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

Deep Joint Source-Channel Coding for Vision-Based Inference at the Wireless Edge

Mikolaj Jankowski

Supervised by Prof. Deniz Gündüz and Prof. Krystian Mikolajczyk

Submitted in part fulfillment of the requirements for the degree of PhD in Electrical and Electronic Engineering and the Diploma of Imperial College London

January 2024

Abstract

Vision-based inference systems have recently reached super-human levels thanks to significant progress in deep learning algorithms. However, application of these algorithms on edge devices is challenging due to their limited computational power and limited local information. Alternatively, edge inference systems can utilize the resources available at more capable edge servers to solve the underlying task. Yet, the design of distributed edge inference solutions is challenging, as it requires carefully considering and optimizing multiple factors related to deep learning together with wireless communications.

This thesis studies the design of vision-based edge systems for solving retrieval, classification, and deep neural network parameter delivery tasks under various constraints including the computational, memory, and communication resource limitations. The presented research is based on the recent advances in the field of deep joint source-channel coding (DeepJSCC), which is an alternative to classical, separation-based communication protocols. DeepJSCC simplifies the design of edge systems by introducing an autoencoder neural network, which is trained to map the information source directly to the channel input symbols, and similarly, to map the noisy channel output directly to the reconstructed signal. Such a DeepJSCC autoencoder pair can be further trained with a task-oriented optimization objective, leading to performance gains in the underlying computer vision tasks. For the tasks studied in this thesis, we provide a set of algorithms for achieving improved performance while meeting the communication and computational constraints. Extensive evaluations show that the proposed DeepJSCC approach is an exceptional alternative to the separation-based algorithms, and can play an important role in future generations of intelligent wireless networks.

Statement of Originality

I declare that the work included in this thesis is solely my own work other than where appropriately referenced.

Mikolaj Jankowski

Copyright Declaration

Acknowledgements

First and foremost, I would like to thank God. He has given me strength and guidance throughout all the challenging moments of my time at Imperial College. I am truly grateful for His unconditional and endless love, mercy, and grace.

I want to express my gratitude to my wife, Patrycja. She has always been there for me through all the hardships I have experienced in these last few years. She is the most loving and caring person I know, and her support has been irreplaceable.

I want to thank my parents - Bogusia and Krzysio, and my amazing siblings - Łukasz, Bartek, and Ola. They always believed in me, loved me, and supported me whenever they could. Having such a family is a genuine blessing, and I cannot express how happy I am to have them by my side. The same applies to my fantastic in-laws, whose constant encouragement and care has been truly invaluable.

If it was not for my supervisors - Deniz and Krystian, none of this thesis would ever have been written. They are amazing advisors who helped me shape every single piece of research included in this thesis. However, what I value most about them is that they served as great role models for an ambitious researcher who I have always dreamed of becoming.

Being a PhD student has been one of the most exciting experiences in my life so far. That would not be possible without being surrounded by a group of cool, like-minded fellow researchers: Tony - the best gym bud ever, Axel, Adrian, Dylan, Roy, Roy (yes, there are two), Sara, Benedikt, Haotian, Justin, Ecenaz, Nitish, Selim, Francesco, Yassine, David, Yulin, Mahdi, and Mohammad. Most importantly, I want to give special thanks to Michal, my long-time best friend, who inspired me to start this PhD in the first place and has always been ready to provide technical and emotional support, which I needed more often than I would like to admit.

I have always been privileged to have a few amazing friends, and they all deserve to be listed here. Przemek, Adam, Marek, Papi, Adrian, you are great buddies, thank you!

Once again, I cannot express how grateful I am to have so many people supporting me. I also apologize if I forget about someone. It seems like leaving acknowledgements until the last minute is not the best idea.

“All things are difficult before they are easy.”

Thomas Fuller

List of Acronyms

5G 5th-Generation technology standard for broadband cellular networks

AR Augmented Reality

AWGN Additive White Gaussian Noise

BN Batch Normalization

bpp bits per pixel

CNN Convolutional Neural Network

CPU Central Processing Unit

CSI Channel State Information

DL Deep Learning

DM NN Decision-Making Neural Network

DNN Deep Nerual Network

FLOP FLoating-point OPeration

GAN Generative Adversarial Network

GDN Generalized Divisive Normalization

GPU Graphics Processing Unit

IoT Internet of Things

JSCC Joint Source-Channel Coding

KD Knowledge Distillation

mAP mean Average Precision

ML Machine Learning

PMF Probability Mass Function

R-D curve Rate-Distortion curve

re-ID re-IDentification

SGD Stochastic Gradient Descent

SK mapping Shannon-Kotelnikov mapping

SNR Signal-to-Noise Ratio

UAV Unmanned Aerial Vehicle

UEP Unequal Error Protection

Contents

Abstract **3**

Acknowledgements **9**

List of Tables **19**

List of Figures **21**

1 Introduction **25**

1.1 Contributions 32

1.2 Outline 34

1.3 Publications 37

2 Background Theory **38**

2.1 Deep JSCC 38

2.2 Deep image compression 43

2.3 Network pruning 46

3 Joint Source-Channel Coding for Wireless Image Retrieval **50**

3.1 Introduction 51

3.2 Related work 54

3.2.1 Person and Vehicle Retrieval 54

3.3 Methods 54

3.3.1 Channel Model 55

3.3.2 Retrieval Baseline 56

3.3.3 Digital Transmission of Compressed Feature Vectors 56

3.3.4 JSCC of Feature Vectors 60

3.4 Results 62

3.4.1 Experimental Setup 62

3.4.2 Datasets 65

3.4.3 Performance for Different Methods 66

3.4.4 Performance for Different Bandwidths 68

3.4.5 Graceful Degradation 68

3.4.6 Training Strategy 70

3.4.7 Comparison of Different Models 72

3.5 Conclusions 73

4 Joint Device-Edge Inference over Wireless Links 74

4.1 Introduction 75

4.2 Methods 77

4.2.1 Channel model 78

4.2.2 Classification baseline 78

4.2.3 Autoencoder architecture 79

4.2.4 Training strategy 80

4.3 Results 81

4.3.1 Experimental setup 82

4.3.2 Channel bandwidth and on-device computation 83

4.3.3 Comparison between different pruning ratios 84

4.3.4 Comparison between different channel bandwidths 85

4.3.5 Selecting the optimal splitting point 85

4.4 Conclusions 85

5 Evaluation of Early Exits for Device-Edge Collaborative Inference over the Wireless Edge 87

5.1 Introduction 88

5.2 Methods 90

5.2.1 System model 90

5.2.2 Channel model 91

5.2.3 Training strategy 91

5.2.4 Transmission-decision neural network (TD NN) 92

5.3 Results 94

5.3.1 Experimental setup 94

5.3.2 Performance comparison 96

5.3.3 Ablation study 99

5.4 Conclusions 102

6 Neural Network Transmission over the Air 103

6.1 Introduction . . . 104

6.2 Related Work . . . 108

6.2.1 Network quantization . . . 108

6.3 System Model . . . 109

6.3.1 Channel model . . . 110

6.4 AirNet: JSCC of DNN Parameters for Reliable Edge Inference . . . 111

6.4.1 Training strategy . . . 111

6.4.2 Pruning . . . 112

6.4.3 Noise injection . . . 112

6.4.4 Knowledge distillation (KD) . . . 113

6.5 AirNet with Unequal Error Protection (UEP) . . . 113

6.5.1 UEP of DNN parameters against channel noise . . . 115

6.6 SNR Robustness . . . 117

6.7 Results . . . 119

6.7.1 Experimental setup . . . 119

6.7.2 Performance comparison . . . 121

6.7.3 Generalizability to different noise SNR . . . 125

6.7.4 Evaluation of training steps . . . 128

6.7.5 Comparison of bandwidth expansion methods . . . 129

6.7.6 Comparison of different sensitivity estimation criteria . . . 130

6.8 Conclusions . . . 131

7 Conclusion **132**

7.1 Future Work 136

7.2 Epilogue 137

Bibliography **137**

List of Tables

3.1 Comparison of the retrieval performance for different training strategies. 71

3.2 Person retrieval accuracy for the CUHK03 dataset achieved by different models at SNR = 0dB and $B = 64$. 71

5.1 Complexity of the TD NN compared to the other parts of the DNN. 101

List of Figures

1.1 Wireless edge inference system. 27

1.2 General communication system model. 31

2.1 A comparison between separation-based and JSCC systems. 39

2.2 A difference between structured and unstructured pruning techniques. 49

3.1 An illustration of the retrieval problem at the edge. 52

3.2 The deep learning based digital transmission scheme for image retrieval at the wireless edge. 56

3.3 The architecture and training of JSCC of feature vectors for wireless image retrieval. 59

3.4 Proposed JSCC encoder and JSCC decoder architecture for wireless image retrieval. 60

3.5 Performance comparison of the separation-based and JSCC schemes for wireless image retrieval. 63

3.6 Wireless image retrieval accuracy as a function of the channel SNR for different channel bandwidths. 66

3.7 Accuracy achieved by the JSCC AE scheme as a function of SNR$_{test}$ for different SNR$_{train}$ values. 69

3.8 Comparison of training strategies for wireless image retrieval at the wireless edge. 70

4.1 An overview of the proposed system for collaborative edge inference. 76

4.2 Proposed encoder and decoder architecture for feature transmission in collaborative edge inference systems. 79

4.3 Required channel bandwidth as a function of the number of on-device computations. 82

4.4 Classification accuracy as a function of the channel SNR for different pruning ratios and channel bandwidth fixed to $\boldsymbol{B}$. 83

4.5 Classification accuracy as a function of the channel SNR for different channel bandwidths. 84

5.1 Wireless collaborative inference system model with early exits available at the edge device. 88

5.2 The architecture of the transmission-decision neural network. 93

5.3 Comparison of the accuracy achieved by different models for comparable communication savings as a function of channel SNR. 96

5.4 Comparison of the accuracy achieved by different models for fixed channel SNR = 0 dB as a function of communication savings. 97

5.5 Average per-class confidences of the early exit predictions for the correct and incorrect predictions. 98

5.6 Comparison of the accuracy and savings of the methods based on early exit predictions confidence. 99

5.7 Comparison of the accuracy and communication savings of the Mixed criterion NN, given different inputs to the transmission-decision neural network. 100

6.1 System model for neural network parameters transmission over the wireless edge. 105

6.2 Examples of Archimedes' spirals used for the SK mapping scheme. 114

6.3 Deep neural network interpolation scheme. 118

6.4 Performance comparison between the proposed AirNet approaches, and the alternative digital and analog schemes on CIFAR10 dataset. 121

6.5 Performance comparison between the proposed AirNet approaches, and the alternative digital and analog schemes on Tiny ImageNet dataset. 123

6.6 Performance comparison of the models trained for different values of SNR$_{train}$ and tested on a range of SNR$_{test}$. 125

6.7 Comparison between vanilla AirNet trained with SNR$_{train}$ = SNR$_{test}$, interpolation scheme, and AirNet trained with variable SNR. $b \approx 0.65 \times 10^6$. 126

6.8 Ablation study for AirNet training strategy and various bandwidth expansion strategies. 127

6.9 Comparison between SK mapping and layer repetition scheme for bandwidth expansion (SNR$_{train}$ = 5dB, $b \approx 1.3 \times 10^6$). 128

6.10 Comparison of per-layer sensitivity measured by the largest eigenvalue λ_i of the Hessian matrix corresponding to a DNN layer parameters. 129

6.11 Comparison between UEP schemes with loss-based and Hessian-based sensitivity metrics (SNR$_{train}$ = 5dB, $b \approx 1.3 \times 10^6$). 131

Chapter 1

Introduction

The world as we know it would not exist without connectivity. According to the recently published McKinsey Technology Trends Outlook [1], advanced connectivity persists as one of the most important branches of technology. Billions of devices are connected nowadays, in fact, it is probably harder to stay disconnected from the global network than connected to it as over two-thirds of the global population has access to the Internet [2]. Once someone accesses the Internet, it becomes possible to exchange information with almost any place on Earth (and beyond). The global network expands rapidly and becomes an essential factor in improving the quality of our everyday lives. Another technology that has exhibited stable growth over the last few years [3] and is becoming increasingly vital to our quality of life is machine learning, which refers to machines and algorithms that are able to learn from examples they are provided with. Such machine learning algorithms are successfully used in many real-life applications, as they can be easily adapted to new scenarios by utilizing existing data samples and can provide tremendous performance improvements compared to old-fashioned handcrafted methods. Unfortunately, one of the main drawbacks of modern machine learning algorithms is that they require significant computational power to operate [4], which prevents most potential users from utilizing them. This problem becomes increasingly difficult to solve if we consider applying machine learning algorithms to visual data, that are known to be highly-dimensional and extremely complex. Designing algorithms that combine mobile connectivity and machine learning is one of the solutions to this problem. One can imagine a system, where a mobile phone user wants to recognize objects on a photo they captured, and utilizes a wireless network to upload the photo to a server, which then applies a modern machine learning algorithm to perform the desired task and transmits the answer back to the user [5]. Just a few years ago, that would have sounded like a scenario of a science-fiction movie, but the high demand for such algorithms is rapidly driven by the increasing number of applications, where such systems could bring significant advantages.

An example of a system, where wireless connectivity can be successfully utilized to support the execution of a machine learning algorithm is smart doorbells and home security cameras [6]. Such devices do not usually have access to the computational power required by the most demanding machine learning algorithms, yet can make use of the wireless network (usually WiFi) available in the majority of houses to connect to a remote server that can help in performing the task. Such tasks may include detection of humans within the proximity of the house, or even recognizing them, as well as their intent.

Another application for such machine learning systems executed at the wireless edge is performing various computer tasks with the data captured by unmanned aerial vehicles (UAVs) [7]. Due to limited payload and battery capacity, these devices cannot have powerful computing engines available onboard, thus offloading the collected data to a server located at the network edge is often the only possibility to enable real-time analysis of the data with advanced machine learning algorithms.

Finally, we can consider various applications in the area of smart cars and autonomous driving. Modern autonomous cars have access to a significant amount of computational power necessary to process the data streams produced by all the sensors they are equipped with, including cameras, LiDAR, or proximity meters. However, their capabilities can be enriched by the connectivity [8] to other nearby vehicles, or cameras monitoring the traffic at surrounding junctions for even more optimal route planning and increased safety. For such purposes, systems have to be designed by considering the coexistence of wireless connectivity and advanced machine learning.

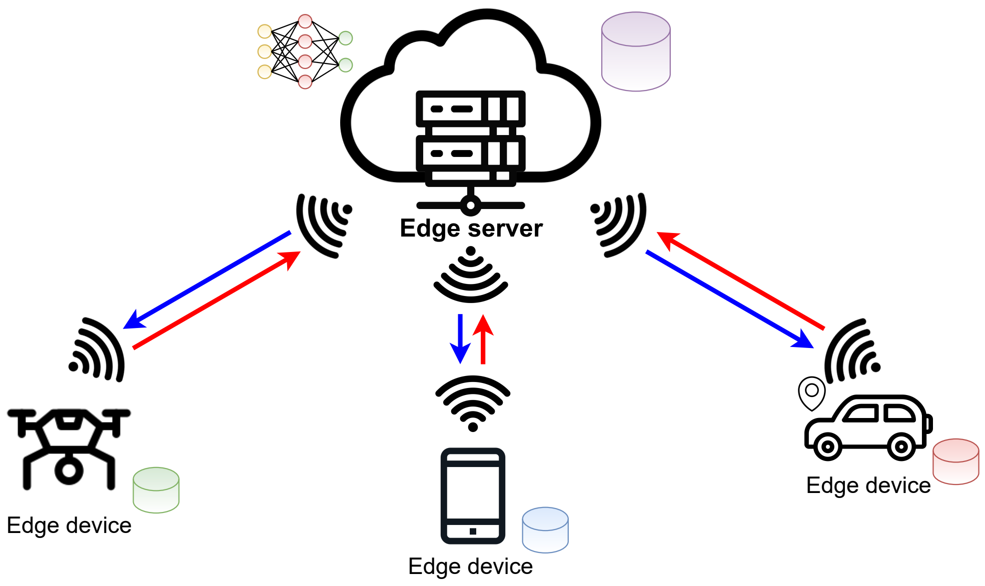


Figure 1.1: Wireless edge inference system.

This thesis focuses on finding both computationally and communication efficient ways to perform machine learning at the wireless edge [9, 10] to solve advanced machine vision tasks, where an *edge device* cooperates with the more capable *edge server* in order to complete the task, as shown in Fig. 1.1. The edge device can be any mobile terminal with limited computational resources, while the edge server can be a base station or a WiFi access point. While the edge device wants to exploit the available computational resources of the edge server, it has to overcome communication constraints due to the noisy channel connecting the two.

Brief history of cellular networks. Historically, all the networks used to rely on wired connection between two points. Even a simple telephone call required first contacting the telephone exchange center, specifying the recipient of the call, and being physically routed by the operator, who had to activate the appropriate switches on the switchboard. Over the years, the rise of wireless communications made it possible to introduce mobile devices, which did not require direct cable connection to operate, but utilized electromagnetic waves to convey the signal. Firstly introduced in the 1960s and 1970s, personal wireless communications systems were originally used only for voice calls [11], and very quickly adapted to other applications, including text messages, and access to the broadband Internet.

One of the crucial moments that contributed to the rapid development of wireless connectivity was the introduction of cellular networks [12]. The concept of cellular networks includes dividing the coverage areas into hexagonal cells, each containing a cell tower providing connectivity for users within that individual cell. Each generation of cellular network technology introduced various advances over the previous one. The first generation (1G) cellular networks were based on analog communications, where uncoded analog voice signal was transmitted by a high-frequency carrier directly to the cell tower. This standard only included voice calls, and had several significant issues by modern standards. The following generation (2G) of cellular systems replaced analog transmission with digital alternatives. Instead of modulating the carrier with the low-frequency voice signal, the voice was sampled and quantized into a bitstream, which was modulated with a high-frequency carrier. Introducing digital communications further allowed for the transmission of text messages. The following, third (3G) and fourth (4G) generations of cellular networks introduced wireless access to the Internet from mobile devices, while also providing the technologies necessary to accommodate the rapidly growing number of users. With the rise of 5G – the fifth generation technology standard for cellular broadband networks, we can reliably access the global network from our personal mobile devices, while achieving download speeds that allow for significantly more than voice or text messages. This includes video calls, video streaming, transfer of large files, and more [13]. We quickly got used to having access to all those resources from the devices that most of us carry in our pockets. Yet, it is worth looking back at the technical developments of the last 75 years that contributed towards creating the technology that we cannot imagine living without.

