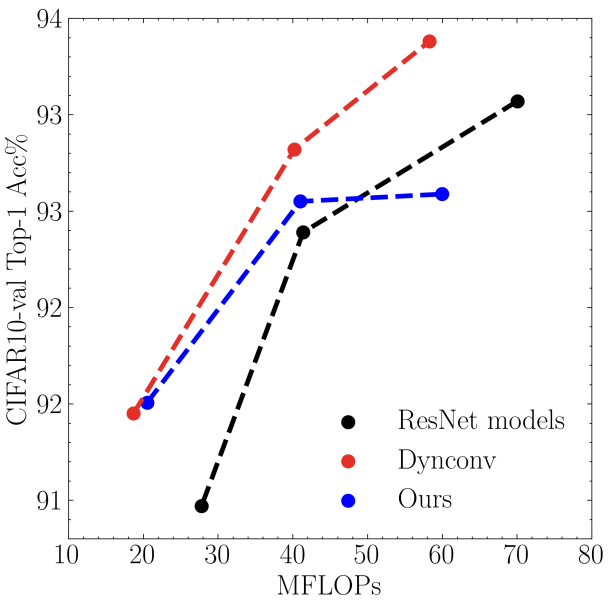
Setup

CIFAR-10. Residual models with 9 (ResNet-20) and 15 (ResNet-32) residual blocks are trained from scratch for image classification on the CIFAR10 (Krizhevsky, 2012) dataset. Training follows the schedule of DynConv (Verelst & Tuytelaars, 2020) but optimizes for three target budgets $\mathcal{B} = \{20\%, 50\%, 70\%\}$. To alleviate noise in results and extract statistically significant conclusions, also given the relatively smaller size of CIFAR10 dataset, results are obtained by performing training 10 times and keeping mean scores for all metrics.

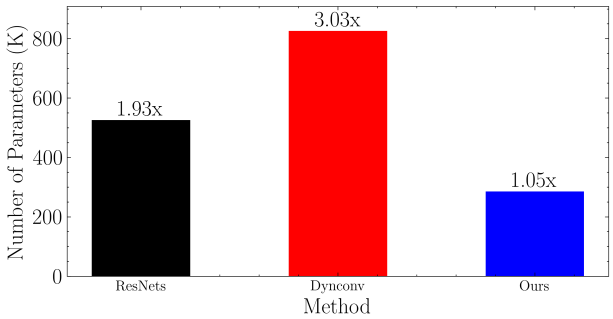
ImageNet-1k. As for the ImagetNet-1k dataset, the schedule of Verelst and Tuytelaars (2020) is followed for training spatially adaptive ResNet-18, ResNet-50 and ResNet-101 models, to allow a fair comparison. Model weights are initialized by the pretrained base models and the target budgets for our multi-target framework are set to $\mathcal{B} = \{30\%, 50\%, 70\%\}$. Three variants of ConvNeXt models (ConvNeXt-T, ConvNeXt-S and ConvNeXt-B) are also trained with our framework for target budgets of $\mathcal{B} = \{30\%, 50\%, 70\%\, 90\%\}$. We adopt the training schedule of the relevant spatially adaptive approach of Rao et al. (2023), which is based on the original paper of ConvNeXt (Z. Liu et al., 2022), but performs training for 120 epochs instead of 300, starting from pretrained checkpoints.



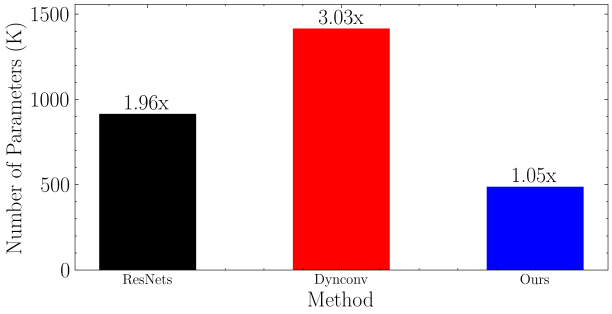
(a) ResNet-20 Performance



(b) ResNet-32 Performance



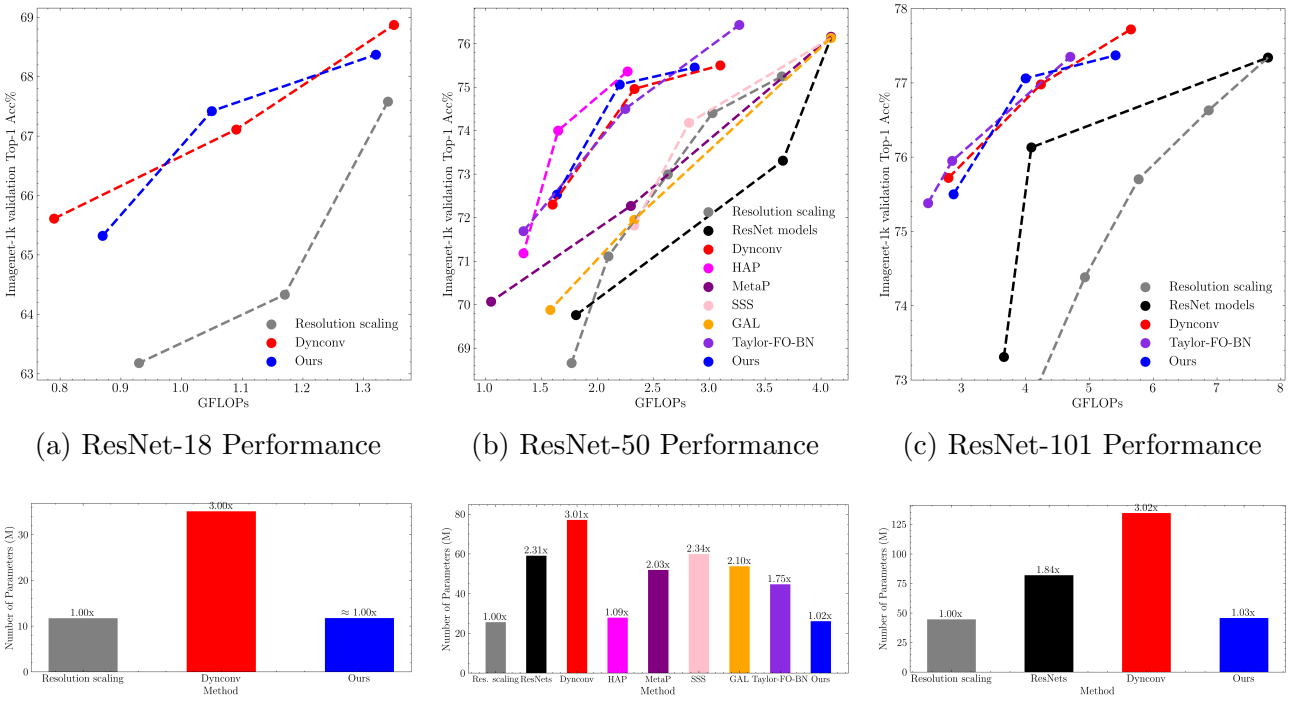
(c) ResNet-20 Parameters



(d) ResNet-32 Parameters

Figure 5.7: Results of multi-target inference with ResNet models on CIFAR10 dataset.

Comparisons. We compare our method with a set of relevant spatially adaptive optimization and compression methods. For each target budget, a spatially adaptive model for a single FLOP budget with (Verelst & Tuytelaars, 2020) is trained. The top-1 accuracy % scores of these models are also used to calculate the target awareness weights w^{b_i} for each target budget of our framework. Moreover, results from spatially adaptive ConvNeXt models using (Rao et al., 2023) are reported for models trained for a specific keeping ratios ρ (0.7 and 0.9). Those models also possess the ability to perform inference at multiple budgets by controlling the ρ value. Reported results from pruning methods for model compression are included (mostly targeting ResNet-50) for HAP (S. Yu et al., 2022), MetaP (Z. Liu et al., 2019), SSS (Huang & Wang, 2018), GAL (S. Lin et al., 2019), Taylor-FO-BN (Molchanov et al., 2019) and UPDP (J. Liu et al., 2024). Finally, in our ImageNet-1k experiments, we analyze inference across



(a) ResNet-18 Performance (b) ResNet-50 Performance (c) ResNet-101 Performance

(d) ResNet-18 Parameters (e) ResNet-50 Parameters (f) ResNet-101 Parameters

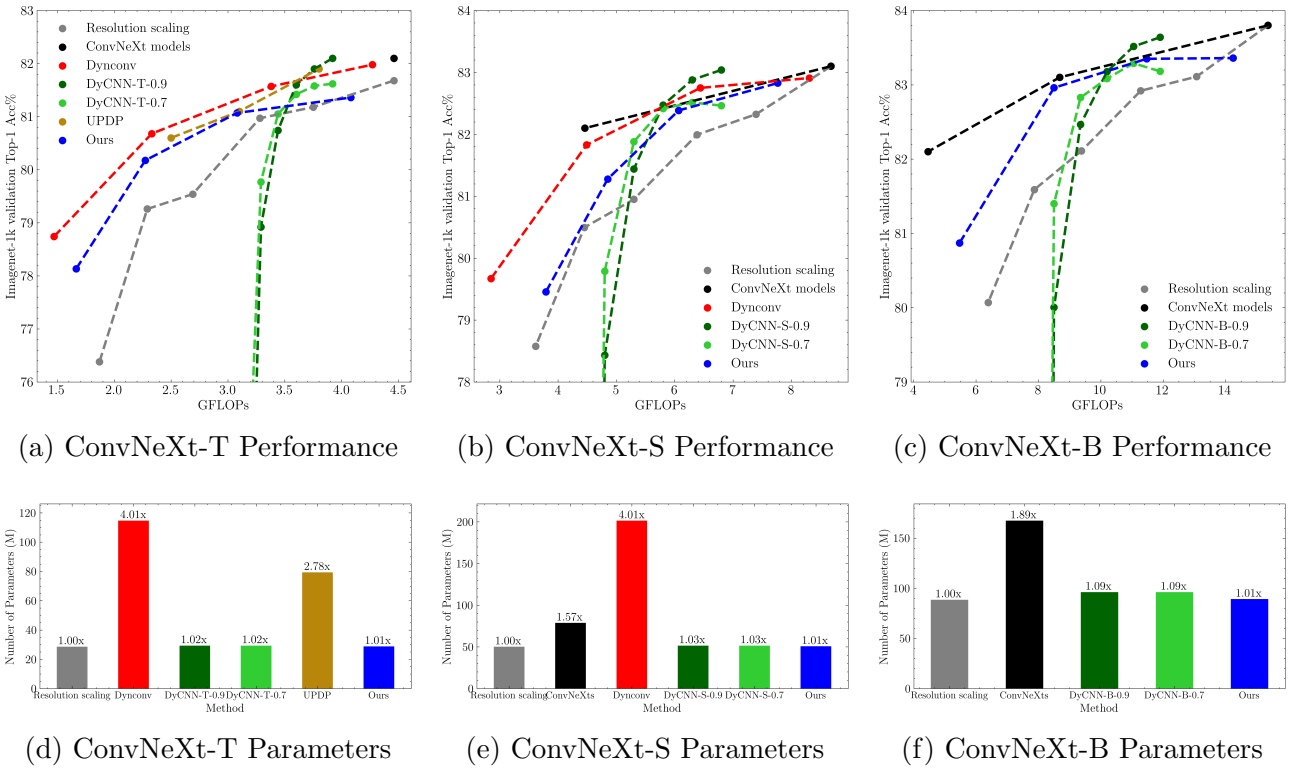
Figure 5.8: Results of multi-target inference with ResNet models on ImageNet-1k dataset.

different computational budgets by adjusting input image resolution. This includes evaluating base models with varying input sizes, reducing by increments of 16 pixels from the original 224×224 resolution.