Artificial intelligence and machine learning for 6G. Artificial intelligence is often referred to as one of the key technologies for achieving increased connectivity necessary for the future, 6G standard for cellular networks. Artificial intelligence is expected to provide even better data rates, lower latency, and improved efficiency in 6G use cases. Some of the applications include smart cities, augmented reality, telemedicine, and holographic communications [14]. Another area, where artificial intelligence can bring significant improvements, is data compression. Machine learning algorithms can model the underlying nature of the data source to be compressed, and achieve significant advantages over the existing, handcrafted schemes. This is extremely important, as we rely on the Internet more and more, with Internet traffic expected to grow by an order of magnitude within the next 10 years [15].

The rise of deep learning. Deep learning is a branch of machine learning, that focuses on trainable models called deep neural networks. These models are composed of multiple processing layers, which are capable of learning representations of data at different levels of abstraction [16]. The origins of deep learning can be traced back to 1943 when the first computer inspired by the human brain was invented [17]. Following this event, researchers across the globe started to look into deep learning and new essential technologies were proposed. These included backpropagation algorithm [18], and convolutional neural networks [19], which finally led to the invention of the first deep learning based digit classifier in 1989 [20]. Since then, deep learning started to grow in popularity, however, because of multiple factors, it was extremely hard to deliver this popularity from the research labs to a wider audience. One of the reasons was the computational complexity of deep learning algorithms. With the less powerful hardware that existed in the 1990s and early 2000s, it was almost impossible for a private user to run deep learning algorithms on their personal device. Another problem was the availability of the data. Each deep neural network requires thousands of examples for the learning purpose to be able to generalize well to unseen data. Finally, in the early years of deep learning, the performance achievable with deep neural networks was far from what we can achieve nowadays. In fact, classical machine learning algorithms provided much better performance at lower complexity. A combination of multiple breakthroughs eventually allowed deep learning to leave the research institutes and slowly be applied in real-life scenarios. Some of the most crucial advances that improved the applicability of deep learning included the introduction of GPUs (Graphics Processing Units), which are computing devices capable of parallelizing thousands of simple arithmetic operations and achieving significant speedups compared to CPUs (Central Processing Units). This ability to run operations in parallel is well-suited for deep learning, as executing each layer of a deep neural network requires performing a massive number of independent simple computations. Alongside the introduction of the GPUs, other advances were made as well, including the development of better optimizers [21, 22], access to large, publicly-available datasets [23, 24, 25], and the introduction of deep learning models [26, 27] that were able to compete against other machine learning algorithms, and ultimately outperform them in a variety of different tasks. Having been available only to researchers, deep learning is now a technology, that everyone can take advantage of. Developers and scientists across the world can easily access and actively contribute towards creating new deep learning methods, through open-source projects like NVIDIA CUDA Toolbox, PyTorch [28] and TensorFlow [29] libraries. This means that almost any software company or individual in the world can develop products built upon deep learning and provide them to users who can effortlessly access them through their personal devices.

Edge inference systems. Improved latency and extreme speeds introduced within the 5G standard enabled *edge devices* (smartphones, autonomous cars, IoT devices, etc.) to efficiently communicate with powerful computing devices deployed at the wireless edge, denoted as *edge servers*. Such connectivity will enable the edge devices to utilize computing resources available at the edge servers to make use of the most advanced artificial intelligence algorithms for their own purposes, effectively creating an *edge inference* system. In the next years, we are likely to see mobile phones capable of translating between arbitrary languages in real-time, providing accurate localization services based on images captured by their built-in cameras, and immersing their users into high-quality AR (augmented reality) experiences. The network does not, however, consist only of mobile phones. The increasing number of other devices connected to mobile networks led to defining a new term – Internet of Things (IoT). IoT includes all sorts of devices equipped with sensors, and processing power, that are connected to the Internet. Those devices are expected to be an essential part of many edge inference systems. Some potential uses of edge inference systems in the IoT context may include CCTV cameras, which automatically recognize suspects, autonomous cars able to communicate with each other to ensure improved safety and traffic control, or mobile wireless access points mounted on drones to provide improved service quality to users. Such edge inference systems have not existed in the past, so the research community has become increasingly active in designing and improving

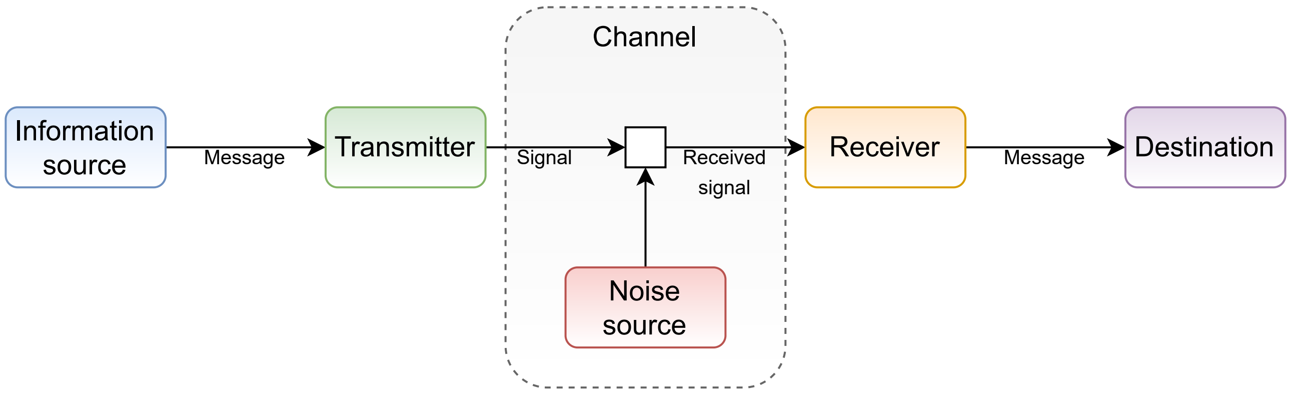


Figure 1.2: General communication system model.

the efficiency of such systems. From the engineering perspective, the data sources related to edge inference systems are distinct from the classical data sources like images, text, or video, for which general-purpose codecs and transmission protocols already exist. Designing new compression standards for these new, edge inference-based sources requires in-depth analysis and treating each system on a case-by-case basis.

Technical, semantic, and effective communications. The first general communication model was proposed in 1948 and contains five basic components: an information source, a transmitter, a channel, a receiver, and a destination [30]. This system, shown in Fig. 1.2 is still commonly used nowadays by researchers worldwide. The information source is responsible for producing a message or a sequence to be communicated to the receiver, the transmitter translates the information to a signal suitable for a communication medium, the channel is a medium used to convey the signal, the receiver performs an inverse operation to that of the transmitter, and, finally, the destination is the intended recipient of the message. Within this model, three levels of communications are considered [31], namely level A – technical, level B – semantic, and level C – effective. The technical level focuses on transmitting communication symbols accurately from the sender to the receiver, the semantic level is concerned with conveying the desired meaning of the information, and the effective level revolves around ensuring that the received meaning affects conduct in the desired way. Most of the works in the field of communications focus on the technical level since applying theoretical results to the other two levels is significantly more complex. Over the last years, however, semantic communications received a lot of attention, thanks to the efforts in developing semantics-related measures and theory, alongside the application of various machine learning and deep learning algorithms to semantic communications tasks [32]. These results are widely applicable to various tasks in the field of over-the-edge machine and deep learning, where the primary goal is to achieve satisfactory task-related performance, rather than good reconstruction quality usually pursued in the technical level communications.

Joint source-channel coding. According to Shannon's separation theorem [30], if we assume infinite blocks of data to be transmitted over a channel, the optimal approach is to perform source coding and channel coding steps separately. Source coding is responsible for removing the redundancy within the data source to achieve compression, whereas channel coding re-introduces structured redundancy in order to better protect it from channel imperfections. Most of the communication systems have been designed according to this paradigm since it ensures certain theoretical guarantees. However, in practice, due to the complexity and latency constraints, the information is transmitted through finite blocks, thus transmission of many sources has been recently shown to benefit from joint source-channel coding [33, 34], with deep neural networks implemented as joint encoders and decoders that map the information sources directly to channel input symbols and back. We note, that such a design is suitable for edge inference problems, where the optimal joint encoder-decoder pairs can be parameterized by deep neural networks and trained to adapt to various sources originating from split/edge inference. An additional benefit of utilizing such models is that they can often be trained in an end-to-end fashion since both inference and communications can be implemented as deep neural networks with a non-trainable channel model.

1.1 Contributions

The overall contributions of this thesis can be summarized as follows:

- We present an extensive summary of recent advances in the fields relevant to performing machine learning over the wireless edge for vision-based inference. These include efficient DNN design, joint source-channel coding, collaborative/split inference, etc.

- We study a problem of vision-based edge inference, where an edge device needs to exchange information with an edge server, in order to complete a machine learning task with deep neural networks trained to process image data. We consider various constraints that may affect the problem, including latency, bandwidth, computational complexity, etc.

- We tackle different problems within the edge inference framework by proposing methods based on joint source-channel coding and deep learning.

- In Chapter 3 we study the problem of wireless image retrieval, where the edge device transmits a feature representation of an image to an edge server, which has access to a large database. We explore and provide principled guidelines for designing such systems, and validate them thoroughly.

- In Chapter 4 we explore the problem of split inference, where the edge device is capable of running only a part of a deep neural network, due to computational power limitations, and has to transmit the intermediate feature representation to the edge server to complete the forward pass of the DNN and produce the final prediction. Through an extensive evaluation of the proposed methods we show, that JSCC can outperform separation-based approaches in a variety of scenarios.

- We extend the approach presented in Chapter 4 to a scenario, where the edge device is equipped with an early exit mechanism that can provide a less confident prediction locally, without the need to utilize channel resources. In order to decide, whether transmitting intermediate features would be beneficial or not, a decision-making mechanism is introduced. The mechanism analyzes the certainty of the early exit and current channel conditions to output the final decision. In Chapter 5, we analyze various decision-making mechanisms with respect to the transmission savings and the overall task accuracy they can achieve.

- In Chapter 6, we propose a method for transmitting parameters of deep neural networks, trained for image-based inference tasks, over noisy channels. This method incorporates recent advances in the deep learning field, alongside JSCC to ensure reliable transmission, while also meeting stringent bandwidth and latency constraints.

1.2 Outline

Performing vision-based machine learning inference over the wireless edge is a relatively new discipline. Early machine learning algorithms were aimed at running on a single machine equipped with sufficient computational power. However, with the recent developments of wireless mobile networks and the improved quality of machine learning algorithms, it is possible to deliver machine learning capabilities directly to the users through the wireless edge. This introduces a multitude of challenges that have to be properly tackled by researchers around the world. One of the most important problems to solve is how to ensure reasonable latency, such that the mobile user can benefit from using machine learning algorithms in real time. Another challenge is computational power, which is one of the most significant limitations in the current state of machine learning approaches aimed at mobile usage. Different ways of tackling these challenges, including efficient design of neural networks, improving the quality of source compression, and incorporating a communication medium into the end-to-end trainable systems, have been summarized in Chapter 2. The works presented in subsequent chapters tackle different problems of vision-based machine learning inference over the wireless edge. The majority of the works build upon recent discoveries in the field of deep learning, computer vision, and communications, particularly focusing on deep joint source-channel coding and semantic communications fields. The summary of each of the works and their corresponding contributions is presented below.

In **Chapter 3: Wireless Image Retrieval at the Edge** [35, 36], we study the image retrieval problem at the wireless edge, where an edge device captures an image, which is then used to retrieve similar images from an edge server. These can be images of the same person or a vehicle taken from other cameras at different times and locations. Our goal is to maximize the accuracy of the retrieval task under power and bandwidth constraints over the wireless link. Due to the stringent delay constraint of the underlying application, sending the whole image at a sufficient quality is not possible. This chapter proposes two alternative schemes based on digital and analog communications, respectively. In the digital approach, we first propose a deep neural network (DNN) aided retrieval-oriented image compression scheme, whose output bit sequence is transmitted over the channel using conventional channel codes. In the analog joint source and channel coding (JSCC) approach, the feature vectors are directly mapped into channel symbols. We evaluate both schemes on image-based re-identification (re-ID) tasks under different channel conditions, including both static and fading channels. We show that the JSCC scheme significantly increases the end-to-end accuracy, speeds up the encoding process, and provides graceful degradation with channel conditions. The proposed architecture is evaluated through extensive simulations on different datasets and channel conditions, as well as through ablation studies. The relevant background theory for this chapter is covered in detail in Section 2.1 and 2.2 of Chapter 2, as well as in Section 3.2 of Chapter 3.

In **Chapter 4: Joint Device-Edge Inference Over Wireless Links** [37], we propose a joint feature compression and transmission scheme for efficient inference at the wireless network edge. We note, that mobile devices may struggle to efficiently run the entire forward pass of modern DNNs, given their limited computational power. This poses a significant limitation to the applicability of the algorithms presented in Chapter 3. Therefore, the goal of Chapter 4 is to enable efficient and reliable inference at the edge server assuming limited computational resources at the edge device. Previous works focused mainly on feature compression, ignoring the computational cost of channel coding. The proposed approach incorporates a deep joint source-channel coding scheme and combines it with novel pruning strategies aimed at reducing the redundant complexity of neural networks. We evaluate our approach on a classification task and show improved results in both end-to-end reliability and workload reduction at the edge device. The background theory for this chapter has been provided in Section 2.1 and 2.3 of Chapter 2.

The idea introduced in Chapter 4 is further extended in **Chapter 5: Evaluation of Early Exits for Device-Edge Collaborative Inference over the Wireless Edge**, where we consider similar joint device-edge inference systems with a particular focus on early exits, which allow for obtaining inference results at the edge device. This approach eliminates the need to transmit the partially processed data to the edge server, leading to further communication savings. The central part of the system is the decision-making (DM) mechanism, which, given the information from the early exit, and the wireless channel conditions, decides whether to keep the early exit prediction or transmit the data to the edge server for further processing. In this chapter, we evaluate various DM mechanisms and show experimentally, that for image classification tasks over the wireless edge, proper utilization of early exits can provide both performance gains and significant communication savings. The relevant background theory for this chapter is covered in Section 2.1 of Chapter 2 and Section 5.1 of Chapter 5.

Finally, we consider a scenario, where the edge device elects not to share its data with the edge server, as in the previous chapters, but requests a DNN architecture and its corresponding parameters to complete the vision-based inference task itself. Therefore, in **Chapter 6: Neural Network Transmission over the Air** [38], we present AirNet, a novel training and analog transmission method to deliver DNNs over the air. Often, the employed DNNs are location- and time-dependent, and the parameters of a specific DNN must be delivered from an edge server to the edge device rapidly and efficiently to carry out time-sensitive inference tasks. This can be considered as a joint source-channel coding (JSCC) problem, in which the goal is not to recover the DNN coefficients with minimal distortion, but in a manner that provides the highest accuracy in the downstream task. In the AirNet method, we first train the DNN with noise injection to counter the wireless channel noise. We also employ pruning to reduce the number of parameters and reduce the channel bandwidth necessary for the transmission. We further identify the most significant DNN parameters by calculating the largest eigenvalues of the Hessian matrix with respect to each of the layers. Based on these values, we introduce a non-linear bandwidth expansion to provide better error protection for the most noise-sensitive layers of the network. We further propose a DNN interpolation scheme, where DNN parameters are calculated as a weighted sum of parameters of two networks trained with two extreme (low and high) values of channel SNR, and show that the interpolated network can achieve satisfactory results when tested on intermediate values of SNR. This indicates that, with our approach, it is not necessary to store multiple sets of DNN parameters trained for a single value of SNR. We also show that AirNet achieves significantly higher test accuracy compared to the separation-based alternative, and exhibits graceful degradation with channel quality. The background theory for this chapter has been provided in Section 2.1 and 2.3 of Chapter 2 and Section 6.2 of Chapter 6.

1.3 Publications

The main results presented in this thesis correspond to the following publications:

- M. Jankowski, D. Gündüz, K. Mikolajczyk, "Adaptive Early Exiting for Collaborative Inference over Noisy Wireless Channels", *IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, 2024.
- M. Jankowski, D. Gündüz, K. Mikolajczyk, "AirNet: Neural Network Transmission over the Air", *arXiv:2105.11166*, 2023.
- M. Jankowski, D. Gündüz, K. Mikolajczyk, "AirNet: Neural Network Transmission over the Air", *IEEE International Symposium on Information Theory (ISIT)*, 2022.
- M. Jankowski, D. Gündüz, K. Mikolajczyk, "Joint Device-Edge Inference over Wireless Links with Pruning", *IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2020.
- M. Jankowski, D. Gündüz, K. Mikolajczyk, "Wireless Image Retrieval at the Edge", *IEEE Journal on Selected Areas in Communications (JSAC)*, 2020.
- M. Jankowski, D. Gündüz, K. Mikolajczyk, "Deep Joint Source-Channel Coding for Wireless Image Retrieval", *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2020.

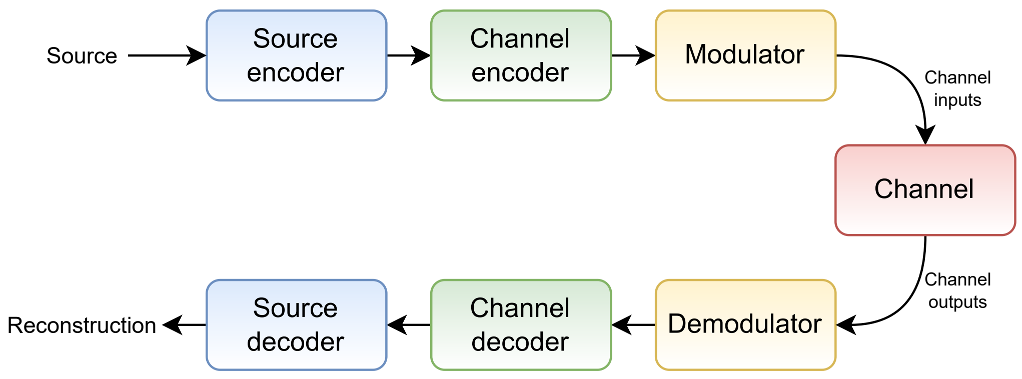
Chapter 2

Background Theory

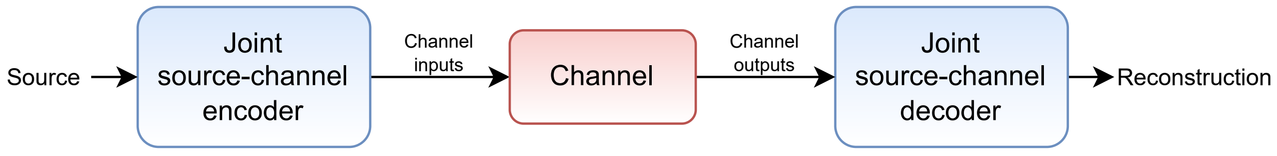
In this chapter, we will present relevant works in the literature related to vision-based inference at the wireless edge. This area is extremely multifaceted, as it combines solutions to problems related to the complexity of running machine learning and computer vision algorithms, and proper design of wireless communications systems.

2.1 Deep JSCC

Shannon's separation theorem. Shannon's source-channel separation theorem [39] states that if the minimum source coding rate is below the capacity of the channel, this source can be reliably transmitted over the channel with the use of proper encoding and decoding operations. In the opposite scenario, that is, when the minimum source coding rate is above the capacity of the channel, transmission of that source is impossible. Another part of this theorem states, that the above findings still hold, even when the source and channel codes are designed independently. These two findings lead to the conclusion that separating source and channel coding is the optimal approach; however, this theorem operates under the assumptions of the infinite source and channel bandwidths, ergodic source and channel distributions, and for an additive distortion measure in general. These assumptions are not realistic in practice. One of the reasons for that is that wireless traffic is usually formed into packets of finite length, which



(a) Separation-based system



(b) JSCC-based system

Figure 2.1: A comparison between separation-based and JSCC systems. JSCC introduces a direct mapping from a source domain to channel input symbols and reverses this mapping at the receiver.

already violates the first assumption. Despite not being fully applicable in practice, Shannon's source-channel separation theory has been commonly utilized to design communication systems as modular, layer-based structures.

Separation-based systems. Classical separation-based approaches first perform source coding steps to remove the redundancy within the source and effectively compress it for more efficient transmission. This step is usually followed by the channel coding step, where, depending on channel conditions, structured redundancy is re-introduced to ensure the information can be delivered without errors (see Fig. 2.1a). Various codes have been developed for channel coding, including LDPC [40], Turbo [41] or Polar [42] codes. The main working principle of these codes is to introduce additional bits into the information bit-stream, that allow for the detection of errors within the received signal and further correct them. This channel coding step is crucial for the performance of the digital communication systems, as the source would be impossible to decode under the presence of even a single bit-error within the bit-stream. Moreover, classical separation-based systems are extremely reliant on accurate channel estimation, since transmitting at rates above the channel capacity is likely to produce bit-errors and lead to *outage*. Therefore, as the channel conditions get worse for a fixed transmission rate, we are likely to observe a *cliff effect*, that is, a sharp drop in the reconstruction quality of the source, once the channel SNR values get below a certain threshold.

Classical JSCC. Instead of performing source and channel coding separately, joint source-channel coding systems apply a direct mapping from a source domain to channel input symbols. After the transmission, received symbols are mapped back to the source domain with a reversed mapping to restore the information, as shown in Fig. 2.1b. One of the most important features of JSCC-based systems is that they avoid the cliff effect, characteristic of the separation-based methods, and exhibit *graceful degradation* instead. With the graceful degradation, the reconstruction quality slowly decreases as the channel conditions get worse. This behavior is extremely useful in the presence of channel estimation uncertainties. In the last 30 years, many systems have been shown to benefit from applying JSCC for various information sources and wireless channel models [43], [44], [45], [46].

Typical approaches for jointly designing source and channel codes for wireless image transmission rely on a rate-distortion (R-D) curve obtained for the scheme. The R-D curve characterizes the trade-off between the expected quality of the image delivered and the minimum transmission rate necessary to achieve that quality. Using a separation-based scheme might seem more convenient in such a scenario since it assumes only the source code is chosen based on the R-D curve, while the channel code is designed such that it guarantees reliable transmission of the compressed source bits. However, in certain scenarios, it has been shown that the joint optimization of the source and channel codes might lead to improved quality of the source reconstruction [47]. The underlying problem in designing classical JSCC algorithms is to find a proper balance between the quantization error introduced by source coding, and the error imposed by the channel noise, which occurs due to the imperfect reliability of the channel code. To this end, an optimization procedure has to be designed, taking into account the characteristics of the source coder, i.e., the R-D operating points, and the characteristics of the channel coder, i.e., its collection of error-correcting strengths and their corresponding code rates [48]. A seminal work presented in [49] describes a methodology for evaluating the joint R-D behavior of the JSCC schemes under the assumption of the fixed bandwidth. The authors further estimate information-theoretic bounds on the performance of such systems. The authors of [50] proposed a JSCC scheme for time-varying channels by considering multiresolution modulation constellations. The proposed approach can further adapt to versatile situations of CSI available at both transmitter and receiver or receiver only. An algorithm for progressive image transmission has been proposed in [51]. In progressive approaches, intermediate rates are optimized alongside target rates to achieve the maximum average performance in the presence of channel noise, which further improves the robustness of the scheme in case of possible outages. This scheme further reformulates the optimization task to maximize the number of correctly decoded packets instead of minimizing the overall distortion, which was shown to significantly reduce the computational complexity of solving the problem. Despite providing certain advantages over separation-based methods, JSCC methods presented in this paragraph still may suffer from the high computational complexity required to find the correct setup for source and channel codes. Moreover, they cannot be easily adapted to versatile sources or channel models, thus their practical application might be limited.

Deep JSCC. In order to overcome some of the limitations of the classical JSCC approaches, DNN-based methods have been proposed for efficient transmission in the JSCC regime [33, 52, 53, 54, 55, 34, 56, 57, 38, 58]. Deep learning based JSCC methods for wireless image transmission usually introduce autoencoder neural networks to learn a mapping from the source to channel input symbols, together with the reverse mapping back to the source domain. These neural networks are usually end-to-end trainable with a non-trainable channel model embedded into the autoencoder's structure, between the encoder and the decoder. The approach proposed in [33] proposes applying JSCC autoencoder for wireless image transmission. The authors directly map input images' pixel values to real or complex-valued channel symbols and show that JSCC outperforms standard separation-based approaches, where the source coding is performed with standard compressive codecs (JPEG, JPEG2000), even under the capacity-achieving channel codes assumption. This approach has been further extended by considering receiver feedback [52], where the channel output information is utilized at the transmitter and guides it through the subsequent transmission to improve the quality of the reconstruction at the receiver. Successive retransmission scheme is proposed in [59], where the authors consider the progressive transmission of images in multiple stages, with and without the presence of feedback from the receiver. Work proposed in [53] studies JSCC systems for image transmission, which can adapt their bandwidth requirements under constantly changing channel conditions. Other works in the literature apply similar techniques to image domain [54, 55], text domain [56, 57, 60], video domain [34], speech domain [61], DNN parameter transmission [38], and industrial sensory data [58].

DeepJSCC for distributed inference. Deep JSCC has also been applied to distributed inference [62] problems [63, 37, 35, 64, 65, 66, 67, 68, 69]. Similarly to standard deep JSCC methods for transmitting classical sources, those applied to distributed inference usually consist of a trainable autoencoder, which learns to compress and reconstruct the intermediate features produced by a DNN, under the presence of a channel model between the encoder and the decoder. The main difference between classical, distortion-based JSCC approaches including image, video, text or speech transmission, and distributed inference approaches, is that the latter do not follow the standard reconstruction-based performance metrics. Instead, the transmitted features have to contain enough information to perform successful inference under the task-defined performance metrics, even under noisy channel conditions. This formulation is inherently more difficult to tackle within the deep JSCC general framework because simply reconstructing the DNN intermediate features with low distortion may lead to suboptimal results, thus task-oriented metrics have to be introduced to training algorithms.

Some examples of such distributed inference JSCC systems can be found in [63], where the authors perform compression of the intermediate feature layers produced by a DNN trained for the classification task. A similar approach is proposed in [37] with the addition of pruning at the transmitter to minimize the complexity of the algorithm for efficient execution at power-constrained devices. The method proposed in [66] studies the progressive transmission of the intermediate features until the desired confidence is achieved in the classification task. Approaches designed for features originating from multiple sources or users are proposed in [64], with additional privacy requirements considered in [65]. Multi-task JSCC-based distributed inference has been studied in [67, 68]. In these approaches, the information extracted from the source has to serve multiple tasks at the receiver, including detection, segmentation, classification, visual question answering, etc.

2.2 Deep image compression

Classical image compression algorithms. Image compression remains one of the most researched areas within communications, information theory, and vision fields nowadays. Transform-coding-based handcrafted methods have been commonly used in recent years. Popular JPEG format [70], based on discrete cosine transform [71] is still the most popular codec because of its versatility and low computational complexity. Nevertheless, there exist formats that are significantly better in terms of image quality and compression rate. These include JPEG2000 [72] or BPG (Better Portable Graphics) [73] or WebP [74].

Deep image compression. Despite the undeniable success of handcrafted methods, we observe a growing interest in the field of DNN-based image compression. Recent designs are able to jointly optimize compression rate and distortion between the input image and its reconstruction [75] [76] [77]. The general working principle of such learned schemes is similar. Firstly, an image is transformed by a convolutional encoder into a lower-dimensional latent representation, in which the probability density function is modeled with various learning-based methods. [75] uses a piecewise linear density model to estimate the prior distribution of the latent variable. Authors of [77] replace this density model with a more flexible nonlinear model of the cumulative distribution of the latent, conditioned on a hyperprior. An alternative method [76] proposes modeling the priors as a mixture of Gaussians. Such mixtures are suitable for modeling the distribution of the latent, thanks to the generalized divisive normalization [78] layer, which imposes Gaussian-like shapes of the distributions of the transformed inputs. The approach proposed in [79] introduces autoregressive and hierarchical priors to better exploit the internal dependencies within the latent vector, and guide the entropy model for better compression. To model the quantization process, quantization noise [80] is commonly employed to ensure the differentiability of the resulting transform, which enables end-to-end training of the whole compressive autoencoder. Alternative differentiable quantization approximation was proposed in [76], where the authors explicitly set the derivative of the operation to 1, and simply round each element of the low-dimensional latent representation to the closest integer in both training and inference. After being encoded and quantized, the latent vector is mapped back to the original image space by the decoder, which usually mirrors the structure of the encoder. During inference, low-entropy latent representation is losslessly compressed, usually with the use of various arithmetic coding schemes [81], based on the approximated distribution model learned by the DNN. Such approaches have been extremely successful in recent years, yielding similar results or even surpassing the reconstruction quality of the handcrafted methods at higher compression rates.

Perception-based optimization of deep image compression. Despite achieving state-of-the-art performance in classical performance metrics, including PSNR or MS-SSIM, deep learning based compression algorithms have been also successfully applied for optimizing perception-based performance metrics. When reconstructing an image, one can explicitly optimize a DNN-based autoencoder to yield high-quality reconstructions, but further improvements can be achieved in the perceptual domains if classical distortion metrics are combined with perceptual quality counterparts. This idea has been studied in various works that combined compressive autoencoders with generative adversarial networks (GANs) [82]. A group of the proposed approaches [83] [84] aim at improving the perceptual quality of reconstructed images and overcoming the problem of blurriness that usually occurs at higher compression rates. The working principle of GAN-based compressive autoencoders is based on the adversarial loss term incorporated in the loss function.

Lossless deep image compression. The field of lossless image compression is highly dominated by classical codecs, including lossless variants of JPEG, JPEG2000, BPG, and WebP, which are usually used as lossy codecs, as well as formats that provide lossless compression only, such as PNG [85] or FLIF [86]. Since the working principle of lossless codecs is to derive a probabilistic model of an input image, which is later utilized to perform efficient arithmetic coding, it was noticed in [87] that generative models that learn probability distributions over pixels can be in theory used for lossless image compression. The design of neural-based methods for lossless compression has been studied in [88]. The approach presented in that work consists of building a hierarchical probabilistic model for estimating the pixel distribution within the image. This model is used to guide the process of arithmetic coding of the pixels for lossless compression. A similar approach is further exploited in [89] to compress an image that is the difference between an input image and its BPG-compressed reconstruction. Such an approach was shown to achieve better compression rates than the lossless version of BPG.

Task-oriented image compression. Despite the good quality of the reconstructed images, both adversarial and standard compressive autoencoders fail to guarantee satisfactory results in task-specific scenarios, where the performance of an algorithm performing the task (e.g., image classification or retrieval) has to be preserved.

The necessity to develop task-oriented compression schemes has already been noticed by researchers, and variations of transform-based compression algorithms have been proposed for certain tasks. In [90], focusing on magnetic resonance tomography, authors propose a wavelet-based lossy compression technique designed to minimize the distortion between automatically generated segmentation maps based on original and recovered images. Authors of [91] introduce a metric based on conditional class entropy, which is incorporated into the JPEG2000 encoder. This method includes replacing the standard mean squared error (MSE) term during the rate allocation step with a task-specific conditional entropy and performing encoding with respect to such allocation, rather than the MSE, which prioritizes image reconstruction quality. This approach leads to a significant decrease in bits-per-pixel (bpp) and increases the success rate in detection and classification tasks at the same time. To provide high task accuracy and better image quality, a mixed approach is proposed in [92], where a convex combination of both objectives is used.

Authors of [5] have shown that latent representation produced by compressive autoencoders can be used to perform the classification task with ResNet-50 [93] network and lead to achieving almost the same accuracy obtained by training on the uncompressed image. This finding leads to the conclusion that the classification network does not need to reconstruct the image, as long as sufficient information is preserved within the compressed representation of that image. A similar approach is presented in [94] for the task of face recognition. More recently, a learning-based method proposed in [95] allowed for compressing features generated by DNNs by jointly optimizing the compressibility of the features, alongside a task-related objective function.

Separation-based compression techniques presented above were shown to produce visually pleasing, high-quality reconstructions of images. Nevertheless, when considering the image transmission problem, or task-oriented image transmission problem, digital schemes require very accurate channel state information to ensure reliable transmission, as they exhibit the *cliff effect* if the channel quality falls below a certain threshold imposed by the desired transmission rate. Therefore, joint source-channel coding has been extensively studied in recent years, constituting a reliable compression and transmission alternative to separation-based schemes. Such joint source-channel coding based approaches are discussed in detail in Section 2.1.

2.3 Network pruning

Network pruning is a set of techniques aimed at reducing the computational or memory complexity of DNNs. State-of-the-art DNNs usually require the storage of millions of floating-point parameters, and their computational complexity reaches billions and trillions of floating-point operations (FLOPs). Unfortunately, even modern computers often struggle to perform such amount of computations quickly, which drastically increases the latency of providing the predictions for the task. Moreover, utilizing deep learning algorithms on mobile devices of even lower computational capabilities would be beneficial for the users, yet extremely hard to achieve. This motivates the researchers to seek techniques that allow them to make the DNNs smaller and less computationally demanding.

Pruning aims at finding the redundant connections within deep neural networks. It has been shown [96], that deep neural networks are usually overparameterized, which means that not all of the parameters are necessary to provide satisfactory performance. In fact, removing some of the DNN parameters can often improve the performance of the network. These redundant parameters are usually found by calculating their saliency/importance. The exact definitions of the saliency measures can vary between different methods, yet the goal is the same - to find DNN parameters that will not lead to performance degradation once removed. Different examples of saliency measures include:

- L1-norm of the DNN parameters [97],
- The increase in the loss function induced by removing the parameter, approximated by first-order Taylor expansion [98],
- The increase in the loss function induced by removing the parameter, approximated by second-order Taylor expansion [99],
- LASSO regression [100],
- Average magnitude of the gradient w.r.t. the connectivity parameter of each connection within a DNN [101],
- Rank of the feature maps produced by the parameter [102],
- Magnitude of the feature maps produced by the next layer of the network [103].

Iterative pruning. Pruning algorithms are usually applied iteratively, with fine-tuning steps performed between individual pruning steps. The main motivation for applying fine-tuning is to regain the accuracy lost due to pruning. After every pruning iteration, certain connections are removed from the network, and the network requires further training to learn how to better utilize the remaining connections. An alternative approach would be to find a smaller DNN before the start of the training and train this smaller network from scratch, to avoid unnecessary training time required by the iterative pruning process. Findings of [104] seem to confirm that repetitive pruning and fine-tuning is a proper approach since finding smaller DNNs and training them from scratch usually leads to sub-optimal performance. The authors first prune a DNN, and, re-initialize its weights after pruning to show that this small DNN structure is not universal, and will not converge. However, when one reuses the same initialization as the one applied to the full DNN before pruning, a smaller sub-network can be successfully trained, despite a significantly lower number of parameters, and achieve the same performance as when pruned iteratively. This leads to the conclusion, that specific sub-networks within a larger network can effectively learn only under very specific initialization of parameters, which is extremely difficult to find. A different approach is proposed in [101], which attempts to find smaller sub-networks before training, by carefully analyzing connection sensitivities for the entire DNN, and calculating the gradient of the loss function w.r.t. the connectivity parameters assigned to each connection of the network. This allows to find the weak connections before the training and remove them, avoiding the training complexity of iterative approaches.

Another approach to reduce the size and complexity of the DNNs, before the training procedure starts, is to choose and train network architecture that is designed specifically for low-power and low-memory devices. Such networks [105, 106] also contain fewer parameters from the outset. These solutions are usually built upon smart architectural choices, including inverted residuals [107], or pointwise convolutions and channel shuffle operations [106].