Results

In CIFAR10 dataset (Figure 5.7), our multi-target method outperforms the multiple ResNet models at most cases, in both performance to cost and memory trade-offs. When compared to DynConv (Verelst & Tuytelaars, 2020), we observe that our method achieves comparable performance, particularly at lower budgets, while a growing discrepancy is observer at higher budgets, especially in ResNet-32. However, the memory cost of parameters of the proposed method is significantly lower, with a mere 5% increase to base model parameter count.

Results of our method compared against spatially adaptive optimization, pruning and scaling methods for ResNet models are presented in Figure 5.8. We observe that our multi-target method significantly outperforms the resolution scaling and multiple ResNet models, using roughly equal (resolution scaling) or less (ResNet models) number of parameters. Moreover, it



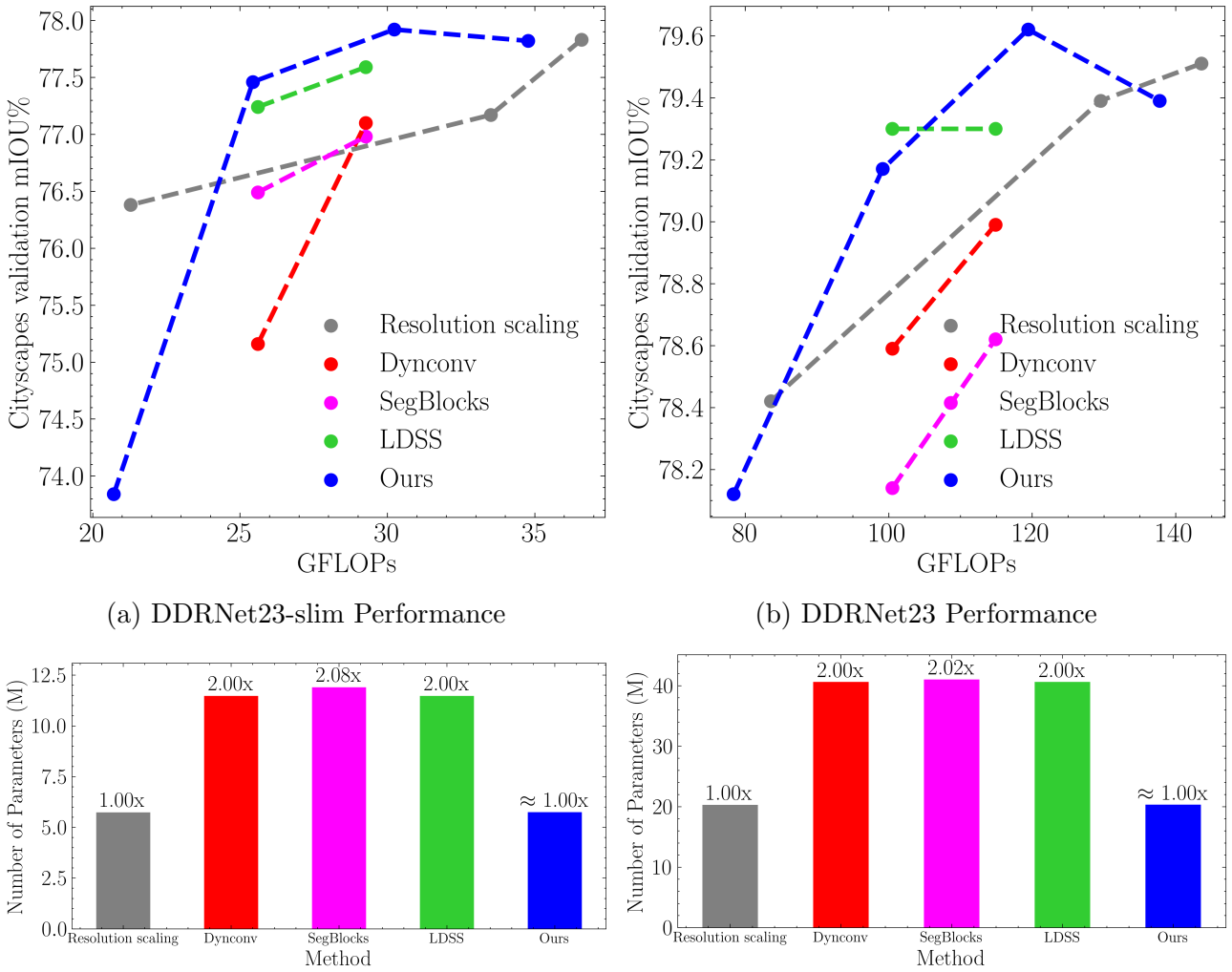
(a) ConvNeXt-T Performance (b) ConvNeXt-S Performance (c) ConvNeXt-B Performance

(d) ConvNeXt-T Parameters (e) ConvNeXt-S Parameters (f) ConvNeXt-B Parameters

Figure 5.9: Results of multi-target inference with ConvNeXt models on ImageNet-1k dataset.

achieves on-par results with DynConv (Verelst & Tuytelaars, 2020) method, while requiring a fraction of parameters. Our spatially adaptive method on ResNet-50 (Figure 5.8-b) is also well-placed compared to state-of-the-art pruning methods of HAP (S. Yu et al., 2022) and Taylor-FO-BN (Molchanov et al., 2019), while still requiring fewer parameters to achieve inference at multiple budgets (Figure 5.8-e).

Figure 5.9 shows that our method outperforms resolution scaling with just 1% of parameter overhead for ConvNeXt models. Scaling with ConvNeXt models and recent pruning method of UPDP (J. Liu et al., 2024) for ConvNeXt-T, demonstrate improved accuracy over our method, however requiring around double the number of parameters. Moreover, DynConv (Verelst & Tuytelaars, 2020) method consistently outperforms our method, albeit with a 4x parameter cost. Scaling with the spatially adaptive DyCNN (Rao et al., 2023) method is efficient memory-wise since it introduces a constant overhead in parameters (2% to 9% of backbone weights) which supports a wide range of budgets. It also demonstrates the best performance but only for a small budget range, close to the budget that the model has been trained for. When moving away from that range, performance deteriorates significantly, while our proposed method maintains



(a) DDRNet23-slim Performance

(b) DDRNet23 Performance

(c) DDRNet23-slim Parameters

(d) DDRNet23 Parameters

Figure 5.10: Results of multi-target inference with semantic segmentation DDRNet23/23-slim models on Cityscapes dataset.

more competitive performance across a much wider range of FLOP budgets.

5.5.2 Semantic Segmentation

Setup

We apply our multi-target optimization framework on DDRNet (Hong et al., 2021) architectures for semantic segmentation on the Cityscapes (Cordts et al., 2016) dataset. We follow the schedule and sparse architecture presented in the previous chapter and Zampokas et al. (2024) to train spatially adaptive DDRNet-23slim and DDRNet-23 models for 4 target FLOP budgets

Method	Parameter N targets
Resolution Scaling	W_B
DynConv (Verelst & Tuytelaars, 2020)	$N * (W_B + W_M)$
DyCNN (Rao et al., 2023)	$W_B + W_M$
Ours	$W_B + N * W_M$

Table 5.3: Scalability of parameters for multi-target inference with N targets. W_B, W_M denote backbone and mask estimation module parameters for each method correspondingly.

$\mathcal{B} = \{30\%, 50\%, 70\%\, 90\%\}$. Results are compared with relevant spatially adaptive optimization methods of DynConv (Verelst & Tuytelaars, 2020), SegBlocks (Verelst & Tuytelaars, 2023) and LDSS (Zampokas et al., 2024).

Results

Problems with large input image resolutions typically incur favorable trade-offs under resolution scaling. This is demonstrated in Figure 5.10, where resolution scaling performs better in various computation budgets compared to DynConv (Verelst & Tuytelaars, 2020) and SegBlocks (Verelst & Tuytelaars, 2023) methods. However, our proposed method clearly outperforms resolution scaling and other spatially adaptive approaches in most cases, with a negligible overhead in required number of parameters ($< 1\%$).

5.5.3 Scalability Analysis

We conduct an analysis on how the parameter count scales as the number of supported budgets increases for spatially adaptive methods. Table 5.3 shows the formula for supporting inference at multiple target budgets for resolution scaling and the methods of DynConv (Verelst & Tuytelaars, 2020) and DyCNN (Rao et al., 2023). Resolution scaling is the most memory-efficient approach, devoid of any additional parameters beyond the base model. DyCNN entails a fixed overhead comprising of the base model alongside parameters for a single mask estimation module, which can be used to perform inference at multiple budgets. Meanwhile, DynConv requires the base model and mask estimation for each target, escalating the backbone overhead by a factor of N. Finally, our method omits the cost of "expensive" backbone and only utilizes

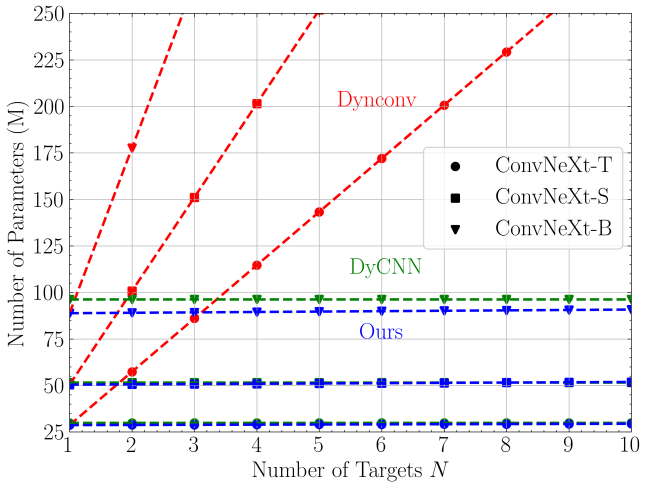


Figure 5.11: Scalability of parameters for ResNet and ConvNeXt models as a multiplier to base network parameters W_B.

	DynConv (Verelst & Tuytelaars, 2020)	DyCNN (Rao et al., 2023)	Ours
Sparse Decisions	N	3	N
Computation Scheme	Skip/Sparse	Skip/Sparse	Skip/Sparse
Mask Granularity	Pixel	Pixel	Pixel
Training Target	Single Budget	Single Pruning Ratio	Multiple Budgets
Budget Range for Pareto optimal FLOPs Efficiency	Very Narrow	Narrow	Wide

Table 5.4: Feature table comparing the key elements of the relevant spatially adaptive methods with our approach on CNNs for image classification at multiple FLOP budgets.

mask estimation modules for each target budget. Figure 5.11 illustrates the advantageous parameter count scalability of DyCNN and our approach in contrast to DynConv for ConvNeXt models across varying numbers of supported budgets. Notably, even though our method uses multiple mask estimation modules, it achieves comparable or better (ConvNeXt-B) parameter count scaling to DyCNN, which incurs a fixed parameter count overhead.