Structured and unstructured pruning. Network pruning techniques can be divided into two categories: structured and unstructured pruning. Unstructured pruning (see Fig. 2.2b) focuses on finding the weakest parameters in the DNN structure. This means, that the algorithm will treat each parameter of the DNN independently from the others, and, as a result, it will erase individual connections within the network by setting their values to 0, effectively producing a sparse DNN structure. Examples of unstructured pruning methods can be found in [97, 101]. One of the most important drawbacks of unstructured pruning is its limited applicability to general-purpose hardware. Classical GPUs are not able to benefit from these sparse structures within the DNNs. Despite many parameters being set to 0, GPUs will still try to execute the operation, thus no speedup will be achieved in this case. The only way to

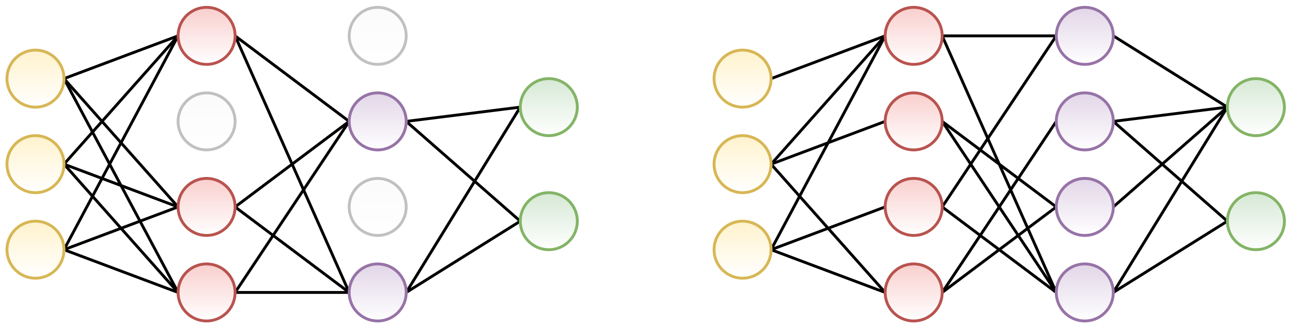

(a) Structured pruning (b) Unstructured pruning

Figure 2.2: A difference between structured and unstructured pruning techniques. Structured pruning removes entire neurons, while unstructured pruning removes individual connections, keeping the number of neurons unchanged.

benefit from sparsity in DNNs is to build specialized hardware specifically for the structure of the pruned network, which will omit the computations corresponding to redundant connections, effectively leading to significant speedups.

Structured pruning algorithms (see Fig. 2.2a), on the contrary, focus on removing structurally significant blocks from the DNN structure. This may correspond to removing entire convolutional filters or entire neurons, depending on the layer it is applied to. This means, that the structural blocks are analyzed as a whole, rather than individual parameters belonging to the block. Once a block is removed from the DNN, one can immediately see complexity and memory improvements from applying this type of pruning. Computations corresponding to this removed block can be effectively omitted, thus no specialized hardware is necessary to deploy a DNN that has been pruned with a structured pruning method. This has a large impact on the applicability and popularity of structured pruning methods, making them the preferred choice compared to unstructured pruning, despite their sub-optimal performance. Some examples of the structured pruning methods can be found in [98, 99, 102].

Chapter 3

Joint Source-Channel Coding for Wireless Image Retrieval

This chapter studies the problem of wireless image retrieval over the wireless edge [35, 36]. The system considered in this work consists of a CCTV camera, which collects images of pedestrians or vehicles, and has sufficient computational capabilities to process the images locally. The goal is to recognize the suspects within the images collected by the camera. This is a difficult task, due to the necessity of accessing a centralized database of suspects. To this end, the edge device that collects the data has to collaborate with an edge server to complete the task. In the considered system, we assume the camera (edge device) is connected through a bandwidth-constrained wireless link with the server which has access to the database of suspects. The classical approach to this problem is to compress images/frames collected by the camera and upload them to a centralized server, which performs further identification of the suspects. We note that transmitting images collected by the CCTV camera in real-time may be infeasible, due to the latency constraints of the underlying application. In our proposed solution, the images collected by the camera are first compressed into low-dimensional feature vectors by a deep neural network (DNN) and these features are transmitted through a wireless link to the server, which finds the most similar feature vectors stored in the database and recognizes the suspects. We propose two transmission schemes, a classical, separation-based approach, where the feature vectors are mapped into bit sequences and transmitted using conventional channel codes. Alongside this approach, we propose a joint source-channel coding scheme, which maps the feature vectors directly into channel inputs. Through extensive evaluations on multiple datasets, we show, that the latter approach is superior, as it achieves increased recognition accuracy and graceful degradation with channel conditions.

3.1 Introduction

Internet of Things (IoT) devices are becoming increasingly widespread. These small specialized computers are present in offices, streets, and homes. Their main goal is to continuously sense their environment, and send the measurements through a wireless channel to an edge server, which performs data collection and further processing. Typical approach in most IoT applications is to convey all the measurements from the IoT devices to an edge server, where state-of-the-art machine learning algorithms are used to analyse the collected data. However, in some applications, the volume of the measurement data (e.g., images, videos or LIDAR data) is large, and transmitting it to the server at the required quality may not be feasible within the limited latency requirements, e.g., in autonomous driving, surveillance, drones, etc. On the other hand, as the computational capabilities of IoT devices advance, they can process the data locally before offloading it to a server. In some cases the desired inference tasks can be carried out locally, which is beneficial as the IoT devices have access to the original data, rather than its quantized version at the edge server, due to the lossy compression and transmission over the wireless channels.

In this work, we study machine learning at the wireless edge. In particular, we focus on distributed inference over a wireless channel, where a centrally-trained algorithm is deployed on IoT devices to perform inference over-the-air. One of the machine learning tasks for which remote inference is essential is retrieval. In autonomous vehicles, drones, or in surveillance systems, agents try to identify objects, vehicles, or humans in their environment through their sensory data. The goal in image retrieval is to identify a query image of a person or a vehicle

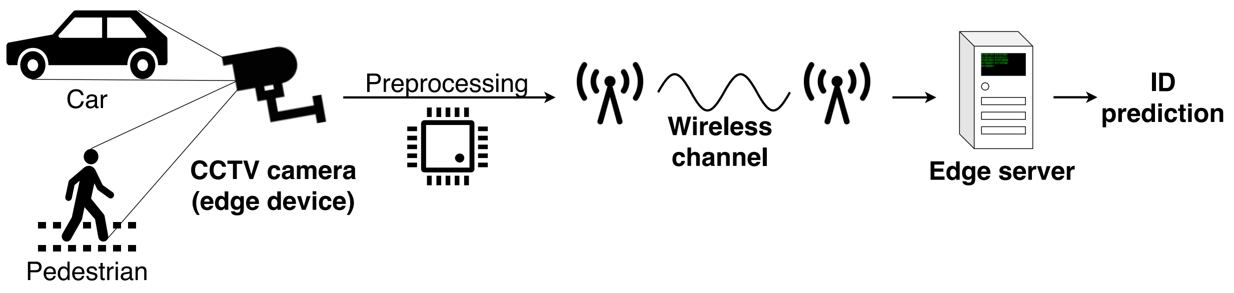


Figure 3.1: An illustration of the retrieval problem at the edge. A CCTV camera takes a picture of a pedestrian or a car, and processes the image locally to obtain a low-dimensional signature, which is then sent through a wireless channel to an edge server that performs identification based on a large database it has access to.

recorded locally by matching with images stored in a large database (gallery), typically available at the edge server (cf. Fig. 3.1). We emphasize that the retrieval task cannot be performed locally at the edge device regardless of its computational power. This is because the centralized database is available only at the edge server, hence, some sensory data has to be transmitted to the edge server. The fundamental question we want to answer in this chapter is what part or function of data must be transmitted, and how.

A trivial response to these questions would be to convey the image to the edge server at the best quality possible. The server first reconstructs the image, and performs the retrieval task with a state-of-the-art retrieval algorithm. Note, however, that a significant part of the image content may not be relevant for the retrieval task, therefore the original image is not needed at the server. Indeed, novel approaches to retrieval employ deep neural networks (DNNs) as feature encoders that map input images to a low-dimensional feature space, such that vectors extracted from the same identities are similar, despite different views or occlusions. Accordingly, we employ DNNs for extracting features that are then transmitted over the wireless link.

We propose two approaches to convey the feature vectors to the edge server. In the conventional "digital" approach, feature vectors are first compressed, and encoded with a channel code for reliable transmission. The features that are most relevant for the retrieval task are extracted and transmitted depending on the capacity and the reliability of the channel between the edge device and the server. To improve the efficiency of this approach, we design a retrieval-oriented image compression scheme, which compresses the feature vectors depending on the available bit budget. This “separate” data compression and channel transmission scheme assumes reliable communication over the channel. Such scenario is typically difficult to achieve in practice, especially for short blocklengths considered in this work, imposed by the strict delay limitations. Alternatively, we consider a joint source and channel coding (JSCC) approach, where the feature vectors are directly mapped into channel input symbols, and the noisy channel output is used by the server to retrieve the most relevant images, without involving any explicit channel code. This can be considered as “analog” communication since the feature vectors are not converted into bits at any stage. For the JSCC approach, we employ an architecture based on DNNs, similar to the novel DeepJSCC [33, 52], which has recently been introduced for wireless image transmission. Our results show that the JSCC scheme can outperform the highly optimized feature compression scheme even if we assume the availability of capacity-achieving channel codes for the digital scheme. To the best of our knowledge, this is the first work to study image retrieval over a wireless channel. The specific technical contributions related to the work presented in this chapter can be summarized as follows:

- We propose a novel retrieval-oriented image compression scheme, which combines a retrieval baseline with a feature encoder, followed by scalar quantization and entropy coding. To estimate the distribution to be used for the entropy coder, we introduce a density model based on a Gaussian mixture.

- We propose an autoencoder-based architecture and training strategy for robust JSCC of feature vectors, generated by a retrieval baseline, under noisy, fading, and bandwidth-limited channel conditions.

- We perform extensive evaluations under different signal-to-noise ratio (SNR) and bandwidth constraints, and show that the JSCC scheme outperforms the digital approach even with capacity-achieving channel codes. Moreover, its performance exhibits graceful degradation when the test and training SNRs do not match. The JSCC scheme is shown to outperform its digital counterpart also over fading channels, even if we assume the availability of channel state information for the digital scheme only.

- We evaluate the proposed schemes on various surveillance tasks, and show that the performance close to the noiseless bound can be achieved even under very harsh SNR and bandwidth constraints, whereas the digital approach falls short of this performance even with idealistic capacity-achieving channel codes.

- Our results show that, in general, it is not possible to separate inference tasks from the communication scheme, and the end-to-end performance can be improved significantly by designing the communication and learning algorithms jointly. We provide a comprehensive analysis of different architectures and training strategies that will serve as solid baselines for future research in wireless edge learning.

3.2 Related work

3.2.1 Person and Vehicle Retrieval

Person and vehicle retrieval tasks have been extensively studied [108, 109, 110, 111, 112, 113, 114]. They share the same motivation to allow for a better and more reliable recognition of people and vehicles, mainly targeting surveillance applications. The most successful recent approaches for image retrieval problems are based on convolutional neural networks, and recent techniques include part classifiers [108, 109], creating bias-invariant feature vectors [110, 114], using attention models [112], and analyzing images at different scales [115, 116]. Despite the popularity of triplet loss in both areas [117, 111], designs based on softmax cross-entropy have also been successfully implemented [113].

3.3 Methods

In this work we propose two approaches for performing retrieval over wireless channels: digital (separate) and JSCC (joint) approaches. In both cases, we consider the transmission of the feature vectors, which are a low-dimensional representation of identities of the items to be retrieved e.g., humans, vehicles (Section 3.3.2), and have to be sent over bandwidth-limited wireless channels. Due to the channel limitations, features cannot be transmitted in a lossless fashion, and have to be compressed. The recovered noisy feature vectors at the receiver are compared to the feature vectors of images previously collected from other edge cameras, called the *gallery*, in order to find the nearest neighbour.

3.3.1 Channel Model

We assume that the edge device is connected to the edge server through an additive white Gaussian noise (AWGN) channel. We consider static as well as slow fading channel. For both approaches presented in this work, we assume that the channel model is known during training, and remains the same during inference.

The AWGN channel is characterized as follows: given a channel input vector $\mathbf{x} \in \mathbb{C}^B$, consisting of B complex channel input symbols x_i, the output $\mathbf{y} \in \mathbb{C}^B$ is given by $\mathbf{y} = \mathbf{x} + \mathbf{z}$, where $z_i \sim \mathcal{CN}(0, \sigma^2)$ are the independent and identically distributed (i.i.d.) elements of the noise vector $\mathbf{z} \in \mathbb{C}^B, i = 1, \ldots, B$. An average power constraint is imposed on the input vectors, such that $\frac{1}{B}\sum_{i=1}^{B} |x_i|^2 \leq P = 1$; which, in the case of a static AWGN channel, translates into a maximum received SNR of $\text{SNR} = 10\log_{10}(\frac{1}{\sigma^2})$ in dB scale.

In the slow fading scenario, we consider a single-tap Rayleigh fading channel model, where all the transmitted symbols experience the same channel gain. That is, given the channel input vector $\mathbf{x} \in \mathbb{C}^B$, the corresponding output vector $\mathbf{y} \in \mathbb{C}^B$ is given by $\mathbf{y} = h\mathbf{x} + \mathbf{z}$, where $h \sim \mathcal{CN}(0, \sigma_h^2)$ and $z_i \sim \mathcal{CN}(0, \sigma^2)$ are drawn from independent zero-mean complex normal distributions with variances σ_h^2 and σ^2, respectively. We impose the same average input power constraint of $P = 1$ as in the AWGN case. For each transmitted feature vector we use a single gain h, which characterizes the *slow fading* behaviour. The maximum average SNR is evaluated by $\text{SNR} = 10\log_{10}(\frac{\sigma_h^2}{\sigma^2})$ dB, while for all the experiments shown in this work we set $\sigma_h^2 = 1$, which corresponds to the same average received power as in the static AWGN channel model.

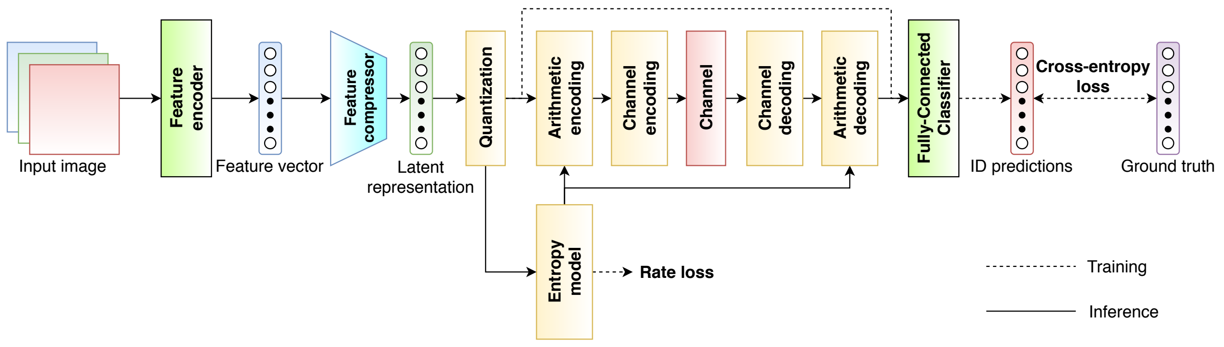


Figure 3.2: The digital transmission scheme. Input is transformed into a feature vector, which is compressed using a DNN. At the receiver, latent representation is classified into IDs to compute the loss during training only. Arithmetic coding and channel coding is bypassed during training.

3.3.2 Retrieval Baseline

Following the state-of-the-art retrieval methods [113, 109] we employ the ResNet-50 network [93], pretrained on ImageNet [24], for feature extraction. This ensures that similar results can be expected in different setups. In more detail, we use ResNet-50 with batch normalization (BN) layers applied after each convolutional layer. As input, we use images resized to a common 256×128 resolution with bicubic interpolation for person datasets and 128×128 resolution for vehicle datasets. For the last layer we use average pooling across all the feature maps, which results in a 2048-dimensional feature vector. During training we use stochastic gradient descent (SGD) with a learning rate of 0.01 and a momentum of 0.9. A relatively large value of the learning rate is motivated by the fact that we do not intend the network to fully converge at this stage, as it will be fine-tuned together with the JSCC network in the subsequent steps of the training procedure. We also apply L_2 regularization, weighted by $5 \cdot 10^{-4}$ to the ResNet-50 parameters. We refer to this architecture as the *feature encoder*.

3.3.3 Digital Transmission of Compressed Feature Vectors

This approach is based on the assumption that a certain number of bits can be reliably conveyed to the edge server for each image. In practice, however, this is highly challenging to achieve. Ultra-reliable channel codes require large blocklengths even in the static AWGN setting, and accurate channel estimation and feedback in the slow-fading case. In our simulations, we assume capacity-achieving channel codes, which will serve as a bound on the performance of practical digital schemes.

An overview of the proposed digital scheme is shown in Fig. 3.2. We first extract features using the retrieval baseline described in Section 3.3.2 as feature encoder. The resulting feature vector is compressed into as few bits as possible through lossy compression followed by arithmetic coding. The compressed bits are then channel coded, with introduced structured redundancy to combat channel impairments.

The lossy feature compressor consists of a single fully-connected layer for dimensionality reduction, followed by quantization. On the receiver side we use the quantized latent representation as a feature vector, which is passed through a fully-connected layer for ID classification. Note that the IDs and their classification are used for calculating the loss during training only. During retrieval, the IDs are not known and the feature vectors are used for nearest neighbour search. This has been shown to perform well in the re-ID community [108, 109, 110, 111, 112, 113, 114].

To enable an end-to-end differentiable approach, we utilize the well-known quantization noise [80] to model the quantization process. Specifically, instead of rounding the latent representation to the nearest integer, in the training phase we add the uniform noise to each element of the latent representation as follows:

$$Q(\mathbf{z}) = \mathbf{z} + \mathcal{U}\left(-\frac{1}{2}, \frac{1}{2}\right), \tag{3.1}$$

where $Q(\cdot)$ is the approximated quantization operation, $\mathbf{z}$ is the latent representation, and $\mathcal{U}(\cdot,\cdot)$ is the uniform noise vector. This formulation ensures a good approximation of quantization during training, whereas we perform rounding to the nearest integer during inference.