5.5.4 Discussion

The performance of DynConv (Verelst & Tuytelaars, 2020) is perceived as the ideal target for our method, since it optimizes a full network for each target budget, whereas our method shares one backbone to serve multiple target budgets. ResNet models lack the efficiency and compactness of more recent approaches, therefore encoding information of multiple backbones

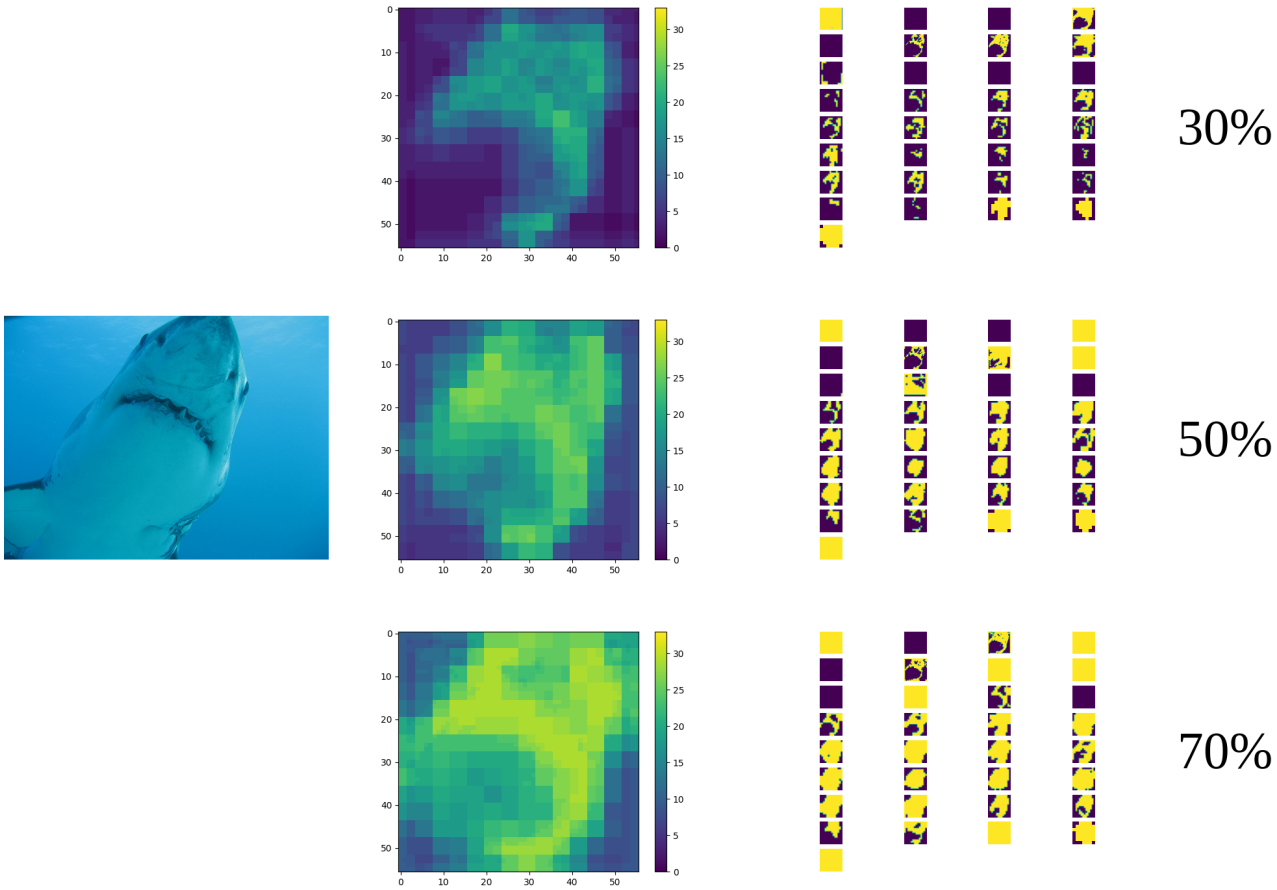


Figure 5.12: Aggregated (summed) layer masks (middle column) and individual masks (right column) for target budgets of 30%, 50% and 70% for ResNet101 model.

into one demonstrates resilience against accuracy degradation across a range of target budgets. On the other hand, modern ConvNeXts are inherently more efficient and optimized, presenting a greater challenge in compressing multiple backbones into one, resulting in performance drops from the ideal method. Overall, the small parameter count overhead associated with our approach ensures favorable scaling across any number of target budgets, positioning our method as one of the most efficient options for supporting multi-target computation objectives. Finally, Table 5.4 provides a comparative overview of the key elements of relevant spatially adaptive methods alongside our proposed approach, focusing on CNNs for image classification across multiple FLOP budgets.

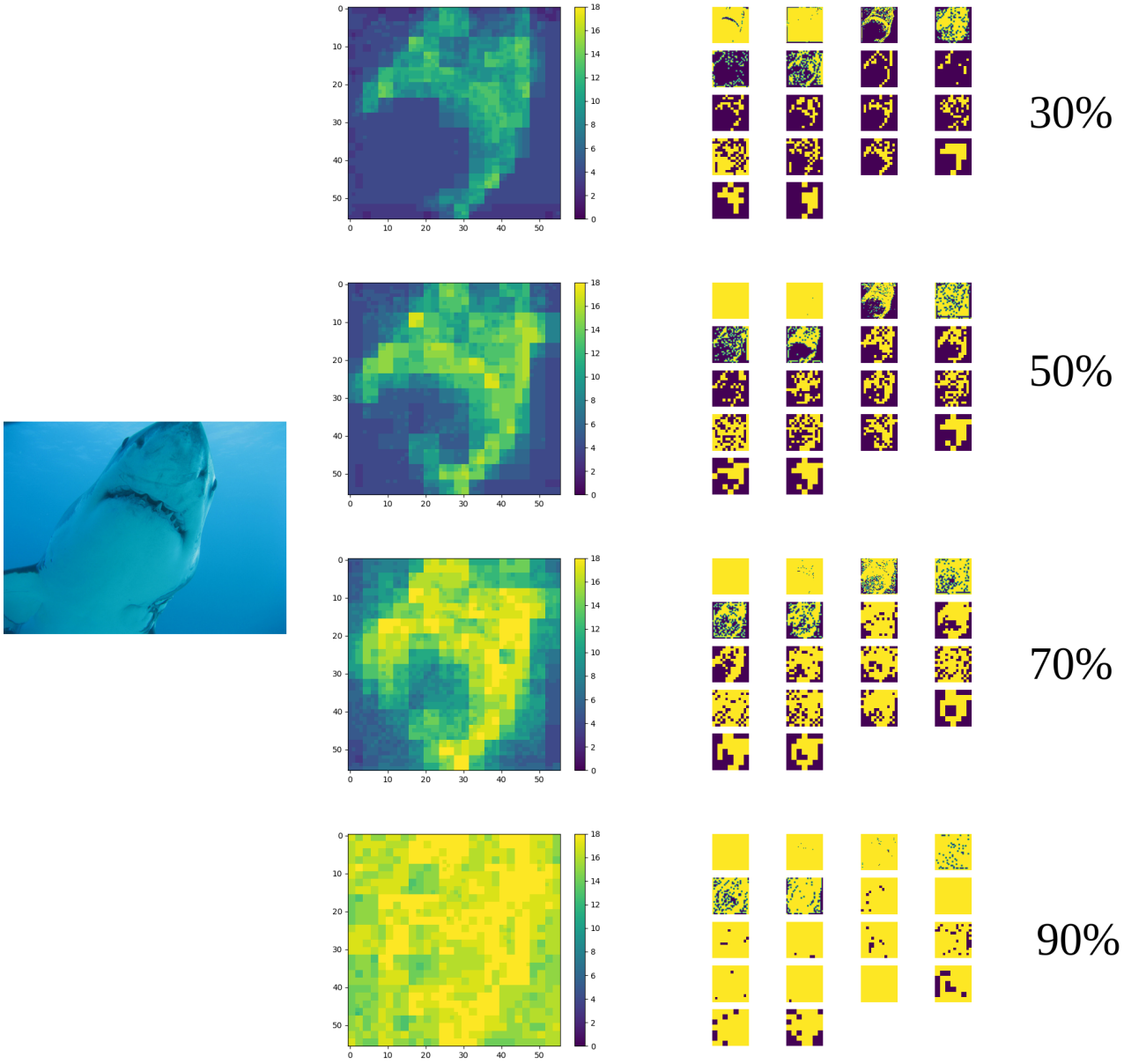


Figure 5.13: Aggregated (summed) layer masks (middle column) and individual masks (right column) for target budgets of 30%, 50%, 70% and 90% for ConvNeXt-T model.

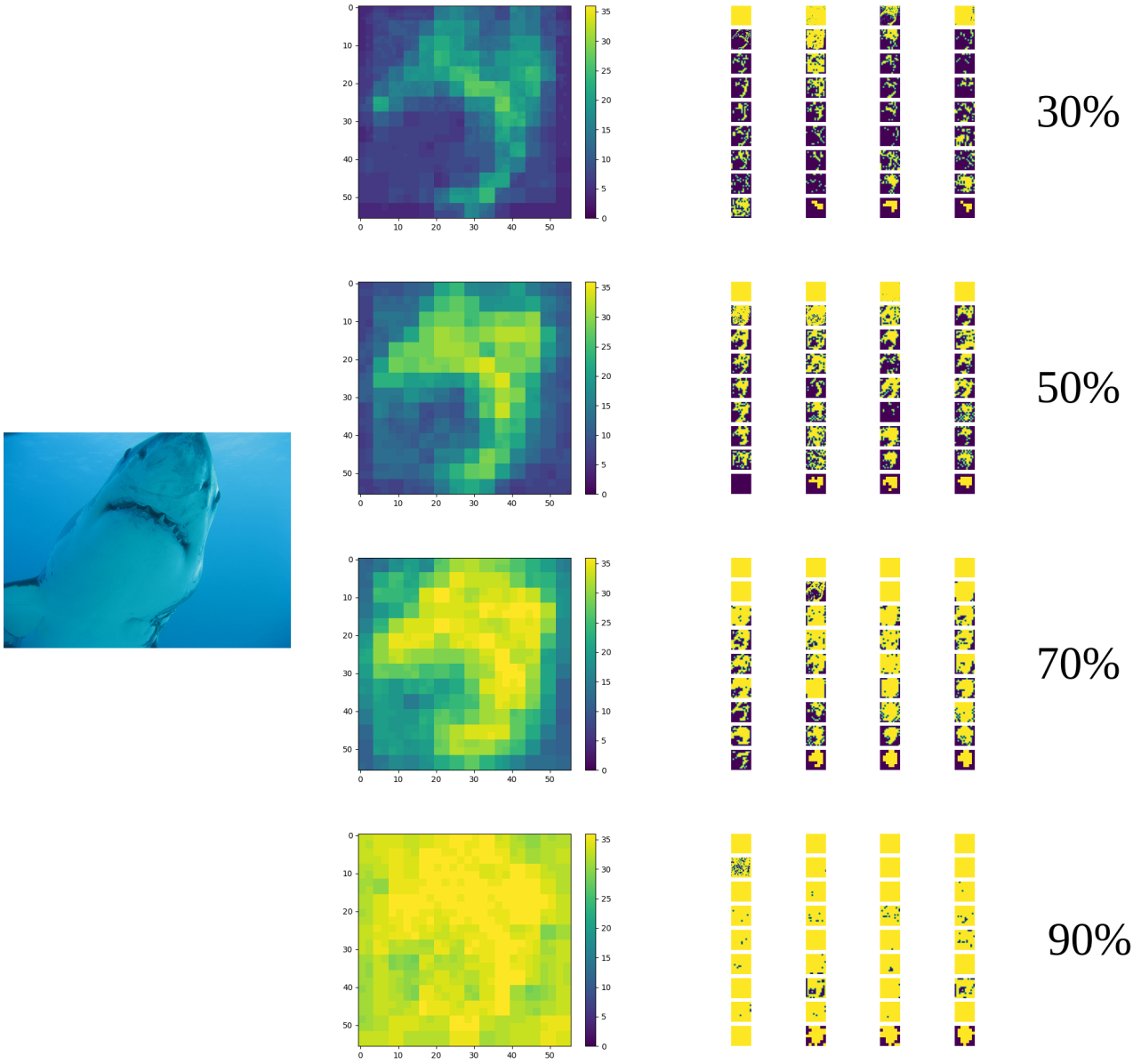


Figure 5.14: Aggregated (summed) layer masks (middle column) and individual masks (right column) for target budgets of 30%, 50%, 70% and 90% for ConvNeXt-B model.

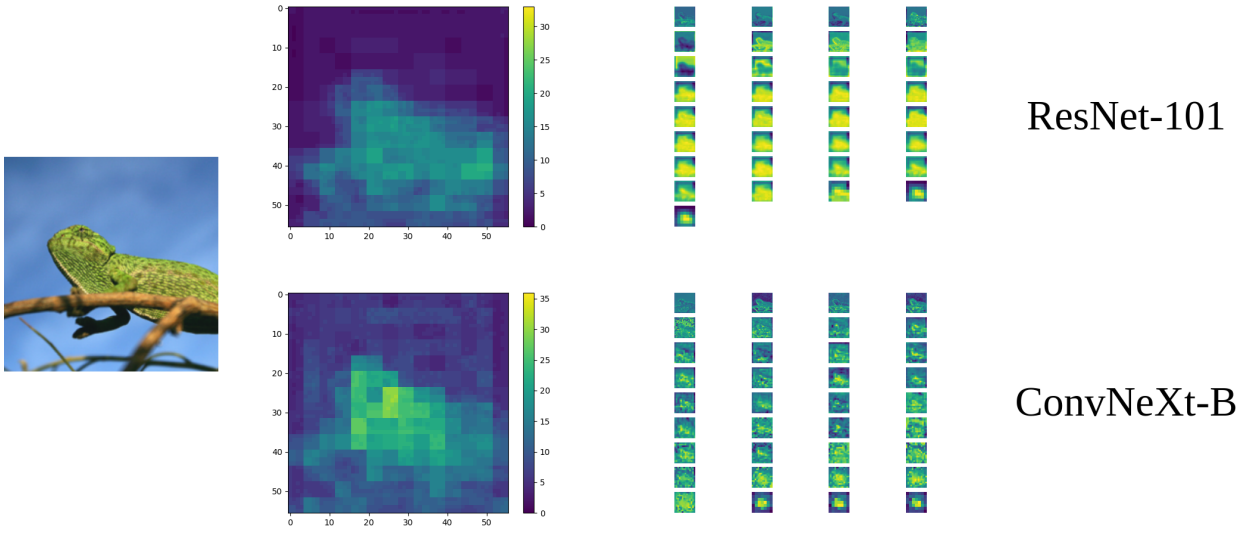


Figure 5.15: Aggregated layer masks (middle column) and individual masks (right column) for ResNet101 and ConvNeXt-B models.

5.5.5 Qualitative Results

A qualitative analysis of the proposed method is performed, by analyzing per-layer mask activations both individually and after aggregation (sum) for multiple sparsity budgets and networks, in Figures 5.12, 5.13, 5.14. Visualization of individual masks demonstrates how resource allocation can vary between each layer, revealing some key observations. There are some key layers where masks always have almost all their spatial locations activated, such as the first layer of ResNet-101 (Figure 5.12), the first two layers of ConvNeXt-T (Figure 5.13) and the first four layers of ConvNeXt-B (Figure 5.14). This leads us to the conclusion that it is crucial for the network to generate a complete feature map as early as possible. By inspecting the aggregated activation masks we observe that the network tends to focus on more critical details, while discarding less relevant areas, indicating an adaptive allocation of resources conditioned on the input image and the available computational resources. Such details include key features of the image such as the mouth and outline of the shark Figures (5.12, 5.13, 5.14). Remarkably, this behavior resembles that of a self-attention mechanism, where the model dynamically allocates attention to different parts of the input, emphasizing relevant features while suppressing noise or less informative regions.

In the comparative analysis of aggregated spatial mask activations between a ResNet-101 and ConvNeXt-B model (Figure 5.15), notable distinctions emerge. In the aggregated masks of ResNet-101 architecture, the separation between active and non-active regions is more distinct, indicating focused activation in key areas and minimal in background areas. Conversely, in masks of ConvNeXt-B model, the activations appear more distributed, with a higher presence in less relevant regions. This phenomenon can be attributed to the difference in the model architectures. ResNet-101 uses standard 3x3 kernels, leading to clear definition of critical areas, whereas ConvNeXt models utilize larger 7x7 convolution kernels with channel mixing, resulting in more widespread activations across the input space.

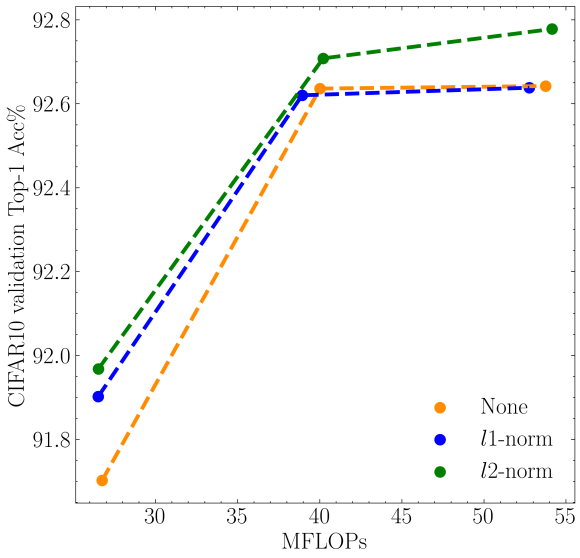
These observations shed light on the architectural differences between the two models and their respective mechanisms for processing and prioritizing spatial information, further enriching our understanding of spatially adaptive computation characteristics and potential applications.

5.6 Ablation Studies

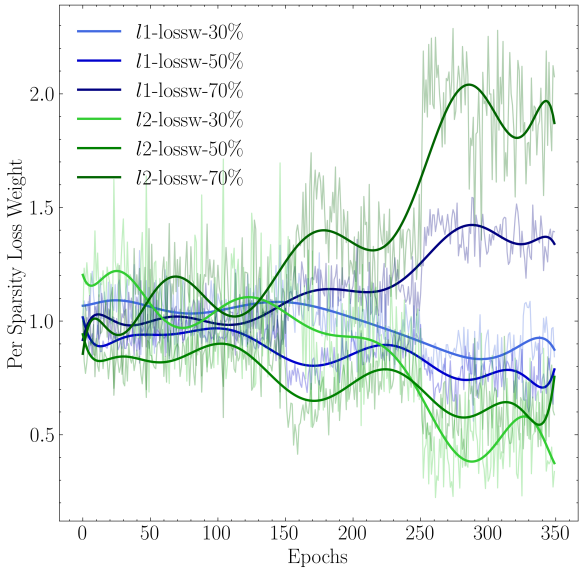
In this section, a series of ablation studies are conducted, to investigate the impact of various components and hyperparameters of our method. The results refer to the mean metrics from 10 training runs of ResNet-32 models trained on CIFAR10 dataset using the proposed method.

Target Awareness. We examine the impact of our target-aware loss weighting scheme and also compare l1 and l2 norms for the calculation of w^{b_i} weights in Figure 5.16a where the use of l2-norm outperforms l1-norm and no awareness weighting across all budgets. The evolution of loss weights during training (Figure 5.16b), shows that training focuses mostly on improving performance for the highest target budget after the halfway mark.

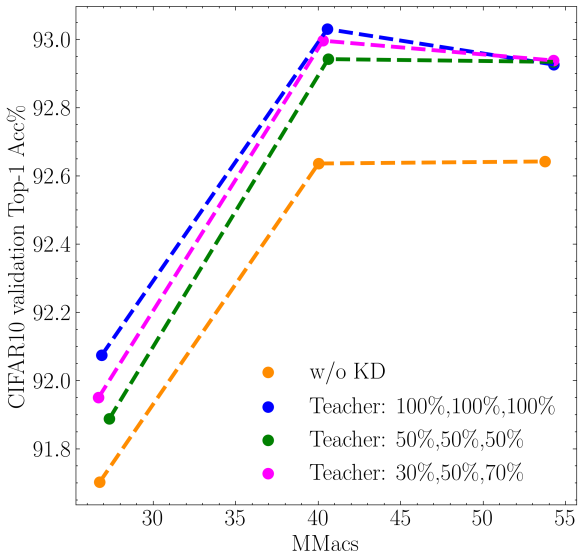
Knowledge Distillation. We conduct an ablation study over potential teachers between the base model (100%) and spatially adaptive models trained with DynConv (Verelst & Tuytelaars, 2020) to facilitate knowledge distillation. Figure 5.16c demonstrates that using the base model as teacher for all budgets yields the best performance. Furthermore, Figure 5.16d presents



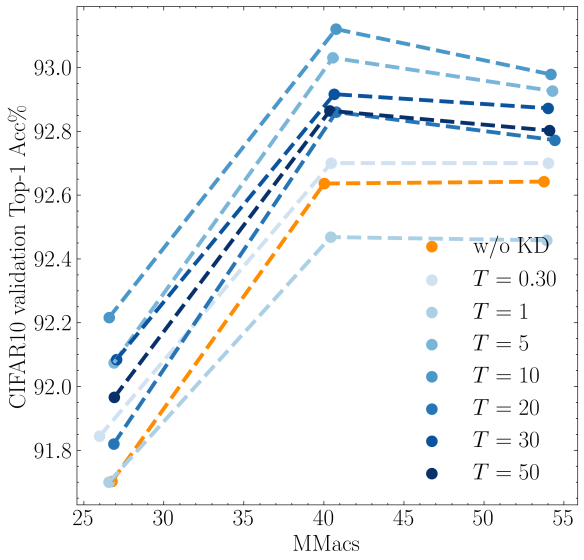
(a) Target Awareness norm



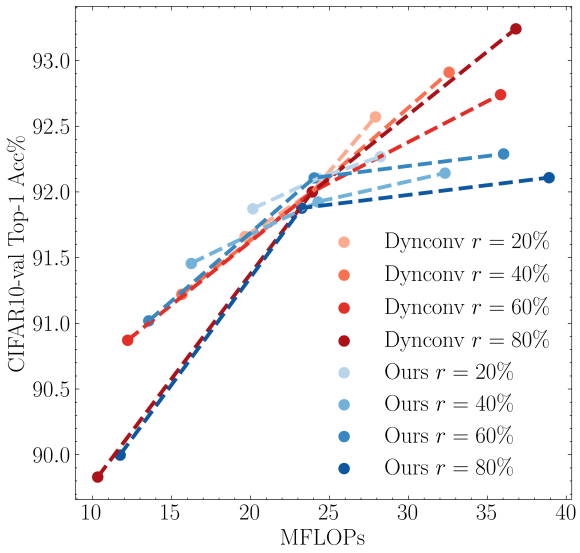
(b) Loss weights w^{b_i} during training



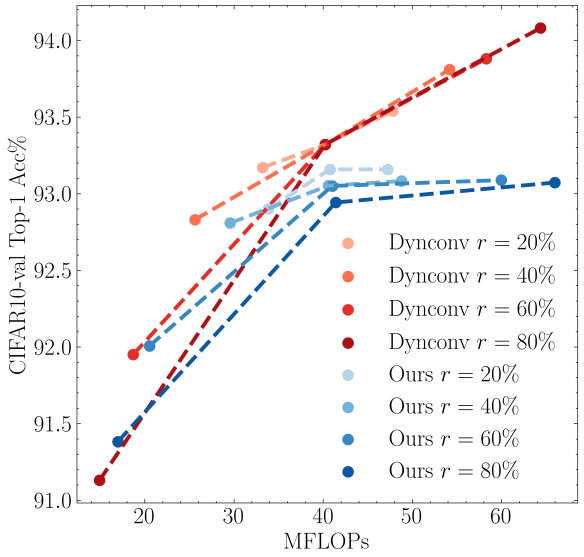
(c) KD Teacher Ablation



(d) KD Temperature T Ablation



(e) ResNet-20 Range Ablation



(f) ResNet-32 Range Ablation

Figure 5.16: Ablation studies for target awareness weighting, knowledge distillation setting and range of supported budgets.

performance for multiple values of the KD temperature T while using the base model as the teacher, with $T = 10$ being the most optimal.