In order to optimize the arithmetic coder, we estimate the distribution of the quantized outputs. We assume that the elements q of vector $\mathbf{q} = Q(\mathbf{z})$ are i.i.d. with some probability mass function (PMF) $p(q)$. To model this PMF, we propose a simple yet flexible solution using a mixture of Gaussians. We first approximate $p(q)$ as a continuous-valued probability density function $p_c(q)$ as follows:

$$p_c(q) = \sum_{k=1}^{K} \alpha_k \frac{1}{\sigma_k \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{q-\mu_k}{\sigma_k}\right)^2}, \tag{3.2}$$

where K is the number of mixtures, σ_k are mixture scales, μ_k are mean values, and α_k are the corresponding mixture weights. In our experiments we set $K = 9$, which we empirically found to perform the best. Then, in order to evaluate our PMF $p(q)$ at discrete values $q \in \mathbb{Z}$, we integrate $p_c(q)$ over $\left[q - \frac{1}{2}, q + \frac{1}{2}\right]$ to obtain:

$$p(q) = \int_{q-\frac{1}{2}}^{q+\frac{1}{2}} p_c(x)dx = F_c\left(q + \frac{1}{2}\right) - F_c\left(q - \frac{1}{2}\right), \tag{3.3}$$

where F_c is the cumulative density function of the distribution $p_c(q)$.

We remark that, here we learn the distribution of the quantized feature vectors, but unlike recent works [79, 118] , we do not consider adaptive probability model and do not introduce another neural network to predict parameters $\{\alpha_k, \mu_k, \sigma_k\}$ of the mixture. The reason for that is the proposed simple model performs sufficiently well, and we want to avoid introducing any communication overhead by sending additional parameters per image. Instead, we use the available bandwidth for sending quantized feature vectors only.

With the model presented above, we can easily estimate the PMF of the quantized vector $\mathbf{q}$, which can be directly used to feed the arithmetic coding engine in the test phase, but also to evaluate the average approximate entropy over the dataset in our loss function, which we define as a weighted sum of two objectives:

$$L = l_{ce} - \lambda \cdot \log_2 p(\mathbf{q}), \tag{3.4}$$

where l_{ce} is the cross-entropy between the predicted class (identity) and the ground truth for the retrieval task. The second component of the loss function corresponds to the empirical Shannon entropy of the quantized vector, representing the average length of the output of the arithmetic encoder. Such formulation allows for a smooth transition between the retrieval accuracy and number of bits necessary to send the feature vector in a lossy fashion. Moreover, minimizing the

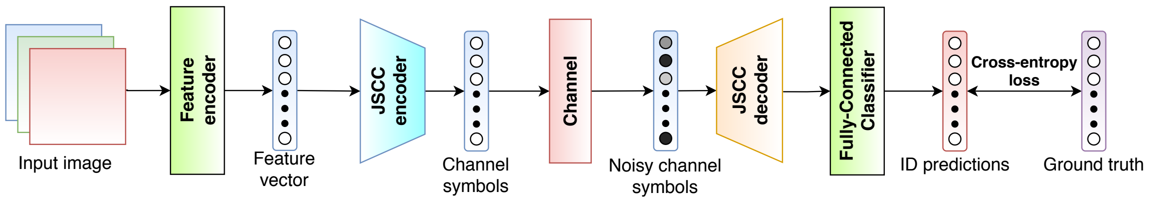


Figure 3.3: The architecture and training of JSCC of feature vectors for wireless image retrieval. The feature vector is directly mapped to channel inputs. Received noisy signal is decoded and processed by a fully-connected layer to obtain ID predictions, which are then compared to the ground truth by the cross-entropy loss.

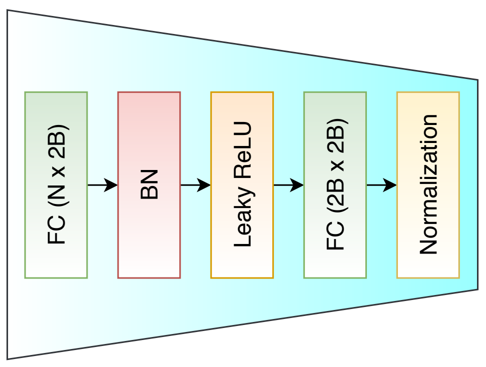
entropy term is equivalent to maximizing the likelihood of $p(q)$, which corresponds to increasing the certainty of our model, and allows a satisfactory fit of our approximated distribution to the true underlying distribution of the discrete symbols.

We apply the same settings discussed in Section 3.3.2 to train the feature encoder, the fully-connected classifier and the density model. We train the whole network for 20 epochs, reduce the learning rate to 0.001 and train for further 30 epochs. We initialize our mixture parameters as follows: $\alpha_k = \frac{1}{K}$, $\mu_k = 0$, $\sigma_k = k^2, k = 1, 2, \dots K$. To ensure the convergence during training, in the first epochs we prioritize the l_{ce} loss term by setting the weight parameter λ_i at epoch i as:

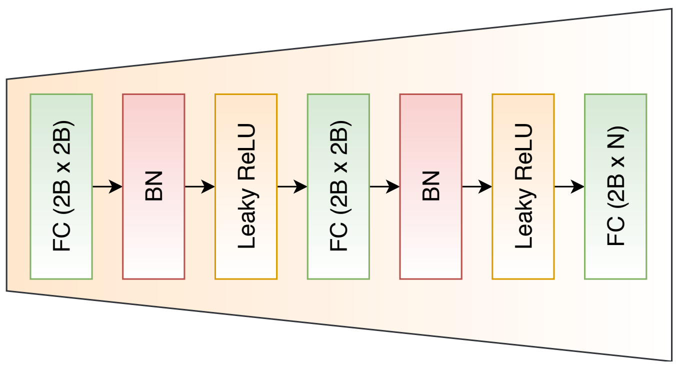
$$\lambda_i = \min\left(\lambda \frac{i}{E-20}, \lambda\right), \; i = 1, \dots, E, \tag{3.5}$$

where $E > 20$ is the total number of epochs. In our experiments we use $\lambda \in [10^{-5}, 10^{-2}]$, and $E = 50$.

In the inference phase we use the arithmetic encoding engine to transmit the information with a channel code. Note that any channel code can introduce errors, there is therefore an inherent trade-off between the compression rate and the channel coding rate under a given constraint on the channel bandwidth, i.e., the number of channel symbols that can be transmitted to the edge server per image pixel. Compressing the feature vector further leads to increased distortion, and hence, reduced retrieval accuracy, but also allows to introduce more redundancy, and hence, increased reliability against noise. In general, the optimal compression and channel coding rates



(a) JSCC encoder



(b) JSCC decoder

Figure 3.4: Proposed JSCC encoder and JSCC decoder architecture for the JSCC scheme illustrated in Fig. 3.3. At the encoder, dimensionality reduction is performed by the first fully-connected layer, which is inverted at the decoder.

depend on the retrieval accuracy-compression rate function of the compression scheme and the error-rate of the channel code. To simplify this task, we assume capacity-achieving channel codes, which provides an upper bound on the performance that can be achieved by any digital scheme that uses the above architecture.

3.3.4 JSCC of Feature Vectors

In this section, we propose an alternative JSCC approach, called JSCC AE, and illustrated in Fig. 3.3. We use the baseline feature encoder as before to produce the feature vector for a given query image. The feature vector is mapped directly to the channel input symbols via a multi-layer fully-connected JSCC encoder (Fig. 3.4a). We set the dimensionality of the channel input vector to $2B$ real symbols, which corresponds to the available channel bandwidth of B complex values. In this work we consider small values of B modeling stringent delay constraints of the underlying surveillance applications. This low-dimensional representation is normalized to satisfy the average power constraint of $P = 1$, and transmitted over the AWGN channel. The noisy channel output vector at the receiver is mapped back to the high-dimensional feature space by a JSCC decoder (Fig. 3.4b). The distance between the query feature vector and the feature vectors stored in the gallery set is calculated to find the nearest neighbours.

In order to train our network, the most straightforward strategy would be to perform end-to-end training, taking images from the dataset as an input, and training both the feature encoder and the JSCC autoencoder jointly, in an end-to-end fashion, with cross-entropy loss between the ID predictions and the ground truth (as shown in Fig. 3). However, our experimentation in Section IV-F shows that this approach leads to suboptimal performance. Alternatively, we propose traininig each component of the network separately at first, and, once the feature encoder and the JSCC autoencoder are pretrained individually, they are combined and trained jointly. Therefore, our training strategy, which we refer to as $T_{1,2,3}$, consists of three steps: feature encoder pretraining (T_1), JSCC autoencoder pretraining (T_2), and end-to-end training (T_3). In the first step, T_1, we attach a single fully-connected layer at the end of the feature encoder that maps 2048-dimensional feature vectors directly to the ID predictions. We pretrain the feature encoder for 30 epochs with a batch size of 16, using cross-entropy between the ID predictions and the ground truth as the loss function. In the second step, T_2, we freeze the pretrained feature encoder, and use it to extract features from all the images in the training dataset. We use these features as inputs to the proposed autoencoder network. We train the autoencoder using the L_1-loss between the feature vectors and the vectors reconstructed by the JSCC decoder. It is trained with SGD for 200 epochs with a learning rate 0.1, reduced to 0.01 after 150 epochs, and momentum of 0.9. We apply L_2 regularizer to the autoencoder model, weighted by $5 \cdot 10^{-4}$. Finally, in the third step, T_3, we train the whole network jointly, the autoencoder and the feature encoder, for 30 epochs, using the cross-entropy loss with a learning rate 0.001, and for further 10 epochs with a learning rate of 0.0001, applying the same optimizer and L_2 regularization as in the previous two steps.

Along with $T_{1,2,3}$ we evaluate four alternative training strategies. The first one, denoted by T_3, corresponds to the end-to-end training of the entire network (feature encoder + JSCC autoencoder + classifier) in a single training step. The second method, $T_{1,2}$, consists of the feature encoder pretraining, T_1, followed by the JSCC autoencoder training, T_2 to reconstruct feature vectors with L_1 as the distortion measure. This method corresponds to using a JSCC scheme whose goal is to reconstruct the feature vector as reliably as possible without taking into account the accuracy of the retrieval task. After T_2, the feature encoder and the autoencoder are combined as in Fig. 2, but the joint training step, T_3, is not performed. The third method, $T_{1,3}$, consists of the feature encoder pretraining, T_1, followed by joint training of the entire network, T_3. Finally, $T_{1,3} + L_1$ approach is different from the $T_{1,3}$ in that it combines the cross-entropy loss and L_1 loss, in the joint training phase.

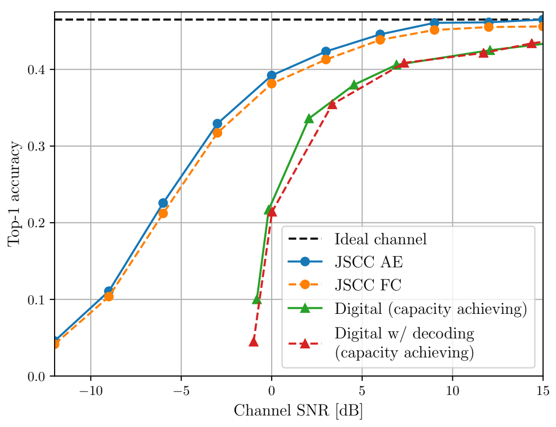
Note that, we opted for an architecture that employs a distinct feature encoder and a separate JSCC autoencoder to transmit the feature vector over the channel. We have then trained these components in multiple training steps. It is possible to introduce a simpler architecture with a single JSCC encoder at the edge device that maps the query image to the channel input vector. Thus, no decoding is required at the receiver, and the retrieval task is directly performed using the noisy channel symbols. To compare our method to this straightforward approach, we introduce JSCC FC, which follows the same structure as in Fig. 3.3, except that the JSCC encoder is replaced by a single fully-connected layer and the JSCC decoder is removed. We train the whole network end-to-end for 50 epochs with cross-entropy loss, learning rate of 0.01, reduced to 0.001 after 30 epochs, and a momentum of 0.9. We also apply L_2 regularization, weighted by $5 \cdot 10^{-4}$, to all the parameters, including ResNet-50, feature encoder and fully-connected classifier.

3.4 Results

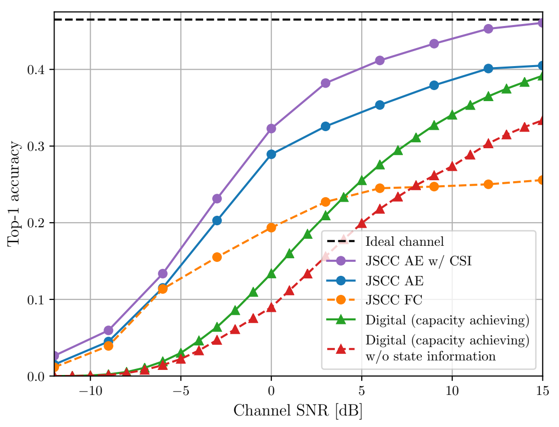
In this section we evaluate the performance of the proposed JSCC AE and JSCC FC architectures, and compare with that of the digital scheme presented in Section 3.3.3, as well as the *ideal channel* scenario with unlimited channel resources, where full, noiseless feature vectors can be transmitted over the channel. We first discuss the experimental setup and the dataset used for the evaluations.

3.4.1 Experimental Setup

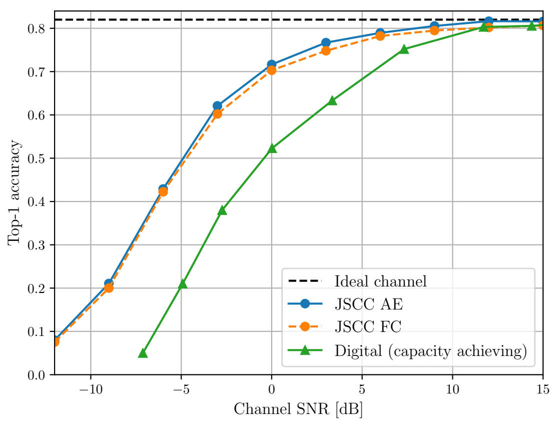
For the JSCC AE and JSCC FC schemes we vary channel SNRs for training, between $\mathrm{SNR}_{train} = -12$dB and $\mathrm{SNR}_{train} = \infty$dB, which corresponds to zero noise power. Training and test SNRs



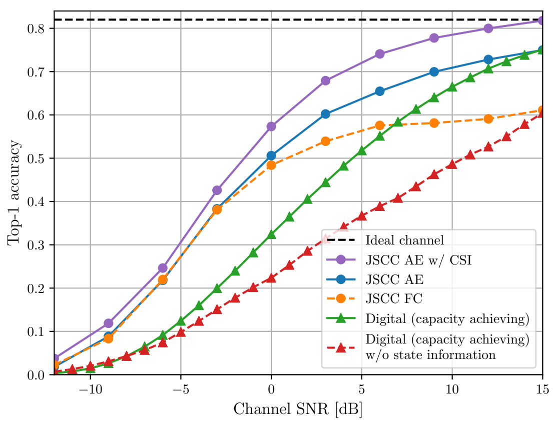
(a) Person re-ID CUHK03 - AWGN



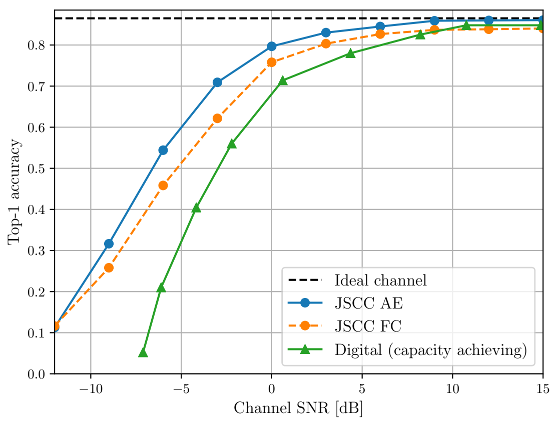
(b) Person re-ID CUHK03 - fading



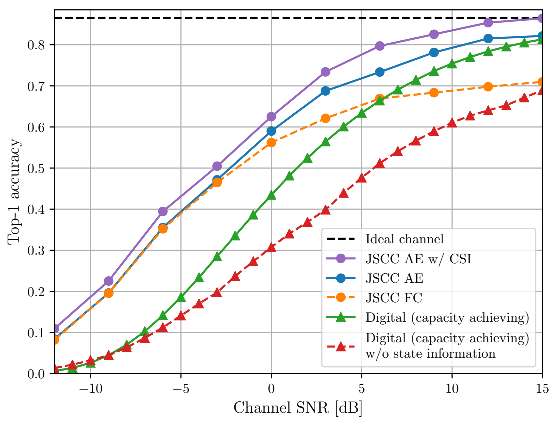
(c) Person re-ID Market-1501 - AWGN



(d) Person re-ID Market-1501 - fading



(e) Car re-ID VeRi - AWGN



(f) Car re-ID VeRi - fading

Figure 3.5: Performance comparison of the proposed three schemes over AWGN and slow fading channels for a range of channel SNRs and bandwidth $B = 64$. Our JSCC AE scheme achieves the best retrieval accuracy over the whole range of tested SNRs and for all three re-ID image retrieval datasets.

are the same unless stated otherwise. In the digital scheme, we experiment with different dimensionality of the latent representation, between 64 and 512, estimate and minimize its entropy in the training phase by varying the value of parameter λ. In the testing phase we perform rounding to the nearest integer on each element of the latent representation and arithmetic coding, which is based on the probabilistic model learned by the entropy estimator, as described in Section 3.3.3. This model assigns a probability estimate to each quantized symbol, which is then passed to the arithmetic encoder. We note that the proposed digital scheme is a variable-length encoder. Therefore, for a given fixed communication rate to the server, one has to determine the λ coefficient that meets the rate constraint for each image. Instead, we fix the λ coefficient and calculate the average number of bits required to encode the latent representations of the test images. We then evaluate the corresponding channel SNR to deliver these many bits to the receiver, assuming capacity-achieving channel codes. This is the upper bound on the real performance as practical codes are far from the capacity bound in the short blocklength regime. This model may correspond to sending multiple images together, and hence, the performance is determined by the average rate across many test images, rather than their individual rates.