Range of budgets. Finally, the performance of our method over multiple target budget ranges $r = 20\%, 40\%, 60\%, 70\%$ centered around 50% is examined in Figure 5.16e, 5.16f, for both ResNet-20 and ResNet-32. As expected, smaller range leads to better performance, whereas widening the target budget support range negatively impacts performance.

5.7 Conclusions

This work introduces a solution to address the multifaceted problem of dynamic inference at multiple FLOP budgets while adhering to memory constraints. It proposes a methodology to converting a base model to a spatially adaptive variant, thereby enabling to perform inference across various target FLOP budgets. By facilitating re-use of backbone weights and utilizing only mask estimation modules for each target budget, a marginal parameter overhead to the base model is maintained. Results demonstrate performance on par with existing optimization methods while showcasing enhanced scalability in parameter count, underscoring the effectiveness of our proposed approach in efficiently navigating trade-offs between performance, computational cost, and parameter count.

The key takeaways from this work are:

- There is an evident need to address the growing memory requirements when performing inference at multiple computational budgets.
- Our proposed method leverages the success of spatially adaptive methods and provides a memory efficient architecture to achieve dynamic inference.
- Results on baseline and modern state-of-the-art CNNs demonstrate both the efficacy and adaptability of our method, while highlighting its limitations, especially on the achieved accuracy.

Chapter 6

Conclusion

This chapter concludes the thesis. First, it presents an overview of the thesis by summarising the key contributions made. Second, it discusses the limitations of the work developed. Lastly, this chapter highlights possible future research avenues that are mainly motivated by the limitations discussed.

6.1 Summary of Thesis Achievements

This thesis presents a multifaceted exploration of spatially adaptive computation within the field of computer vision, visiting multiple tasks crucial to visual understanding. It introduces spatially adaptive computation techniques into state-of-the-art neural network architectures, yielding improvements in computational efficiency while maintaining or even enhancing performance across various trade-offs. In this section, we will revisit the research questions that were initially posed, highlighting the key contributions and presenting key insights and conclusions.

The first question focuses on leveraging spatial sparsity to increase the efficiency of stereo-matching CNNs. Through the adaptation of the StereoNet network, this research demonstrates how selectively skipping image regions with minimal contributions to disparity estimation can create a performance-to-computation trade-off. By controlling a sparsity factor, the approach

enables a dynamic range of operational points, which can adapt to available computational resources and execution speed requirements. This method, compatible with existing optimization techniques like channel pruning, highlights the potential of spatial sparsity to also deliver practical speedups, especially in high-resolution convolutional layers where computational redundancy is more pronounced.

The second question examines the computational aspects of spatially adaptive methods for semantic segmentation CNNs. The thesis presents an optimization methodology that integrates spatial sparsity into CNNs on a per-layer basis, controlled by a specified computational goal. Focus is given on advancing the state-of-the-art that dual-branch semantic segmentation networks have achieved and combine a data-driven approach to introduce spatial sparsity in their computations, formulated into an end-to-end trainable optimization process. To address the misalignment between theoretical FLOPs reductions and actual device latency, the framework incorporates latency profiling, ensuring networks are optimized for real-world performance on specific hardware. This latency-aware approach delivers better practical efficiency than hardware-agnostic methods by bridging computational optimization goals with device-specific characteristics, validating the viability of resulting networks for deployment in resource-constrained environments.

Finally, the thesis addresses the challenge of dynamic inference for multiple computational budgets while optimizing storage memory usage. A novel methodology is proposed to convert base models into spatially adaptive variants, enabling inference across varying computational budgets with minimal memory overhead. By reusing backbone weights and employing lightweight mask estimation modules, this approach achieves improved scalability and flexibility compared to existing optimization methods. The results highlight the effectiveness of the method in navigating trade-offs between accuracy, computational cost, and memory usage, making it particularly suitable for resource-constrained applications.

Through these contributions, the thesis establishes a comprehensive framework for leveraging spatially adaptive methods to enhance the efficiency, scalability, and adaptability of CNNs. These findings provide valuable insights into the design of deep learning solutions, especially

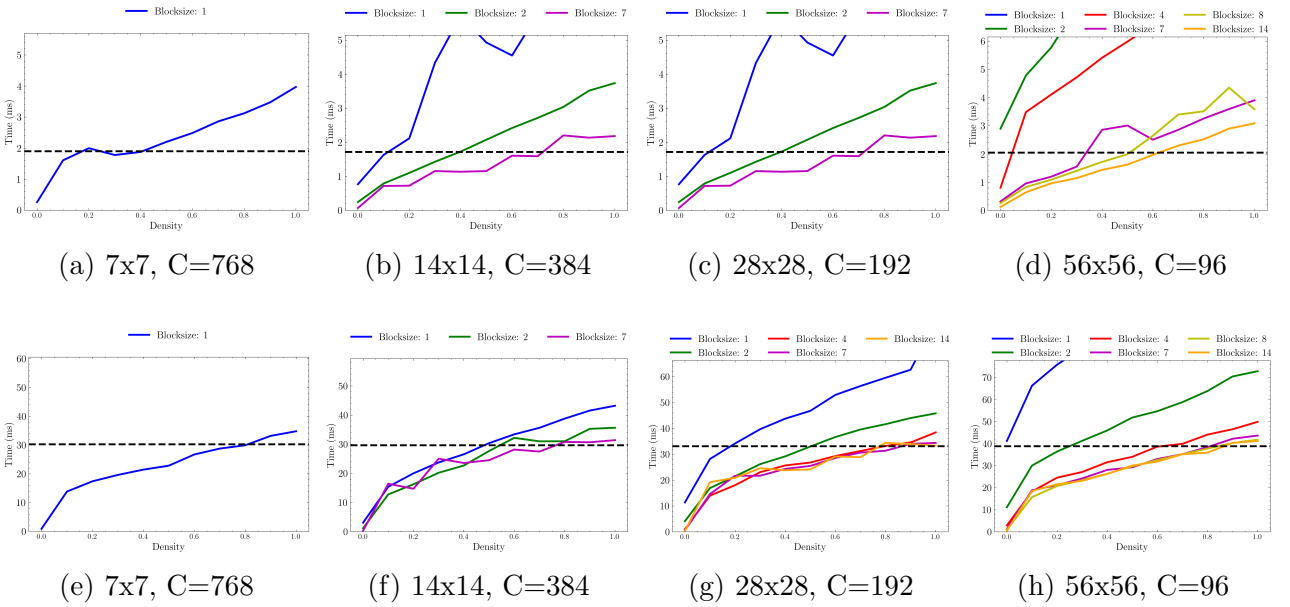
for computationally limited environments, paving the way for more efficient deployments in embedded systems and edge platforms.

Finally, by examining the experimental results of all chapters, it is evident that CNNs indeed exhibit some degree of redundancy in computations, which can be effectively mitigated by employing spatially adaptive processing. An important observation is the diminishing extent to which spatially adaptive methods can alleviate redundancy in more modern networks compared to older ones, suggesting that the overall efficiency and compactness of CNN models has increased throughout the years. Nevertheless, given the inherently sparse nature of visual data, we maintain confidence that spatially adaptive computation will continue to play a crucial role in enhancing the efficiency of deep learning models in the future.

6.2 Limitations and Challenges

To enable spatially adaptive processing, it is necessary to change the inherent structure of data, allowing for the application of varying processing techniques across different locations, which incurs notable overheads. These overheads, often disregarded when solely considering efficiency through FLOPs, significantly impact actual execution times, especially on devices optimized for dense computations such as GPUs. Therefore, to achieve tangible latency gains, the development of highly optimized software libraries tailored to the target platforms or customized hardware is imperative for efficiently managing the conversion between dense and sparse data formats. Despite the effectiveness of spatially adaptive computation, the availability of such optimized software libraries or hardware remains limited.

A potential reason for this when considering GPUs, is the reliance of most libraries on the closed source code of cuDNN for the convolution operation, given it's highly optimized nature. To take advantage of that, spatially adaptive processing processing frameworks focus solely on implementing data conversion methods, decoupled from the convolution operation. Such operations essentially convert dense data to sparse and vice-versa, by simply generating new dense structures with a subset of initial elements and the opposite. Therefore, convolution operation



(a) 7x7, C=768 (b) 14x14, C=384 (c) 28x28, C=192 (d) 56x56, C=96

(e) 7x7, C=768 (f) 14x14, C=384 (g) 28x28, C=192 (h) 56x56, C=96

Figure 6.1: Figure presenting the latency of executing different layers of ConvNeXt-T Z. Liu et al. (2022) under varying sparsity levels in two CPU devices: Intel i7-12700 (a-d) and an AMD Ryzen 3 1200 (e-h) under various blocksizes (colored lines). Dashed line denotes the dense execution time for the corresponding layer.

and sparse data conversion mechanisms remain decoupled, potentially leading to redundant data accessing and copying patterns. For this reason, it is important to consider solutions where data transformation and convolutions are jointly optimized to support computation under unstructured sparsity or using alternative data structures, able to handle spatially varying processing in a more efficient manner.

On the other hand CPUs allow more favorable mapping of spatially adaptive computation, since their efficiency is not greatly dependent on structure of data, compared to GPUs which are mostly optimized for dense computation. In this thesis, latency analysis has been explicitly focused on GPU devices. To investigate the potential performance of spatially adaptive computation approaches for CPU devices, we implemented the data conversion framework similar to SBNet Ren et al. (2018) for CPU devices, written in C language. Figure 6.1 presents results of benchmarking the individual layers of a modern CNN, ConvNeXt-T Z. Liu et al. (2022) on two desktop CPUs: an Intel i7-12700 with 14 cores and 20 threads, released in 2020 and an AMD Ryzen 3 1200 with 4 cores and 4 threads, released in 2017. We observe that latency gains are achievable and scale favorably as the local blocksize increases, suggesting that spatially adaptive inference can be used as a strategy to achieve practical latency gains on CPU devices.

6.3 Future Work

6.3.1 Future Research

Spatially adaptive processing demonstrated a consistent way of increasing the efficiency of computer vision models, by altering the structure of networks and modifying information flow in a data-driven manner. This approach yields a diverse set of models derived from a common base model.

As future work, it would be intriguing to systematically investigate the inherent properties of these models, investigating whether they possess distinctive characteristics or behaviors compared to conventional static architectures. To this end, a systematic examination is warranted to assess their performance in extreme scenarios and when confronted with out-of-distribution samples. These models inherently adapt computations to specific spatial regions, providing an unexplored property for scenarios involving noisy, low-quality, distorted, corrupted inputs, or artifacts. By systematically examining their performance in such extreme conditions and when confronted with OOD samples, we can gain valuable insights into their robustness and adaptability, and even uncover cases where they posses a competitive advantage against their static counterparts.

Moreover, it is also promising to explore spatially adaptive methods beyond the scope of model optimization. For example, Masked Image Modeling (MIM) He et al. (2022), which has demonstrated state-of-the-art results in pre-training large models, involves dividing an input image into patches and masking a portion of these patches. Then the model is then trained to predict the missing patches, learning an improved representation of the input image by capturing local and global patterns. While not focusing on improving inference efficiency, it varies the computation spatially between visible and masked regions, highlighting the importance of exploring spatially adaptive computation in different directions.

6.3.2 Short-term Extension

For short-term extensions of this work, a promising direction involves integrating spatially adaptive optimization with other optimization approaches, such as LR decomposition, pruning and/or quantization. The primary goal is to continue pushing efficiency gains, expecting further improvements in efficiency. Additionally, it is interesting to explore the interactions and trade-offs between different compression techniques. This investigation is expected to yield valuable insights into the synergies and potential compromises when combining spatially adaptive methods with other compression strategies, enhancing our understanding of efficient model deployment in practical settings.

6.3.3 Long-term Extension

Looking towards a long-term extension of this research, an intriguing avenue would be to develop a method that dynamically controls the allocation of computational resources within a model to accommodate varying resource availability while maintaining accurate performance conditioned also on the input complexity. Formulating this for the spatially adaptive computation, would involve creating a model capable of operating effectively across multiple computation budgets, adapting its structure and spatial computation allocation to suit changing requirements and input characteristics. Such an approach could also involve integrating a module that orchestrates the selection of optimal model structures and spatial computation strategies based on available resources and input complexity, ultimately enhancing the flexibility and efficiency of spatially adaptive models in real-world applications. This long-term vision aims to empower adaptive models to seamlessly adjust and optimize their computational behavior to deliver optimal performance under diverse computational constraints and input scenarios. Moreover, such an approach would also improve the energy efficiency of such systems, since less computation would be invested in inference of less challenging inputs. A decisive first step towards this vision is presented in Chapter 5 of this thesis, where we effectively compress multiple spatially adaptive models trained to operate at varying computational budgets, in a compact model with minimal parameter count overhead.

References

Abnar, S., & Zuidema, W. (2020, July). Quantifying attention flow in transformers. In D. Jurafsky, J. Chai, N. Schluter, & J. Tetreault (Eds.), *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 4190–4197). Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.acl-main.385

Adelson, E. H., Anderson, C. H., Bergen, J. R., Burt, P. J., & Ogden, J. M. (1984). Pyramid methods in image processing. *RCA engineer*, *29*(6), 33–41.

Ali, M., Zain, J. M., Zolkipli, M. F., & Badshah, G. (2015). Battery efficiency of mobile devices through computational offloading: A review. *2015 IEEE Student Conference on Research and Development (SCOReD)*, 317–322. https://doi.org/10.1109/SCORED.2015.7449347

Anandan, P. (1989). A computational framework and an algorithm for the measurement of visual motion. *Int. J. Comput. Vis.*, *2*(3), 283–310. http://dblp.uni-trier.de/db/journals/ijcv/ijcv2.html#Anandan89

Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, *39*(12), 2481–2495.

Ballard, D. H., & Brown, C. M. (1982). *Computer vision* (1st). Prentice Hall Professional Technical Reference.

Banner, R., Nahshan, Y., & Soudry, D. (2019). Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems*, *32*.

Bantock, J. R. B., Tenentes, V., Al-Hashimi, B. M., & Merrett, G. V. (2017). Online tuning of dynamic power management for efficient execution of interactive workloads. *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 1–6. https://doi.org/10.1109/ISLPED.2017.8009195

Barnes, C., Shechtman, E., Finkelstein, A., & Goldman, D. B. (2009). PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics*. https://doi.org/10.1145/1531326.1531330

Bay, H., Ess, A., Tuytelaars, T., & Van Gool, L. (2008). Speeded-up robust features (surf) [Similarity Matching in Computer Vision and Multimedia]. *Computer Vision and Image Understanding*, *110*(3), 346–359. https://doi.org/https://doi.org/10.1016/j.cviu.2007.09.014

Biederman, I. (1985). Human image understanding: Recent research and a theory. *Computer Vision, Graphics, and Image Processing*, *32*(1), 29–73. https://doi.org/https://doi.org/10.1016/0734-189X(85)90002-7

Boykov, Y., Veksler, O., & Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *23*(11), 1222–1239. https://doi.org/10.1109/34.969114

Brosnan, T., & Sun, D.-W. (2002). Inspection and grading of agricultural and food products by computer vision systems—a review. *Computers and Electronics in Agriculture*, *36*(2), 193–213. https://doi.org/https://doi.org/10.1016/S0168-1699(02)00101-1

Brosnan, T., & Sun, D.-W. (2004). Improving quality inspection of food products by computer vision—a review [Applications of computer vision in the food industry]. *Journal of Food Engineering*, *61*(1), 3–16. https://doi.org/https://doi.org/10.1016/S0260-8774(03)00183-3

Brostow, G. J., Fauqueur, J., & Cipolla, R. (2009). Semantic object classes in video: A high-definition ground truth database [Video-based Object and Event Analysis]. *Pattern Recognition Letters*, *30*(2), 88–97. https://doi.org/https://doi.org/10.1016/j.patrec.2008.04.005

Cadieu, C. F., Hong, H., Yamins, D. L., Pinto, N., Ardila, D., Solomon, E. A., Majaj, N. J., & DiCarlo, J. J. (2014). Deep neural networks rival the representation of primate it cortex for core visual object recognition. *PLoS computational biology*, *10*(12), e1003963.

Cai, S., Shu, Y., & Wang, W. (2021). Dynamic routing networks. *proceedings of the IEEE/CVF winter conference on applications of computer vision*, 3588–3597.

Cao, Q., & Zhang, H. (2017). Combined holistic and local patches for recovering 6d object pose. *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2219–2227. https://doi.org/10.1109/ICCVW.2017.259

Chabra, R., Straub, J., Sweeney, C., Newcombe, R., & Fuchs, H. (2019). Stereodrnet: Dilated residual stereonet. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2019-June*, 11778–11787. https://doi.org/10.1109/CVPR.2019.01206

Chang, J.-R., Chang, P.-C., & Chen, Y.-S. (2020). Attention-aware feature aggregation for real-time stereo matching on edge devices. *Proceedings of the Asian Conference on Computer Vision (ACCV)*.

Chang, J., & Chen, Y. (2018). Pyramid stereo matching network. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, *abs/1803.08669*.

Chefer, H., Gur, S., & Wolf, L. (2021). Transformer interpretability beyond attention visualization. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 782–791.

Chen, C., Li, L., & Sabry Aly, M. M. (2024). Vita: A highly efficient dataflow and architecture for vision transformers. *2024 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 1–6. https://doi.org/10.23919/DATE58400.2024.10546565

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2015). Semantic image segmentation with deep convolutional nets and fully connected crfs. *ICLR*. http://arxiv.org/abs/1412.7062

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2018). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*(4), 834–848. https://doi.org/10.1109/TPAMI.2017.2699184

Chen, L.-C., Papandreou, G., Schroff, F., & Adam, H. (2017). Rethinking atrous convolution for semantic image segmentation.

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. *Proceedings of the European conference on computer vision (ECCV)*, 801–818.

Chen, M., Lin, M., Li, K., Shen, Y., Wu, Y., Chao, F., & Ji, R. (2023). Cf-vit: A general coarse-to-fine method for vision transformer. *Proceedings of the AAAI Conference on Artificial Intelligence*, *37*(6), 7042–7052.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1800–1807. https://doi.org/10.1109/CVPR.2017.195

Coifman, B., Beymer, D., McLauchlan, P., & Malik, J. (1998). A real-time computer vision system for vehicle tracking and traffic surveillance. *Transportation Research Part C: Emerging Technologies*, *6*(4), 271–288. https://doi.org/https://doi.org/10.1016/S0968-090X(98)00019-9

Coles, S. (2001). An introduction to statistical modeling of extreme values. *Journal of the American Statistical Association*, *97*, 1204–1204. https://api.semanticscholar.org/CorpusID:19678794

Cootes, T. F., & Taylor, C. J. (2001). Statistical models of appearance for medical image analysis and computer vision. In M. Sonka & K. M. Hanson (Eds.), *Medical imaging 2001: Image processing* (pp. 236–248, Vol. 4322). SPIE. https://doi.org/10.1117/12.431093

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., & Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3213–3223.

Dandois, J. P., & Ellis, E. C. (2010). Remote sensing of vegetation structure using computer vision. *Remote Sensing*, *2*(4), 1157–1176. https://doi.org/10.3390/rs2041157

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE conference on computer vision and pattern recognition*, 248–255.

Deng, L., Li, G., Han, S., Shi, L., & Xie, Y. (2020). Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, *108*(4), 485–532. https://doi.org/10.1109/JPROC.2020.2976475

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, *29*(6), 141–142. https://doi.org/10.1109/MSP.2012.2211477

Denning, D. E. (1987). An intrusion-detection model. *IEEE Transactions on software engineering*, (2), 222–232.

Dhanachandra, N., Manglem, K., & Chanu, Y. J. (2015). Image segmentation using k -means clustering algorithm and subtractive clustering algorithm [Eleventh International Conference on Communication Networks, ICCN 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Data Mining and Warehousing, ICDMW 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Image and Signal Processing, ICISP 2015, August 21-23, 2015, Bangalore, India]. *Procedia Computer Science*, *54*, 764–771. https://doi.org/https://doi.org/10.1016/j.procs.2015.06.090

Doi, K. (2007). Computer-aided diagnosis in medical imaging: Historical review, current status and future potential [Computer-aided Diagnosis (CAD) and Image-guided Decision Support]. *Computerized Medical Imaging and Graphics*, *31*(4), 198–211. https://doi.org/https://doi.org/10.1016/j.compmedimag.2007.02.002

Domínguez-Morales, M., Jiménez-Fernandez, A., Paz-Vicente, R., Linares-Barranco, A., & Jimenez, G. (2012, July). Stereo matching: From the basis to neuromorphic engineering. IntechOpen. https://doi.org/10.5772/45901

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Du, X., El-Khamy, M., & Lee, J. (2019). AMNet: Deep Atrous Multiscale Stereo Disparity Estimation Networks, 1–25. http://arxiv.org/abs/1904.09099

Duggal, S., Wang, S., Ma, W.-C., Hu, R., & Urtasun, R. (2019). Deeppruner: Learning efficient stereo matching via differentiable patchmatch. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Everingham, M., Eslami, S. A., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2015). The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, *111*, 98–136.

Feng, C., Han, P., Zhang, X., Yang, B., Liu, Y., & Guo, L. (2022). Computation offloading in mobile edge computing networks: A survey. *Journal of Network and Computer Applications*, *202*, 103366. https://doi.org/https://doi.org/10.1016/j.jnca.2022.103366

Figurnov, Ibraimova, Vetrov, & Kohli. (2016). Perforatedcnns: Acceleration through elimination of redundant convolutions. *Advances in neural information processing systems*, *29*.

Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., & Salakhutdinov, R. (2017). Spatially adaptive computation time for residual networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1039–1048.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, *36*(4), 193–202.

Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, *1*(2), 119–130.

Fukushima, K., Miyake, S., & Ito, T. (1983). Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (5), 826–834.