For digital transmission over a fading channel, we consider two scenarios. In the first one, we assume perfect channel state information available at both the transmitter and the receiver. Then, for each query image and a corresponding random channel gain, we identify the λ parameter that results in a bit rate that is as close as possible from below to the corresponding channel capacity. Then, we find the average accuracy across many random queries and channel conditions, following the underlying fading distribution. In the second scenario, we fix the λ parameter, and for each query image and the corresponding random channel condition, we compare the required bit rate of the query image and the channel capacity. If the capacity is lower than the bit rate required by the compression scheme, we assume the transmission is failed. We then calculate the fraction of successful transmissions and multiply it by the average accuracy of the queries whose compressed feature vector can be successfully transmitted, for a given λ. Note that, there is a trade-off between the accuracy loss due to compression and the outage over the channel. The higher λ values results in more compact representations of the feature vector, and hence less accurate retrieval performance even if they can be successfully conveyed to the server. Higher λ values relaxes the compression constraint, but may result in higher loss over the channel. Note that, we report only the results for the λ values that lead to the highest average accuracy for each average SNR.

To train our model we used NVIDIA GeForce RTX 2080Ti GPU. A single end-to-end training of our digital model took approximately 35 minutes, which was similar to the training time of the JSCC FC. For JSCC AE, the training took approximately 20 minutes, 3 minutes, and 30 minutes for T_1, T_2, and T_3, respectively. Please note that T_1 has to be performed only once, as this step does not depend on the channel model.

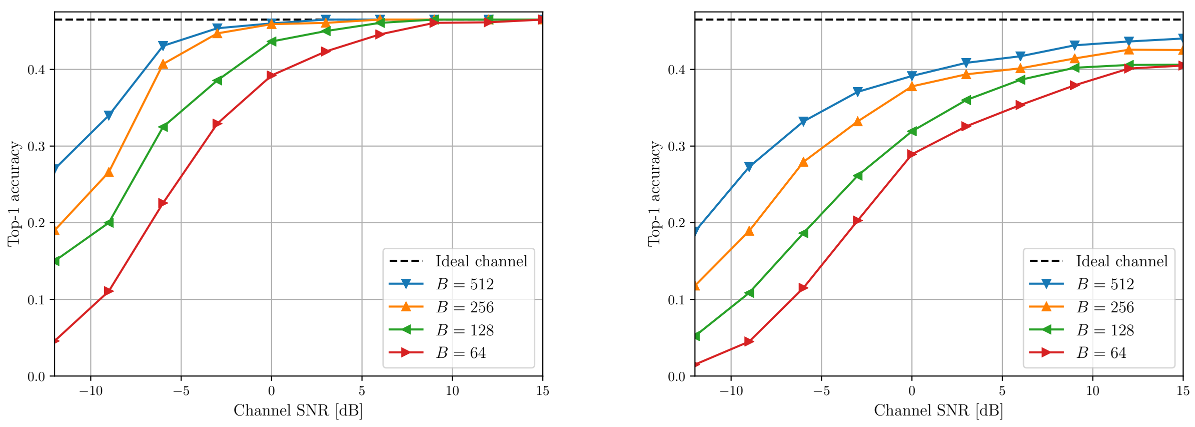
3.4.2 Datasets

In order to measure the performance of the retrieval task, we employ three widely used datasets:

CUHK03 [119] is a benchmark for person retrieval that contains 14096 images of 1467 identities taken from two different camera views. The dataset was captured with six surveillance cameras and each identity within the dataset is represented by an average of 4.8 images per each of the two camera views. We use the *labeled* variant of the dataset, where each image of the pedestrian was manually cropped by a human.

Market-1501 [120] contains 32217 images of 1501 pedestrians taken from a total of six cameras in front of a supermarket at Tsinghua University. Five out of six cameras are high-resolution cameras and the remaining one is low-resolution. Training and testing splits proposed by the authors contains 12936 and 19732 images, respectively. 750 identities are additionally selected as a query set containing 3368 images (maximum of 6 per person). The dataset is different from CUHK03 in that it contains junk images capturing only partial pose and distractors presenting small fragments of pedestrian appearance or irrelevant objects.

VeRi [121, 122] is a vehicle retrieval dataset. It contains over 50000 images of 776 vehicles captured by 20 cameras within 24 hours over the area of 1km^2. Each identity is captured by 2-18 cameras in different viewpoints, occlusions, resolutions, and lighting conditions. All the



(a) AWGN channel (b) Fading channel

Figure 3.6: Accuracy as a function of the channel SNR for different channel bandwidths. Higher bandwidth introduces more robustness against the channel noise.

images within the dataset are annotated with attributes, brands and colors, but in this work we do not utilize this information, and focus on retrieving the identity only based on the image.

The evaluation measure for all the datasets is the top-1 retrieval accuracy, which calculates the fraction of correct IDs at the top of the ranked list retrieved for each query.

3.4.3 Performance for Different Methods

We plot the accuracy achieved by various schemes as a function of the test SNR in Fig. 3.5. For these experiments we use the bandwidth of 64, which corresponds to the transmission of 64 complex symbols through the channel. One can see that JSCC AE outperforms the digital scheme in all considered scenarios. For CUHK03 dataset the digital approach is not able to recover the noiseless accuracy even at SNR = 15dB, whereas the proposed JSCC AE scheme obtains accuracy close to the ideal channel baseline at around 10dB for the AWGN channel. JSCC FC follows JSCC AE very closely, but the increase in the performance provided by the autoencoder is visible for all the SNRs considered, which proves the superiority of the proposed architecture in comparison to the relatively simpler JSCC FC. The lower accuracy of JSCC FC may stem from the fact that the noise directly affects the low-dimensional feature vector, while the autoencoder-based scheme introduces certain level of denoising, which improves the feature estimates at the receiver. In Fig. 3.5a, we also show that feature decoding is not beneficial for the digital scenario. An alternative scheme, which we called Digital w/ decoding (capacity achieving), follows the same training strategy as discussed in Section 3.3.3, but we further introduce a fully-connected decoder. This decoder is placed before the fully-connected classifier, and maps low-dimensional quantized latents back to the original, 2048-dimensional feature vector space. We show that this decoding step brings no improvement to the digital scheme performance, compared to the scenario without decoding. This result was consistent across all the datasets, but to avoid clutter we show it only in Fig. 3.5a. Another observation is that the relative performances of the three schemes are similar for all the datasets considered, while JSCC FC seems to perform worse for the Car VeRi dataset, and even surpassed by the digital scheme at SNR = 10dB.

Fading channels introduce additional perturbation to the channel symbols, reducing the accuracy of all the proposed approaches. Similarly to the AWGN channel, JSCC AE achieves the best performance across all three datasets and the average SNR values considered in this chapter. The digital scheme performs worse when the channel state information is not available (which is also the case for the JSCC schemes). We have also included the performance of the digital scheme when perfect channel state information is available. We observe that even in this case the proposed JSCC AE scheme outperforms the digital alternative. JSCC FC closely follows JSCC AE at the low SNR regime, but its performance saturates to a level significantly below that of JSCC AE, and even below the digital scheme for the CUHK03 dataset. This result further validates the denoising interpretation of the autoencoder structure in JSCC AE, which becomes even more critical in recovering the noisy feature vector in the presence of channel fading. Fading not only applies random attenuation to the received signal strength, but also random rotations in the complex plain, which makes it very difficult for the receiver to recover the features for correct retrieval without any channel state information. We note that, while the digital scheme suffers significant performance loss in the absence of the channel state information, JSCC AE seems to perform reasonably well. We can argue that the autoencoder learns to mitigate the effect of random fading despite the lack of explicit pilot signals. We also provide the performance of JSCC AE, when perfect channel state information is available at the receiver. The JSCC decoder first divides the received signal by the channel gain: $\hat{\mathbf{y}} = \frac{\mathbf{y}}{h} = \mathbf{x} + \frac{\mathbf{z}}{h}$, and reconstructs the input feature from $\hat{\mathbf{y}}$. In this scenario, our method is able to recover the noiseless bound at a reasonable SNR of 15dB, and the gap between JSCC AE w/ CSI and the digital approach grows even further.

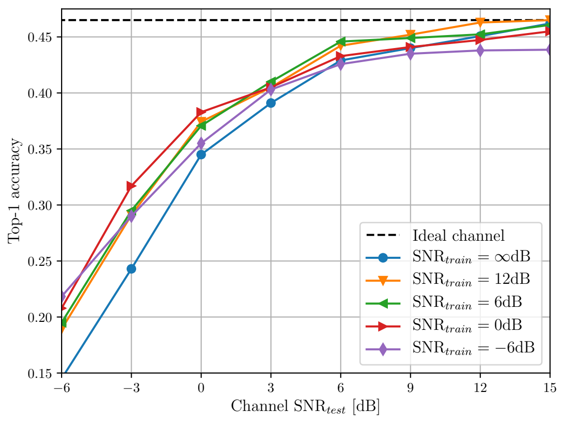
3.4.4 Performance for Different Bandwidths

In this experiment we investigate the effect of the channel bandwidth B on the retrieval performance for the person retrieval CUHK03 dataset, achieved by the JSCC AE scheme. We emphasize that the previously considered bandwidth of $B = 64$ is extremely limited, corresponding to extremely low-latency communications, which may be essential for many surveillance and security applications. The top-1 accuracy as a function of the channel SNR is plotted in Fig. 3.6 for different channel bandwidth values of $64, 128, 256$, and 512. It shows that the accuracy and robustness increases significantly with the bandwidth, but the relative gain becomes smaller as we approach the original feature vector dimension.

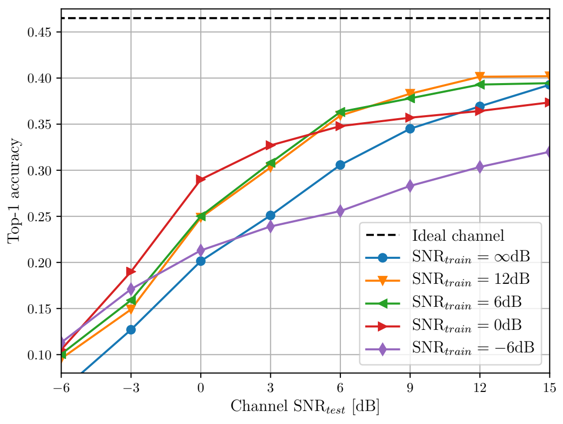
For the fading channel, it is visible in Fig. 3.6b that the proposed JSCC AE scheme without the channel state information is not able to recover the original accuracy even for a significant bandwidth, and reaches a plateau at around SNR = 12dB. As pointed out in Section 3.4.3, this may stem from the fact that our approach cannot fully cancel the effect of the variable channel gain. Channel estimation and feedback techniques can be utilized to mitigate the impact of random channel fading, as shown in Section 3.4.3.

3.4.5 Graceful Degradation

In this section we evaluate the behaviour of our models on the CUHK03 dataset when the training and test SNRs do not match. In the experiments with the digital scheme, we assume that capacity-achieving channel codes are in use, and the quality of the channel is always estimated correctly. However, in practice, digital approaches suffer from the *cliff effect*, which results in a sharp decrease in the performance when the channel condition is worse than the



(a) AWGN channel



(b) Fading channel

Figure 3.7: Accuracy achieved by the proposed JSCC AE scheme as a function of SNR$_{test}$ for different SNR$_{train}$ values for $B = 64$. JSCC AE achieves graceful degradation with the channel SNR as opposed to the digital scheme, which suffers from the cliff effect. Models trained at moderate SNR$_{train}$ values achieve relatively good performance for a wide range of test SNRs values.

channel state, for which the channel code is designed. If the code rate is above the current channel capacity, it is known that true error probability converges to 1 [123].

On the other hand, unlike digital models, analog transmission schemes are known to achieve graceful degradation when we are interested in the end-to-end reconstruction quality [33]; that is the average reconstruction quality smoothly decreases as the channel conditions become worse. This behaviour is quite beneficial, since we do not have to train multiple autoencoders, one for each channel SNR value, or even introduce channel estimation and feedback feature if the performance does not critically depend on applying the same training and testing SNRs. In the previous sections we showed the best possible accuracy for a specific SNR, which means each data point corresponds to a model trained specifically for that targer SNR. In Fig. 3.7 we show that graceful degradation can be achieved with the proposed JSCC AE architecture, and it is not necessary to train a separate model for every SNR value. Instead, we can take a model trained with a moderate SNR$_{train}$ and apply it to a wide range of SNR$_{test}$ in the inference time at the expense of a moderate loss in accuracy. To the best of our knowledge, this is the first time graceful degradation is demonstrated for the inference as opposed to the average reconstruction quality that is typically considered in the literature.

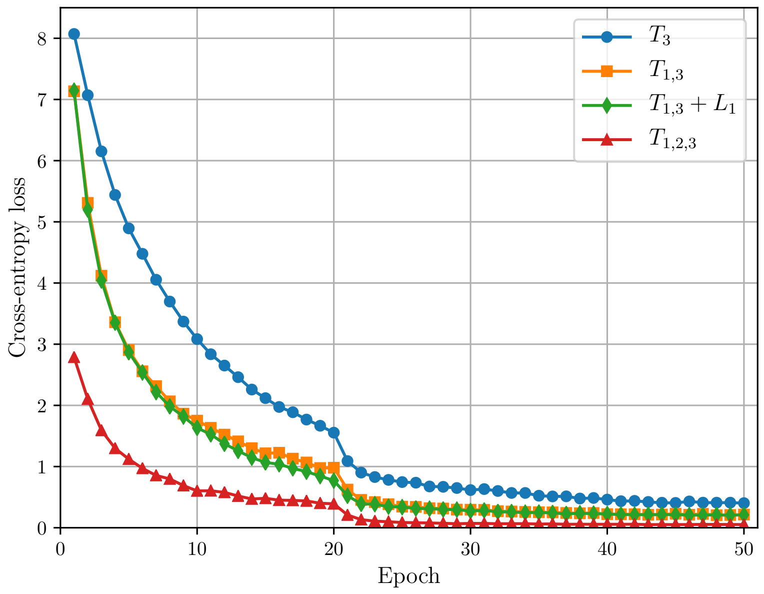


Figure 3.8: Comparison of training strategies through the evolution of the cross-entropy loss in the final joint training step. The proposed $T_{1,2,3}$ is superior to the alternative approaches.

Note that the approach trained without noise ($\text{SNR}_{train} = \infty$dB) is not robust against the channel noise. Therefore, its accuracy decreases much faster than for the networks trained under different noise levels, yet it still shows graceful degradation as the channel noise increases.

3.4.6 Training Strategy

In this section we show the superiority of $T_{1,2,3}$ training strategy by comparing to the alternative training methods introduced in Section 3.3.4. Note that, for the fairness of the comparison, we perform the first step of the training, which is the feature encoder pretraining, only once for $T_{1,2,3}$, $T_{1,2}$ $T_{1,3}$, and $T_{1,3} + L_1$.

The evolution of the cross-entropy loss over training epochs of the final joint training phase for different training strategies is shown in Fig. 3.8. In the experiment we used the bandwidth $B = 64$ and SNR=0dB. The proposed three-step training allows to achieve much better final performance, as shown in Table 3.1. Here, we also shown the top-5 recognition accuracy i.e., the correct match was listed within the top 5 ranklist elements, and the mean average precision

Table 3.1: Comparison of the retrieval performance for different training strategies.

Method	Top-1 accuracy	Top-5 accuracy	mAP
T_3	0.225	0.409	0.195
$T_{1,3}$	0.312	0.533	0.286
$T_{1,3} + L_1$	0.317	0.536	0.287
$T_{1,2}$	0.330	0.557	0.306
$T_{1,2,3}$	**0.392**	**0.602**	**0.351**

Table 3.2: Person retrieval accuracy for the CUHK03 dataset achieved by different models at SNR = 0dB and $B = 64$.

Model	# JSCC encoder layers	# JSCC decoder layers	Activation	MSE	Top-1 accuracy	Top-5 accuracy	mAP
A	3	3	Leaky ReLU	0.204	0.382	**0.602**	0.354
B	3	2	Leaky ReLU	0.222	0.391	0.597	0.354
C	3	4	Leaky ReLU	0.199	0.390	0.601	0.358
D	2	3	Leaky ReLU	0.202	**0.392**	**0.602**	**0.359**
D	4	3	PReLU	**0.181**	0.383	0.589	0.343
E	4	3	Leaky ReLU	0.208	0.383	0.598	0.356
F	1	1	N/A	0.207	0.387	0.592	0.352
G	2	2	Leaky ReLU	0.206	0.387	0.592	0.352
H	1	2	Leaky ReLU	0.207	0.386	0.593	0.353
I	2	1	Leaky ReLU	0.206	0.389	0.600	0.356

(mAP), which are standard evaluation measures for the retrieval tasks. Adding each training step increases the performance gradually, and there is a significant difference between T_3, and $T_{1,3}$, as well as between $T_{1,3}$ and $T_{1,2,3}$. Our three-step strategy outperforms all three alternatives by a large margin as it converges faster and achieves the smallest loss after the last epoch. As expected, T_3 performs the worse, since it has to learn both the retrieval and robustness against the noise in a single training step with randomly initialized weights. Interestingly, the convergence of the $T_{1,3} + L_1$ seems to slightly outperform the convergence of the $T_{1,3}$, thanks to the additional loss term, which forces the reconstructed features to be similar to the original ones. However, while this seems to speed-up the convergence of the autoencoder network marginally, it does not affect the final performance. The reasonable performance of $T_{1,2}$ shows that T_2 allows the autoencoder to produce good reconstructions of the feature vectors under noisy environment, but the gap between $T_{1,2}$ and $T_{1,2,3}$ indicate the necessity of joint training phase, T_3, which maximizes the task performance. One may argue that our $T_{1,2,3}$ strategy is slower compared to the alternatives, nevertheless adding the autoencoder pretraining phase is negligible in comparison to the joint training phase ($\sim$ 3min vs. $\sim$ 1hr).

3.4.7 Comparison of Different Models

In this section we present the results of architecture search for the JSCC autoencoder that resulted in the best performing model presented in Fig. 3.4. We considered 9 models designed as follows: both the JSCC encoder and the JSCC decoder are built of fully-connected layers, followed by the BN and activation layers. The only exceptions are the last layers in the JSCC encoder and the JSCC decoder which are without BN and activations. The first layer of the JSCC encoder maps 2048-dimensional features to $2\boldsymbol{B}$ real-valued symbols, which eventually forms a $\boldsymbol{B}$ complex symbols transmitted over the channel. Similarly, the last layer of the JSCC decoder maps $2\boldsymbol{B}$-dimensional vectors back to the original 2048-dimensional feature space and the remaining FC layers keep the dimension at $2\boldsymbol{B}$. The evaluated architectures and results are shown in Table 3.2. We select the models by starting from the baseline (denoted as A) from [36] and then removing or adding layers from the JSCC encoder and the JSCC decoder networks to explore the impact of depth on the overall performance.

For each model we trained the network according to the three-step strategy described in Section 3.3.4 and performed evaluation on the CUHK03 dataset at SNR = 0dB, $\boldsymbol{B} = 64$. We also show the mean squared error between the original feature vectors and their noisy reconstructions, after JSCC autoencoder pretraining $\boldsymbol{T}_2$. The results show that the differences between the models are marginal. Model D, which corresponds to the architecture presented in Fig. 3.4 and was used in the rest of the paper, performs slightly better than the others in terms of final retrieval performance. This model was selected also due to its low computational cost, as it consists of only 5 fully-connected layers in total. We also used PReLU as the activation for the model variant D, and observed that even though it achieves better MSE in step $\boldsymbol{T}_2$, it fails to provide a good generalization capabilities in the final step, as it overfits to the data. Please note that the model F, does not have the activation function, as the only layers in both the encoder, and the decoder are the last layers, therefore, as described above, the activation and BN are removed.

3.5 Conclusions

In this work, we have introduced the image retrieval problem over wireless channels in the context of the edge network, where wireless edge devices send queries of images over a bandwidth and power limited channel to an edge server that stores the image database, also called the gallery. We first introduced a digital approach, which is based on a novel retrieval-oriented deep image compression scheme, and applied it to feature vectors obtained from the feature encoder. Next, we proposed a JSCC-based scheme, where feature vectors are directly mapped to the channel symbols and decoded at the receiver. We showed the latter approach not only achieves a superior retrieval accuracy at a target channel SNR, but also provides graceful degradation with the test SNR when it does not match the training SNR. We further introduced JSCC FC, which is a simplified version of the proposed model and showed that decoding is necessary at the receiver to mitigate the effects of channel impairments. We also proposed a novel strategy for training our JSCC scheme, that can be adapted to other machine learning applications performed over noisy channels. Our strategy achieves superior performance for training the JSCC scheme. We have also performed an extensive ablation study of different architectures and training strategies and compared the alternatives under various performance measures for a wide range of different channel conditions. The results show the superiority of the proposed architecture and the joint training approach.

We note, that edge devices are usually not equipped with powerful hardware that allows for the real-time processing of images with DNNs, as assumed in this work. Therefore, the next chapter studies the problem of collaborative edge inference, where only the first few layers of DNN can be efficiently executed at the edge device, whereas the remaining part of the forward pass of the DNN has to be finished at the edge server, with significantly higher computational capabilities.

Chapter 4

Joint Device-Edge Inference over Wireless Links

In this chapter, we focus on edge inference systems, where an edge device cooperates with an edge server to perform a machine learning task over the wireless edge [124]. As opposed to the previous chapter, instead of the image retrieval task, we consider image classification tasks tackled by deep neural networks (DNNs). These tasks do not require access to large databases of objects to retrieve and in theory can be completed locally, at the edge device. However, while the previous chapter assumed sufficient computational resources available at the edge devices, these are not usually capable of performing such visual inference tasks locally due to the large computational complexity of running DNNs. The solution we consider in this work is splitting the DNNs into two parts [37, 125, 63]. The first part is deployed at the edge device and further pruned to meet the computational power constraints of the device, while the feature map extracted from the input image by the first part is transmitted through a wireless link to the edge server, which finishes the forward pass of the DNN and obtains the classification result. We build upon the recent success of deep joint source-channel coding (JSCC) algorithms and propose a trainable compression and transmission scheme that learns a direct mapping from the feature maps to channel input symbols. In the results section we show the superiority of the proposed approach compared to the existing schemes based on classical separation-based transmission and JSCC.

4.1 Introduction

The number of Internet of Things (IoT) devices reached 22 billion at the end of 2018 and is expected to grow up to 75 billion by the end of 2025 [126]. Currently, most of these devices act as wireless sensors that collect data and offload it to a cloud or edge server for processing. This creates a major bottleneck in many emerging IoT applications as communication consumes significant energy and introduces errors and latency.

In this work, we consider deep neural network (DNN) based inference at an edge device [124]. Due to limited computational power and memory, edge devices typically cannot perform all the computations required by a complex DNN architecture. For example, a single forward pass of the ResNet-152 [93] architecture requires 11×10^9 floating-point operations (FLOPs) for a 224×224 input image. This would take a few minutes on a simple edge device, which is usually limited to a few MFLOPs per second. We assume that an edge server is available to help the device to perform the inference task. In most current implementations, the edge device offloads all its data to the edge server, where a DNN of any complexity can be deployed. Note, however, that parts of the data that the device is sending may not be useful for the underlying task. An alternative approach is to preprocess the data on the edge device, within the available computational limits, and transmit only the resulting features to the edge server. We therefore encounter two main challenges: minimizing the number of computations that have to be done locally, and designing a robust communication scheme within the limited available transmission power and bandwidth.

To address the first challenge, DNN architectures that operate within the low-complexity constraints of mobile devices are proposed in [107]. These may still need hundreds of millions FLOPs to perform a single forward-pass, which may be unacceptable for certain edge devices. Some recent works [127, 125, 63] suggest splitting DNNs into two parts, where only the first few layers are implemented on the device within its computational constraints, while the remaining

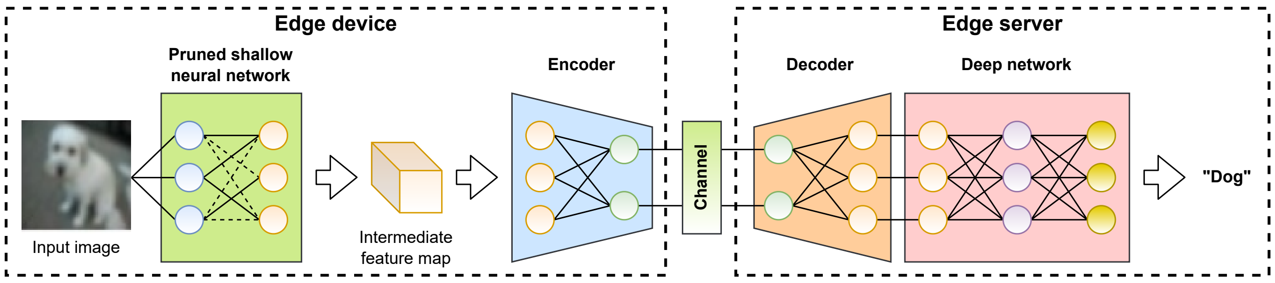


Figure 4.1: An overview of the proposed system. The baseline neural network is split between an edge device and an edge server. Given input data (e.g., an image), pruned sub-network performs the first part of the forward pass to generate an intermediate feature map, which is then compressed by DeepJSCC encoder and sent through a wireless link. At the receiver side, first the compressed feature map is reconstructed, and the remaining part of the forward pass is completed to obtain the final prediction.

layers are deployed on the edge server. However, this approach requires reliable transmission of the intermediate feature vectors to the edge server. To reduce the communication requirements, a typical approach is to quantize and/or compress the feature vectors before transmitting over the channel [127, 125]. These methods consider the amount of information (e.g., the number of bits) that must be conveyed to the edge server, but ignore the energy and latency cost of communications, and potential errors that may be introduced. Moreover, reliable transmission of the feature vectors requires an accurate estimate of the channel state at the edge device, and separate compression and channel coding is known to be suboptimal under strict delay constraints. Recently, a DNN-based DeepJSCC scheme has been shown to provide improved performance and robustness in wireless image transmission [33, 59, 52]. DeepJSCC scheme has been applied in distributed inference scenarios as well [128, 63, 36], but they require a significant number of on-device computations to run a forward pass of the underlying DNN.

In this work, we propose to reduce the on-device computational load by incorporating a pruning step in the network training. Network pruning [96] aims at reducing the computational redundancy within DNNs by efficiently removing certain neurons, convolutional filters, or entire layers based on a saliency measure or a regularization term. Therefore, given a certain channel condition and a computational constraint at the edge device, our goal is to find the optimal DNN splitting point and the pruning parameters to ensure the best possible accuracy as well as efficiency in a given scenario. We consider image classification as the target application.

Network splitting together with pruning for edge devices has recently been studied in [129], but similarly to other works, they ignore the errors that may be introduced over the channel. In contrast, our work is the first to study a joint device-edge inference architecture combined with pruning taking into account the noisy wireless channel. Contributions of the work presented in this chapter can be summarized as follows:

- We propose a DNN training procedure for joint device-edge inference systems under extreme power and latency constraints by combining novel pruning and splitting techniques with end-to-end feature transmission.
- Inspired by the DeepJSCC architecture [33], we propose an autoencoder-based network for intermediate feature map transmission to allow bandwidth reduction.
- We present extensive evaluations of the proposed approach at various DNN splitting points, channel SNRs, and bandwidth and computational power constraints, and show its efficacy in a wide range of settings.

4.2 Methods

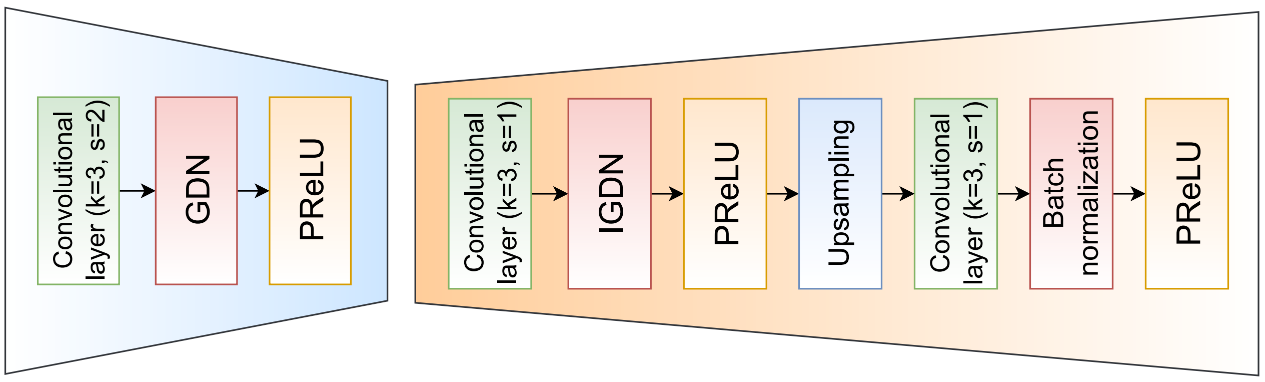
We propose a 4-step training strategy for combining partial network pruning with an end-to-end autoencoder architecture for transmitting intermediate feature maps of an arbitrary hidden layer of a DNN (Fig. 4.1). Such an approach allows for a reduction in the computations carried out at the edge device, while taking into account the effect of channel noise on the performance (within the specified bandwidth constraint). Most popular convolutional neural networks (CNNs), such as VGG [27] or ResNet [93] perform spatial dimensionality reduction of intermediate feature maps by applying pooling operations or convolutional filters with stride greater than 1. Nevertheless, as spatial dimension is being reduced, number of channels is usually expanded to extract the most significant features from the input image. Therefore, in the first few layers of such networks, the total dimension of the feature map increases up to a certain point, after which it starts to decrease due to further downsampling. As a consequence, there exists a hidden layer within such a network, where the size of the intermediate feature map is lower than that of the input; which acts as data compression for the underlying task. Transmitting such a feature map instead of the original input can help to reduce the bandwidth, but will impose significant computational resources on the device as it will have to preprocess the image through many layers of the network. This defines a trade-off between on-device computation and communication bandwidth, which we aim to optimize.

4.2.1 Channel model

We consider an additive white Gaussian noise (AWGN) channel, but any differentiable channel model can be employed instead. Specifically, given a channel input vector $\mathbf{x} \in \mathbb{R}^B$, where B represents the available channel bandwidth, the channel output $\mathbf{y} \in \mathbb{R}^B$ is given by $\mathbf{y} = \mathbf{x} + \mathbf{z}$, where $\mathbf{z}$ is an independent and identically distributed (i.i.d.) noise vector with elements $z \sim \mathcal{N}\left(0, \sigma^2\right)$. An average power constraint of $P = 1$ is imposed on the channel input vector, i.e., $\frac{1}{B}\sum_{i=1}^{B} x_i^2 \leq P$. We evaluate the accuracy for different channel signal-to-noise ratios (SNRs) given by $\frac{P}{\sigma^2}$. To compare our JSCC approach to digital methods we use standard Shannon capacity formula given by $C = \frac{1}{2}\log_2\left(1 + \frac{P}{\sigma^2}\right)$.

4.2.2 Classification baseline

Our framework is flexible, and can be easily adapted to any system that incorporates DNNs. We focus on image classification task as it is the most frequent approach to automatically analyse image content and generate its metadata. Given an image and a finite set of possible classes, the classification task aims at assigning the correct class label to the image. We experiment with VGG16 [27] with batch normalization (BN) added after each convolutional layer as it is one of the most popular networks employed for image classification. The network consists of 13 convolutional layers with stride 1 divided into 5 blocks, where each block is followed by a pooling operation. We consider each of the pooling operations as a potential network splitting point as it provides feature compression by construction, and does not affect the accuracy. After the


Convolutional layer (k=3, s=2)
GDN
PReLU
Convolutional layer (k=3, s=1)
IGDN
PReLU
Upsampling
Convolutional layer (k=3, s=1)
Batch normalization
PReLU

(a) Encoder (b) Decoder

Figure 4.2: Proposed encoder and decoder architecture for feature transmission. At the encoder, dimensionality reduction is performed by the convolutional layer. Shallow structure of the encoder reduces the computational load on the power-constrained device. Here, k represents the symmetric, 2-dimensional kernel size of the convolutional layers, and s represents their stride.

last pooling layer we also employ a fully-connected classifier consisting of three fully-connected layers, where the first two have the output size of 512 and the last one maps 512-dimensional vector to 100-dimensional class predictions.

4.2.3 Autoencoder architecture

In this work, we explicitly model and evaluate the impact of the noisy wireless channel on the performance. Therefore, we design the communication scheme in conjunction with the DNN architecture employed for the underlying classification task. As opposed to most of the literature on device-edge co-inference, we do not employ digital codes to transmit the feature maps, which are known to be suboptimal in finite blocklengths. Instead, we employ the autoencoder architecture shown in Fig. 4.2. Its asymmetrical structure is designed to reduce on-device computations; therefore, the encoder's architecture (Fig. 4.2a) consists of a single convolutional layer of stride 2×2 and 3×3 kernels, which perform both spatial and channel-wise compression of the feature map in a single step. The convolutional layer is followed by the generalized divisive normalization (GDN) layer [78], which is commonly used in most successful deep compression schemes such as [75] as a replacement of BN. GDN operation is defined as:

$$u_i^{(k+1)}(m,n) = \frac{w_i^{(k)}(m,n)}{\left(\beta_{k,i} + \sum_j \gamma_{k,ij} \left(w_j^{(k)}(m,n)\right)^2\right)^{\frac{1}{2}}}, \tag{4.1}$$

where $u_i^{(k)}(m,n)$ denotes the ith output channel of the kth stage of the encoder at the spatial location (m,n) and $w_i^{(k)}(m,n)$ denotes the corresponding input value. The approximate inverse operation, called IGDN is given by:

$$\hat{w}_i^{(k+1)}(m,n) = \hat{u}_i^{(k)}(m,n) \left(\hat{\beta}_{k,i} + \sum_j \hat{\gamma}_{k,ij} \left(\hat{u}_j^{(k)}(m,n)\right)^2\right)^{\frac{1}{2}}, \tag{4.2}$$

where $\hat{w}$ and $\hat{u}$ are the output and input of IGDN, respectively. Finally, we employ parametric rectified linear unit (PReLU) [130] as an activation function to further increase the learning capacity of our model. The output of the encoder network is directly transmitted over the channel (after normalization - to meet the power constraint).

At the decoder (Fig. 4.2b) we first perform a single convolution with stride 1×1 and 3×3 kernel size on the compressed feature map. This is followed by the IGDN operation, PReLU activation, and upsampling to restore the original spatial dimension of the intermediate feature map. Finally, another convolutional layer with the same stride and kernel size is applied to increase the feature map's depth to its original value, followed by BN and PReLU. Note that the number of channels effectively controls the size of the transmitted vector as our encoder always reduces the spatial dimensionality by a factor of 4. The only exception is the last block of VGG16 network (after pooling 5), where the feature map of size $1 \times 1 \times 512$ cannot be downsampled so we only control the number of channels.

4.2.4 Training strategy

Our training strategy consists of 4 steps. Firstly, we pretrain the VGG16 network with cross-entropy loss for 60 epochs, using SGD [131] optimizer with a learning rate of 0.01, momentum of 0.9, and L_2 penalty on network parameters weighted by $5 \cdot 10^{-4}$ to avoid overfitting. We reduce the learning rate by a factor of 0.1 after 20th and 40th epochs.

Next, we select the splitting point after one of the pooling layers of the network and employ network pruning. We use the pruning algorithm in [98], which uses Taylor expansion to approximate the change in the loss function induced by pruning. In principle, the algorithm evaluates the importance of each convolutional filter up to the splitting point, and removes the least significant ones. In our setup, the algorithm removes 512 convolutional filters at a single pruning iteration, followed by additional 10 training epochs with a learning rate of 0.0001 to recover the accuracy lost by the filter removal.

Afterwards, we run a forward pass through the pruned network with each image in the training set to extract all the possible feature maps at the splitting point. We use the feature maps as a new training set for our autoencoder, which we pretrain for 40 epochs with a learning rate of 0.1, momentum of 0.9, and L_2 penalty weighted by $5 \cdot 10^{-4}$. We use L_1-loss to recover the feature maps as close to their original versions as possible. This step is crucial to speed-up the convergence of the end-to-end training; since the feature maps are low-dimensional and autoencoder architecture is very simple, its execution is relatively fast. Starting from this step, we incorporate an AWGN channel model between the encoder and decoder to gain robustness against channel noise.