Gao, R. (2023). Rethinking dilated convolution for real-time semantic segmentation. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 4675–4684. https://doi.org/10.1109/CVPRW59228.2023.00493

Georgiou, T., Liu, Y., Chen, W., & Lew, M. S. (2019). A survey of traditional and deep learning-based feature descriptors for high dimensional data in computer vision. *International Journal of Multimedia Information Retrieval*, *9*, 135–170. https://api.semanticscholar.org/CorpusID:208210895

Gottesfeld, B. L. (1992). A survey of image registration techniques. *ACM Comput. Surv.*, *24*(4), 325–376. https://doi.org/10.1145/146370.146374

Graham, B., & van der Maaten, L. (2017). Submanifold sparse convolutional networks.

Graves, A. (2016). Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

Griffin, G., Holub, A., & Perona, P. (2007, March). *Caltech-256 object category dataset*.

Gumbel, E. J. (1958). *Statistics of extremes*. Columbia university press.

Guo, C., Hsueh, B. Y., Leng, J., Qiu, Y., Guan, Y., Wang, Z., Jia, X., Li, X., Guo, M., & Zhu, Y. (2020). Accelerating sparse dnn models without hardware-support via tile-wise sparsity. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–15. https://doi.org/10.1109/SC41405.2020.00020

Haase, D., & Amthor, M. (2020). Rethinking depthwise separable convolutions: How intra-kernel correlations lead to improved mobilenets. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 14588–14597. https://doi.org/10.1109/CVPR42600.2020.01461

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., & Wang, Y. (2021). Dynamic neural networks: A survey. https://doi.org/10.48550/ARXIV.2102.04906

Han, Y., Huang, G., Song, S., Yang, L., Zhang, Y., & Jiang, H. (2021). Spatially adaptive feature refinement for efficient inference. *IEEE Transactions on Image Processing*, *30*, 9345–9358.

Han, Y., Liu, Z., Yuan, Z., Pu, Y., Wang, C., Song, S., & Huang, G. (2023). Latency-aware unified dynamic networks for efficient image recognition.

Han, Y., Yuan, Z., Pu, Y., Xue, C., Song, S., Sun, G., & Huang, G. (2022). Latency-aware spatial-wise dynamic networks. *Advances in Neural Information Processing Systems*, *35*, 36845–36857.

Hannah, M. J. (1974). Computer matching of areas in stereo images. https://api.semanticscholar.org/CorpusID:62672515

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., & Girshick, R. (2022). Masked autoencoders are scalable vision learners. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 16000–16009.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, *37*(9), 1904–1916.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Heimann, T., & Meinzer, H.-P. (2009). Statistical shape models for 3d medical image segmentation: A review. *Medical Image Analysis*, *13*(4), 543–563. https://doi.org/https://doi.org/10.1016/j.media.2009.05.004

Hernandez-Juarez, D., Chacón, A., Espinosa, A., Vázquez, D., Moure, J., & López, A. (2016). Embedded real-time stereo estimation via semi-global matching on the gpu [International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA]. *Procedia Computer Science*, *80*, 143–153. https://doi.org/https://doi.org/10.1016/j.procs.2016.05.305

Hirschmüller, H. (2008). Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *30*(2), 328–341. https://doi.org/10.1109/TPAMI.2007.1166
BASIC PAPER gia SGM. Se auto egine introduce apo hirschmuller.

Hoang, V.-T., & Jo, K.-H. (2019). Practical analysis on running speed for efficient cnn architecture design. *2019 12th International Conference on Human System Interaction (HSI)*, 221–224. https://doi.org/10.1109/HSI47298.2019.8942599

Hong, Y., Pan, H., Sun, W., & Jia, Y. (2021). Deep dual-resolution networks for real-time and accurate semantic segmentation of road scenes. *arXiv preprint arXiv:2101.06085*.

Hou, Q., Zhou, D., & Feng, J. (2021). Coordinate attention for efficient mobile network design. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 13713–13722.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. (2019). Searching for mobilenetv3. *Proceedings of the IEEE/CVF international conference on computer vision*, 1314–1324.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

https://developersbreach.com/convolution-neural-network-deep-learning/ [Accessed: Date of Access]. (n.d.). https://developersbreach.com/convolution-neural-network-deep-learning

https://hips.hearstapps.com/hmg-prod/images/dog-puppy-on-garden-royalty-free-image-1586966191.jpg?
[Accessed: Date of Access]. (n.d.). https://hips.hearstapps.com/hmg-prod/images/dog-puppy-on-garden-royalty-free-image-1586966191.jpg?ref=blog.paperspace.com

Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7132–7141.

Huang, Z., & Wang, N. (2018). Data-driven sparse structure selection for deep neural networks. *Proceedings of the European conference on computer vision (ECCV)*, 304–320.

Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, *160*(1), 106.

Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., & Brox, T. (2016). FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks. https://doi.org/10.1109/CVPR.2017.179

Jang, E., Gu, S., & Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Jang E. (2020). Tutorial: Categorical variational autoencoders using gumbel-softmax.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., & Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.

Jiang, Z.-H., Hou, Q., Yuan, L., Zhou, D., Shi, Y., Jin, X., Wang, A., & Feng, J. (2021). All tokens matter: Token labeling for training better vision transformers. *Advances in neural information processing systems*, *34*, 18590–18602.

Kanade, T., Kano, H., Kimura, S., Yoshida, A., & Oda, K. (1995). Development of a video-rate stereo machine. *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, *3*, 95–100 vol.3. https://doi.org/10.1109/IROS.1995.525868

Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., & Tang, L. (2017). Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *SIGARCH Comput. Archit. News*, *45*(1), 615–629. https://doi.org/10.1145/3093337.3037698

Kass, M., Witkin, A. P., & Terzopoulos, D. (2004). Snakes: Active contour models. *International Journal of Computer Vision*, *1*, 321–331. https://api.semanticscholar.org/CorpusID:12849354

Kendall, A., Martirosyan, H., Dasgupta, S., Henry, P., Kennedy, R., Bachrach, A., & Bry, A. (2017). End-to-End Learning of Geometry and Context for Deep Stereo Regression.

Proceedings of the IEEE International Conference on Computer Vision, *2017-Octob*, 66–75. https://doi.org/10.1109/ICCV.2017.17

Khamis, S., Fanello, S., Rhemann, C., Kowdle, A., Valentin, J., & Izadi, S. (2018). Stereonet: Guided hierarchical refinement for real-time edge-aware depth prediction. *Proceedings of the European Conference on Computer Vision (ECCV)*, 573–590.

Krizhevsky, A. (2012). Learning multiple layers of features from tiny images. *University of Toronto*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, & K. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 25). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Kumar, A. (2008). Computer-vision-based fabric defect detection: A survey. *IEEE Transactions on Industrial Electronics*, *55*(1), 348–363. https://doi.org/10.1109/TIE.1930.896476

Lacoste, A., Lehmann, N., Rodriguez, P., Sherwin, E., Kerner, H., Lütjens, B., Irvin, J., Dao, D., Alemohammad, H., Drouin, A., Gunturkun, M., Huang, G., Vazquez, D., Newman, D., Bengio, Y., Ermon, S., & Zhu, X. (2023). Geo-bench: Toward foundation models for earth monitoring. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, & S. Levine (Eds.), *Advances in neural information processing systems* (pp. 51080–51093, Vol. 36). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2023/file/a0644215d9cff6646fa334dfa5d29c5a-Paper-Datasets_and_Benchmarks.pdf

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436–444.

Lei, G., Li, A., Bian, J., Naboureh, A., Zhang, Z., & Nan, X. (2023). A simple and automatic method for detecting large-scale land cover changes without training data. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *PP*, 1–17. https://doi.org/10.1109/JSTARS.2023.3299413

Li, E., Zeng, L., Zhou, Z., & Chen, X. (2019). Edge ai: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications*, *19*(1), 447–457.

Li, H., Xiong, P., Fan, H., & Sun, J. (2019). Dfanet: Deep feature aggregation for real-time semantic segmentation. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 9522–9531.

Li, Y., Song, L., Chen, Y., Li, Z., Zhang, X., Wang, X., & Sun, J. (2020). Learning dynamic routing for semantic segmentation. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8553–8562.

Li, Y., Geller, T., Kim, Y., & Panda, P. (2023). Seenn: Towards temporal spiking early exit neural networks. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, & S. Levine (Eds.), *Advances in neural information processing systems* (pp. 63327–63342, Vol. 36). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2023/file/c801e68207da477bbc44182b9fac1129-Paper-Conference.pdf

Lin, S., Ji, R., Yan, C., Zhang, B., Cao, L., Ye, Q., Huang, F., & Doermann, D. (2019). Towards optimal structured cnn pruning via generative adversarial learning. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2790–2799.

Lin, Z., Wang, Y., Zhang, J., & Chu, X. (2023). Dynamicdet: A unified dynamic architecture for object detection. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 6282–6291.

Liu, J., Tang, D., Huang, Y., Zhang, L., Zeng, X., Li, D., Lu, M., Peng, J., Wang, Y., Jiang, F., et al. (2024). Updp: A unified progressive depth pruner for cnn and vision transformer. *arXiv preprint arXiv:2401.06426*.

Liu, X., Peng, H., Zheng, N., Yang, Y., Hu, H., & Yuan, Y. (2023). Efficientvit: Memory efficient vision transformer with cascaded group attention. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 14420–14430.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., & Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. *Proceedings of the IEEE/CVF international conference on computer vision*, 10012–10022.

Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.-T., & Sun, J. (2019). Metapruning: Meta learning for automatic neural network channel pruning. *Proceedings of the IEEE/CVF international conference on computer vision*, 3296–3305.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., & Xie, S. (2022). A convnet for the 2020s. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 11976–11986.

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3431–3440.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. *Proceedings of the seventh IEEE international conference on computer vision*, *2*, 1150–1157.

Lu, C., Uchiyama, H., Thomas, D., Shimada, A., & Taniguchi, R.-i. (2018). Sparse Cost Volume for Ef fi cient Stereo Matching, 1–12. https://doi.org/10.3390/rs10111844

Ma, N., Zhang, X., Zheng, H.-T., & Sun, J. (2018). Shufflenet v2: Practical guidelines for efficient cnn architecture design. *Proceedings of the European conference on computer vision (ECCV)*, 116–131.

Maddison, C. J., Mnih, A., & Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Maintz, J., & Viergever, M. A. (1998). A survey of medical image registration. *Medical Image Analysis*, *2*(1), 1–36. https://doi.org/https://doi.org/10.1016/S1361-8415(01)80026-8

Marr, D., & Hildreth, E. (1980). Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, *207*(1167), 187–217.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, ... Xiaoqiang Zheng. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems [Software available from tensorflow.org]. https://www.tensorflow.org/

Mayer, N., Ilg, E., Häusser, P., Fischer, P., Cremers, D., Dosovitskiy, A., & Brox, T. (2016). A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4040–4048. https://doi.org/10.1109/CVPR.2016.438

Mazzini, D. (2018). Guided upsampling network for real-time semantic segmentation. *British Machine Vision Conference 2018, BMVC 2018, Newcastle, UK, September 3-6, 2018*, 117. http://bmvc2018.org/contents/papers/0423.pdf

Menze, M., Heipke, C., & Geiger, A. (2015). Joint 3d estimation of vehicles and scene flow. *ISPRS Workshop on Image Sequence Analysis (ISA)*.

Menze, M., Heipke, C., & Geiger, A. (2018). Object scene flow. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*.

Molchanov, P., Mallya, A., Tyree, S., Frosio, I., & Kautz, J. (2019). Importance estimation for neural network pruning. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Moons, T., Van Gool, L., & Vergauwen, M. (2010).

Moor, M., Banerjee, O., Abad, Z. S. H., Krumholz, H. M., Leskovec, J., Topol, E. J., & Rajpurkar, P. (2023). Foundation models for generalist medical artificial intelligence. *Nature*, *616*(7956), 259–265.

Nevatia, R., & Babu, K. R. (1980). Linear feature extraction and description. *Computer graphics and image processing*, *13*(3), 257–269.

Nirkin, Y., Wolf, L., & Hassner, T. (2021). HyperSeg: Patch-wise hypernetwork for real-time semantic segmentation. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Nock, R., & Nielsen, F. (2004). Statistical region merging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*(11), 1452–1458. https://doi.org/10.1109/TPAMI.2004.110

Oliver, N., Rosario, B., & Pentland, A. (2000). A bayesian computer vision system for modeling human interactions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(8), 831–843. https://doi.org/10.1109/34.868684

O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G. V., Krpalkova, L., Riordan, D., & Walsh, J. (2020). Deep learning vs. traditional computer vision. *Advances in Computer Vision: Proceedings of the 2019 Computer Vision Conference (CVC), Volume 1 1*, 128–144.

Orsic, M., Kreso, I., Bevandic, P., & Segvic, S. (2019). In defense of pre-trained imagenet architectures for real-time semantic segmentation of road-driving images. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12607–12616.

O'shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Pang, J., Sun, W., Ren, J. S., Yang, C., & Yan, Q. (2018). Cascade Residual Learning: A Two-Stage Convolutional Neural Network for Stereo Matching. *Proceedings - 2017 IEEE International Conference on Computer Vision Workshops, ICCVW 2017*, *2018-Janua*, 878–886. https://doi.org/10.1109/ICCVW.2017.108

Papageorgiou, C. P., Oren, M., & Poggio, T. (1998). A general framework for object detection. *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*, 555–562.

Passalis, N., Raitoharju, J., Tefas, A., & Gabbouj, M. (2020). Efficient adaptive inference for deep convolutional neural networks using hierarchical early exits. *Pattern Recognition*, *105*, 107346. https://doi.org/https://doi.org/10.1016/j.patcog.2020.107346

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Plath, N., Toussaint, M., & Nakajima, S. (2009). Multi-class image segmentation using conditional random fields and global classification. *Proceedings of the 26th Annual International Conference on Machine Learning*, 817–824. https://doi.org/10.1145/1553374.1553479

Poggio, T. (1984). Vision by man and machine. *Scientific American*, *250*(4), 106–117. Retrieved August 27, 2024, from http://www.jstor.org/stable/24969349

Radosavovic, I., Kosaraju, R. P., Girshick, R., He, K., & Dollár, P. (2020). Designing network design spaces. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 10428–10436.

Rao, Y., Liu, Z., Zhao, W., Zhou, J., & Lu, J. (2023). Dynamic spatial sparsification for efficient vision transformers and convolutional neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Rao, Y., Zhao, W., Liu, B., Lu, J., Zhou, J., & Hsieh, C.-J. (2021). Dynamicvit: Efficient vision transformers with dynamic token sparsification. *Advances in neural information processing systems*, *34*, 13937–13949.

Rawat, W., & Wang, Z. (2017). Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, *29*(9), 2352–2449. https://doi.org/10.1162/neco_a_00990

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 779–788.

Ren, M., Pokrovsky, A., Yang, B., & Urtasun, R. (2018). Sbnet: Sparse blocks network for fast inference. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8711–8720.

Ridnik, T., Ben-Baruch, E., Noy, A., & Zelnik-Manor, L. (2021). Imagenet-21k pretraining for the masses. *arXiv preprint arXiv:2104.10972*.

Riesenhuber, M., & Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nature neuroscience*, *2*(11), 1019–1025.

Romera, E., Álvarez, J. M., Bergasa, L. M., & Arroyo, R. (2018). Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, *19*(1), 263–272. https://doi.org/10.1109/TITS.2017.2750080

Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). Orb: An efficient alternative to sift or surf. *2011 International Conference on Computer Vision*, 2564–2571. https://doi.org/10.1109/ICCV.2011.6126544

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*, 211–252.

Ryali, C., Hu, Y.-T., Bolya, D., Wei, C., Fan, H., Huang, P.-Y., Aggarwal, V., Chowdhury, A., Poursaeed, O., Hoffman, J., et al. (2023). Hiera: A hierarchical vision transformer without the bells-and-whistles. *International Conference on Machine Learning*, 29441–29454.

Saksena, S. (2020, August). Cabinet : Efficient context aggregation network for low-latency semantic segmentation. http://essay.utwente.nl/84370/

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520.

Scharstein, D., Szeliski, R., & Zabih, R. (2001). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*, 131–140. https://doi.org/10.1109/SMBV.2001.988771

Scharstein, D., & Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, *47*, 7–42.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. *2017 IEEE International Conference on Computer Vision (ICCV)*, 618–626. https://doi.org/10.1109/ICCV.2017.74

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*.

Stockham, T. G. (1972). Image processing in the context of a visual model. *Proceedings of the IEEE*, *60*(7), 828–842.

Stockman, G., Kopstein, S., & Benett, S. (1982). Matching images to models for registration and object detection via clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (3), 229–241.

Stoyanov, D., Scarzanella, M. V., Pratt, P., & Yang, G.-Z. (2010). Real-time stereo reconstruction in robotically assisted minimally invasive surgery. *Medical image computing and computer-assisted intervention : MICCAI ... International Conference on Medical Image Computing and Computer-Assisted Intervention*, *13 Pt 1*, 275–82. https://api.semanticscholar.org/CorpusID:10156501

Sun, J., Shum, H. Y., & Zheng, N. N. (2002). Stereo matching using belief propagation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *2351*(7), 510–524. https://doi.org/10.1007/3-540-47967-8_34

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *International conference on machine learning*, 6105–6114.

Tan, M., & Le, Q. V. (2021). Efficientnetv2: Smaller models and faster training.

Teerapittayanon, S., McDanel, B., & Kung, H.-T. (2016). Branchynet: Fast inference via early exiting from deep neural networks. *2016 23rd international conference on pattern recognition (ICPR)*, 2464–2469.

Tomasi, C., & Kanade, T. (1991). Detection and tracking of point. *Int J Comput Vis*, *9*(137-154), 3.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., & Jégou, H. (2021). Training data-efficient image transformers & distillation through attention. *International conference on machine learning*, 10347–10357.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Vedaldi, A., & Lenc, K. (2015). Matconvnet: Convolutional neural networks for matlab. *Proceedings of the 23rd ACM International Conference on Multimedia*, 689–692. https://doi.org/10.1145/2733373.2807412

Verelst, T., & Tuytelaars, T. (2020). Dynamic convolutions: Exploiting spatial sparsity for faster inference. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr42600.2020.00239

Verelst, T., & Tuytelaars, T. (2023). Segblocks: Block-based dynamic resolution networks for real-time segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *45*(2), 2400–2411. https://doi.org/10.1109/TPAMI.2022.3162528

Voulodimos, A., Doulamis, N. D., Doulamis, A. D., & Protopapadakis, E. E. (2018). Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience*, *2018*. https://api.semanticscholar.org/CorpusID:3557281

Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., Liu, W., & Xiao, B. (2021). Deep high-resolution representation learning for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*(10), 3349–3364. https://doi.org/10.1109/TPAMI.2020.2983686

Wang, K., Cao, J., Zhou, Z., & Li, Z. (2024). Swapnet: Efficient swapping for dnn inference on edge ai devices beyond the memory budget. *IEEE Transactions on Mobile Computing*.

Wang, L., Dong, X., Wang, Y., Ying, X., Lin, Z., An, W., & Guo, Y. (2021). Exploring sparsity in image super-resolution for efficient inference. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 4917–4926.

Wang, Y., Lai, Z., Huang, G., Wang, B. H., van der Maaten, L., Campbell, M., & Weinberger, K. Q. (2019). Anytime stereo image depth estimation on mobile devices. *2019 International Conference on Robotics and Automation (ICRA)*, 5893–5900. https://doi.org/10.1109/ICRA.2019.8794003

Wang, Z. (2020). Sparsert: Accelerating unstructured sparsity on gpus for deep learning inference. *Proceedings of the ACM international conference on parallel architectures and compilation techniques*, 31–42.

Weng, J., Ahuja, N., & Huang, T. S. (1992). Cresceptron: A self-organizing neural network which grows adaptively. *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, *1*, 576–581.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Xie, C.-W., Zhou, H.-Y., & Wu, J. (2018). Vortex Pooling: Improving Context Representation in Semantic Segmentation. http://arxiv.org/abs/1804.06242

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1492–1500.

Xie, Z., Zhang, Z., Zhu, X., Huang, G., & Lin, S. (2020). Spatially adaptive inference with stochastic feature sampling and interpolation. *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, 531–548.

Xu, H., & Zhang, J. (2020). Aanet: Adaptive aggregation network for efficient stereo matching. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yu, C., Gao, C., Wang, J., Yu, G., Shen, C., & Sang, N. (2020). Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation.

Yu, C., Wang, J., Peng, C., Gao, C., Yu, G., & Sang, N. (2018). Bisenet: Bilateral segmentation network for real-time semantic segmentation.

Yu, F., & Koltun, V. (2016). Multi-scale context aggregation by dilated convolutions. *International Conference on Learning Representations*.

Yu, M., Wang, A., Chen, D., Yu, H., Luo, X., Li, Z., Wang, W., Chen, R., Nie, D., & Yang, H. (2023). Faaswap: Slo-aware, gpu-efficient serverless inference via model swapping. *arXiv preprint arXiv:2306.03622*.

Yu, S., Yao, Z., Gholami, A., Dong, Z., Kim, S., Mahoney, M. W., & Keutzer, K. (2022). Hessian-aware pruning and optimal neural implant. *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 3880–3891.

Zabih, R., & Woodfill, J. (1994). Non-parametric local transforms for computing visual correspondence. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *801 LNCS*, 151–158. https://doi.org/10.1007/bfb0028345

Zampokas, G., Bouganis, C.-S., & Tzovaras, D. (2022). Pushing the efficiency of stereonet: Exploiting spatial sparsity. *VISIGRAPP*.

Zampokas, G., Bouganis, C.-S., & Tzovaras, D. (2024). Latency driven spatially sparse optimization for multi-branch cnns for semantic segmentation. *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops*, 939–947.

Zampokas, G., Tsiolis, K., Peleka, G., Mariolis, I., Malasiotis, S., & Tzovaras, D. (2018). Real-time 3d reconstruction in minimally invasive surgery with quasi-dense matching. *2018 IEEE International Conference on Imaging Systems and Techniques (IST)*, 1–6. https://doi.org/10.1109/IST.2018.8577106

Žbontar, J., & LeCun, Y. (2015). Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches. *17*, 1–32. https://doi.org/10.1103/PhysRevE.93.033307

Zhang, F., Prisacariu, V., Yang, R., & Torr, P. H. (2019). GA-net: Guided aggregation net for end-to-end stereo matching. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2019-June*, 185–194. https://doi.org/10.1109/CVPR.2019.00027

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 6848–6856.

Zhang, X., Zou, J., He, K., & Sun, J. (2015). Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, *38*(10), 1943–1955.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning deep features for discriminative localization. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2921–2929.

Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., & Torralba, A. (2017). Scene parsing through ade20k dataset. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Zhou, B., Zhao, H., Puig, X., Xiao, T., Fidler, S., Barriuso, A., & Torralba, A. (2019). Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, *127*(3), 302–321.

Zhu, M., & Xie, Y. (2020). Taming unstructured sparsity on gpus via latency-aware optimization. *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 1–6. https://doi.org/10.1109/DAC18072.2020.9218644