In the last step, we perform end-to-end training of the entire network. Specifically, we combine both parts of the VGG16 network and place pretrained autoencoder at the splitting point. Similarly to the first step, we train the network with cross-entropy loss, SGD optimizer with learning rate of 0.0001 and the other parameters unchanged.

4.3 Results

In this section we evaluate the performance of the proposed approach and compare it with other schemes in the literature.

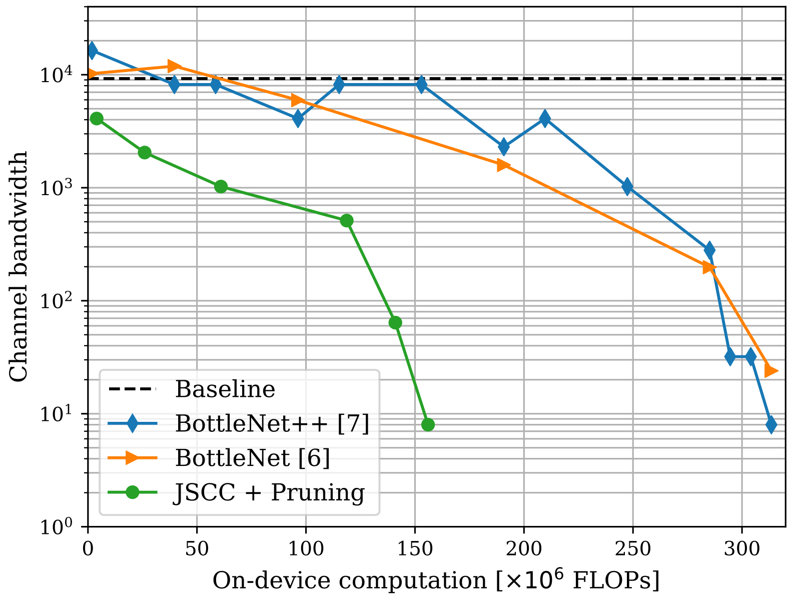
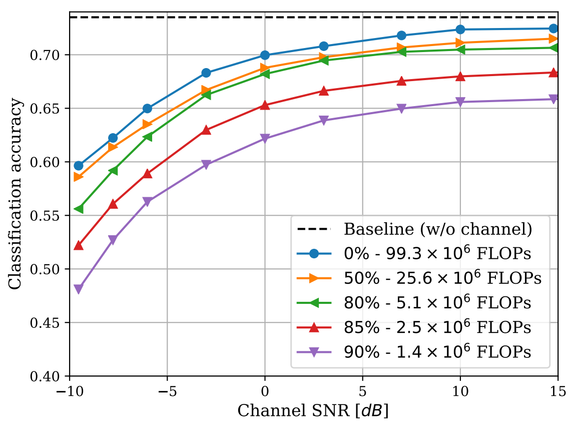


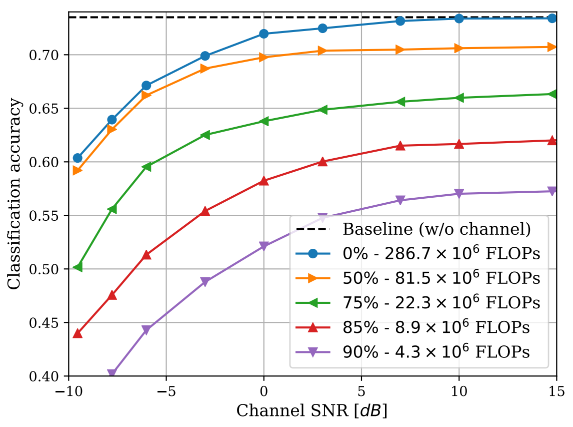
Figure 4.3: Required channel bandwidth as a function of the number of on-device computations to guarantee an accuracy level within 2% of the classification baseline.

4.3.1 Experimental setup

In order to evaluate the accuracy of the proposed method, we employ popular CIFAR100 dataset, which consists of 60000 RGB images divided into 100 different classes (e.g., bicycle, fox, oranges, etc.) [23]. Each class is represented evenly by 600 images of size 32×32 pixels, 500 for training and 100 for testing. During training, we first perform common data augmentation steps, namely we apply 4 pixel zero-padding at each side of an image and randomly crop 32×32 pixel tiles. Moreover, we randomly flip images horizontally with a probability of 50% and normalize them to have zero mean and unit variance. After such preprocessing, we perform multiple training runs of the proposed system, according to the strategy in Section 4.2.4 for different values of channel SNR, pruning ratios, network splits, and channel bandwidths, and evaluate the corresponding classification accuracy and required number of computations. In order to calculate the computational complexity of our approach, we count the number of FLOPs necessary to perform a single forward pass of the layers executed at the edge device (pruned shallow sub-network and the encoder).



(a) $B = 2048$, splitting after pooling 2



(b) $B = 128$, splitting after pooling 4

Figure 4.4: Classification accuracy as a function of the channel SNR for different pruning ratios and channel bandwidth fixed to B.

4.3.2 Channel bandwidth and on-device computation

In this section we select the models that minimize the channel bandwidth, which we define as the number of real symbols transmitted per image, and maximize the pruning ratio (which results in the minimal on-device computation), under a channel SNR of 14.5dB, allowing for a maximum drop of 2% in the classification accuracy compared to the baseline. Results in Fig. 4.3 clearly show that our proposed approach beats both the JSCC-based BottleNet++ [63] and the digital communication based BottleNet [125] schemes by a large margin. The proposed scheme requires only 4×10^6 FLOPs to achieve approximately 3× bandwidth reduction compared to the *baseline*, which we define as transmitting the original PNG image and performing classification on the edge server without any processing at the edge device.

Another superiority of our approach is that achieving 64× bandwidth reduction (from 512 symbols to 8 symbols) after the last pooling in VGG16 network is possible with only 156×10^6 FLOPs, which is only half of the operations necessary to run a single forward pass of the unpruned network. More importantly, given 156×10^6 FLOPs limitation, which is the number of operations that can be performed on a single Apple Watch device within $0.05s$, we achieve 1024× bandwidth reduction compared to [63].

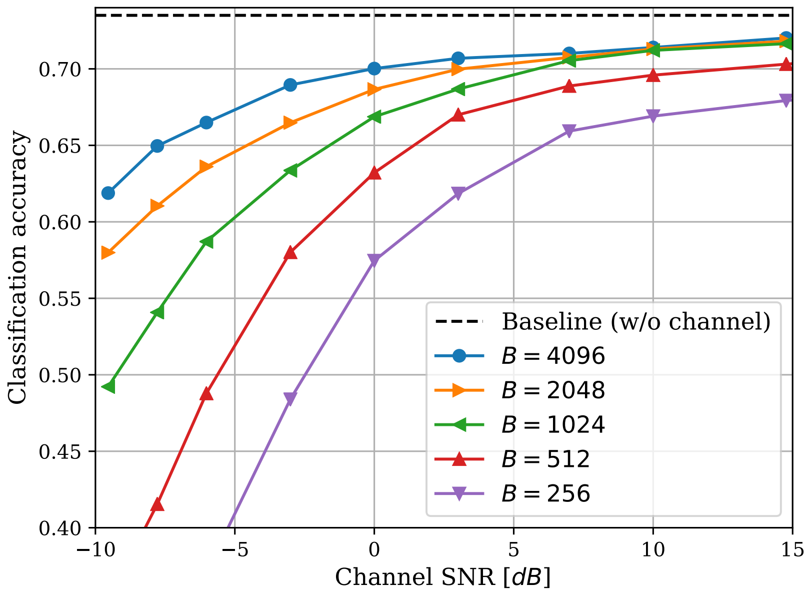


Figure 4.5: Classification accuracy as a function of the channel SNR for different channel bandwidths. Splitting after pooling 2 with a pruning ratio of 0.5 (24.3×10^6 FLOPs).

4.3.3 Comparison between different pruning ratios

In this section we evaluate the influence of different pruning ratios on classification accuracy under different channel SNRs for fixed splitting points and available channel bandwidths (Fig. 4.4a and Fig. 4.4b).

It is clear, that pruning leads to a drop in accuracy, as expected. Nevertheless, given reasonable pruning ratios of up to 50%, accuracy drop decreases as we approach very low values of channel SNR. This behaviour may stem from the fact that feature distortion caused by network pruning becomes less significant when the channel is very noisy. Another important observation is that very high pruning ratios do not seem to reduce nor improve the robustness of the communication scheme - the accuracy drop caused by reducing the channel SNR follows a similar trend for every pruning ratio considered in this experiment.

4.3.4 Comparison between different channel bandwidths

In our last experiment, we fix the pruning ratio and the splitting point and examine the influence of the available bandwidth on the classification accuracy under different channel SNR values (Fig. 4.5). One can clearly see that the available bandwidth is a crucial factor in the performance. In our experiments, reducing the bandwidth produced similar results for high SNR values. Nevertheless, the more limited the available bandwidth is, the sharper the drop in the accuracy with channel SNR.

4.3.5 Selecting the optimal splitting point

As demonstrated by the results, the selection of the optimal splitting point is a challenging and multi-dimensional problem. It is necessary to consider factors such as desired classification performance, the channel quality, available bandwidth, as well as the computational power of the edge device. For example, given the strict performance bounds in Fig. 4.3, where each point of the *JSCC + Pruning* curve corresponds to pruning layers up to pooling 1, 2, 3, 3 (higher pruning rate), 4, and 5, respectively, it is shown that changing the splitting point towards the end of the network decreases the bandwidth needed to send the information, but increases the required on-device computations, as more layers have to be processed by the edge device. Moreover, lowering the accuracy of the classification task allows to further reduce the computational requirements, or gain more robustness against channel noise.

4.4 Conclusions

We studied joint device-edge inference considering an edge device with limited computational resources, and a wireless channel to the network edge. In particular, we considered image classification over a power and bandwidth limited edge device. We proposed pruning of the baseline taking into account the noise introduced over the channel, under a constraint on the available bandwidth. Our approach achieves superior results in classification accuracy even with extremely limited computational and communication resources.

We note, that in the presence of extremely adverse channel conditions, the quality of the intermediate feature maps delivered to the edge server may be insufficient for a successful classification. The next chapter explores the possibility of obtaining class predictions locally on the edge device through the application of an early exit mechanism. Such a mechanism can be utilized in case of extremely bad channel conditions, or in case the input samples are easy enough to classify locally, without the need for utilizing the full capabilities of a DNN applied to the task.

Chapter 5

Evaluation of Early Exits for Device-Edge Collaborative Inference over the Wireless Edge

This chapter extends the work presented in Chapter 4 by introducing an early exit mechanism [132, 133]. This mechanism allows to obtain a solution to a DL task early, without the need to execute each layer of the DNN. Early exits can be implemented in the context of joint device-edge collaborative inference systems in order to obtain a rough prediction and avoid the transmission of the intermediate feature maps to the edge server, thus providing significant savings of communication resources. This is different from the system studied in Chapter 4, where the intermediate features had to be transmitted to the edge server in order to finish the forward pass of a DNN and obtain the final predictions. We note, that the predictions obtained early through the early exit may lead to decreased accuracy compared to utilizing the final exit only. However, in the context of edge inference systems, it might sometimes be beneficial to keep the prediction provided by the early exit, as the edge server might perform even worse given highly adverse channel conditions. This chapter focuses on comparing various decision-making (DM) mechanisms [134, 135], which, given the output of the early exit and the current channel state information, provide a binary decision on whether to transmit the intermediate feature map to the edge server or to keep the prediction provided by the early exit.

5.1 Introduction

The rise of the fifth-generation (5G) standard for broadband cellular networks drives the transformation of how we understand connectivity and apply it in our everyday lives. The wide availability of reliable wireless connectivity allows many new technologies to be delivered closer to end users. This includes video streaming, augmented reality (AR), various Internet of Things (IoT) applications, among others.

One of the most emerging application areas, where the users can significantly benefit from the improved connectivity is edge machine learning (ML). The majority of edge devices are unable to run complex ML algorithms, such as deep neural networks (DNNs), due to either computational power or memory limitations. Currently, edge devices offload their data to a powerful *edge server* through a wireless link. The edge server can then run the desired ML algorithm on the delivered data, and return the inference result to the device.

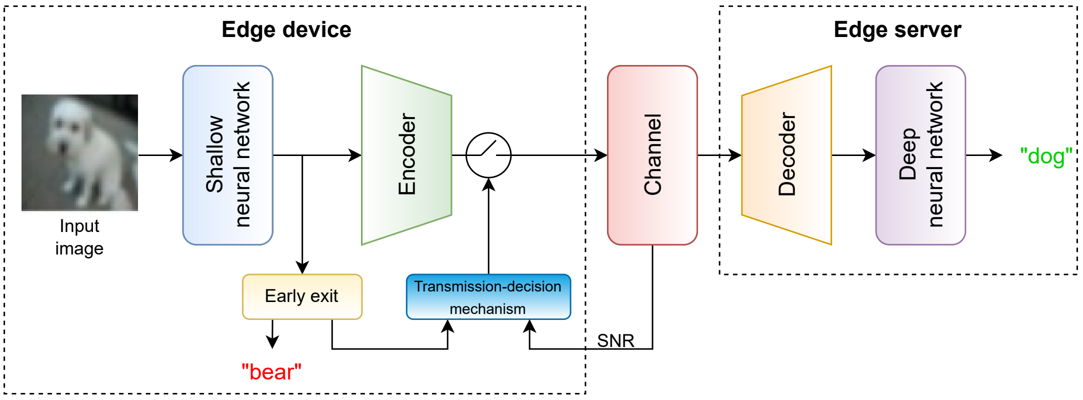


Figure 5.1: System model. The edge device performs a forward pass of a shallow NN. Early exit attempts to make a prediction based on the output of NN, and passes its result to a TD mechanism, which combines this result with the current channel state to decide whether to transmit the output of the shallow NN to the edge server, where additional processing is performed by a DNN to reach a more refined prediction.

Collaborative inference is a recent extension to the edge ML paradigm [136, 9]. In collaborative inference, a complex ML algorithm is split into two parts between the edge device and the edge server. The edge device utilizes its limited computational power to run the initial part of the algorithm, i.e., the first layers of the DNN, and offloads the intermediate result to the edge server, which runs the remainder of the algorithm, and provides the final result to the edge device. Deep neural networks are particularly suitable for collaborative inference thanks to their sequential multi-layered structure, which provides many potential splitting points. An important benefit of executing an initial part of the DNN on the edge device is that DNNs trained for a specific task learn to only preserve the information essential for that task. Therefore, each layer of a DNN reduces the amount of information contained in the intermediate results, denoted as intermediate feature maps. As a result, executing even a few layers of a DNN on the edge device can significantly reduce the amount of information to be transmitted to the edge server. Such collaborative inference systems for the wireless edge have been extensively studied in [37, 125, 63, 62, 137].

In the case of constantly changing wireless channel conditions, it may not always be the best choice to offload the intermediate feature maps to the edge server for inference. When the channel conditions are poor, the delivered features may become too distorted to allow for successful inference at the edge server. Moreover, certain input samples may require less processing than others, making edge processing unnecessary. In this chapter, we study the application of early exits in the context of wireless collaborative inference at the edge. The idea of early exits, initially proposed in [133], is to attach additional outputs to a DNN, which allow obtaining earlier predictions without the need to execute all the layers of the DNN. Originally, early exits were proposed as a solution to vanishing gradient problems. Since then, they have been extensively studied for efficient ML as a method for avoiding excessive computations [138, 139, 132, 140]. We note that, in the collaborative inference problem, early exits can allow the edge device to obtain an inference result locally, without the need to transmit the intermediate features to the edge server, which has been noticed in [141], while [142] introduced a scheme for dynamic early exiting based on classification confidence thresholds. The benefits of such an approach are communication savings, reduced computations at the edge server, and better inference performance in the presence of a low channel quality between the device and the edge server.

We consider a collaborative inference system (see Fig. 5.1), where an edge device can decide whether it should rely on its own inference following an early exit, or transmit the obtained features to the edge server. Optimizing such a decision policy is challenging as it has to rely solely on the information available at the edge device, i.e., the available features and channel state information (CSI), and with the lowest possible computational cost. We propose an automated mechanism for making this decision, based on a lightweight neural network, which analyzes the data available from the early exit combined with the CSI. To the best of our knowledge, this is the first work to analyze early exits in the context of wireless edge collaborative inference systems, and one of the first (similar NNs for accepting early exit output were proposed in [134, 135] in the context of efficient DNN inference) to study a DL-based decision mechanism for accepting early exit inference results. We provide a systematic comparison of such mechanisms.

5.2 Methods

5.2.1 System model

The collaborative inference system studied in this chapter consists of a DNN for image classification, which is split into two parts (see Fig. 5.1). The first part is deployed at the edge device, and the second at the edge server. The intermediate features produced by the first part of the DNN are compressed by a joint source-channel coding (JSCC) encoder neural network, which maps the intermediate features directly into channel input symbols, and follows the architecture proposed in [37]. These symbols are then transmitted through a wireless channel to the edge server, where they are reconstructed by a JSCC decoder neural network and further processed by the remaining part of the image classification DNN. Early exit layers $\boldsymbol{F_e}$ are added at the splitting point of the DNN, in order to obtain early predictions for the underlying task. We note, that obtaining correct prediction through early exit allows the edge device to avoid transmitting intermediate features to the receiver, thus saving communication resources. We introduce a transmission-decision (TD) mechanism for obtaining a decision on whether the edge device should stop at the early exit prediction, or transmit the intermediate features to the edge server for further processing. The goal of a TD mechanism is to maximize the classification accuracy as well as *communication savings*, defined as a fraction of images classified at the edge device by the early exit, without transmitting the intermediate feature maps to the edge server for further processing. For example, transmission savings of 0.4 indicate that 40% of images were classified locally, while the remaining 60% required transmitting features to the edge server. Alongside several simple TD mechanisms, we design a transmission-decision NN (TD NN) to learn the correct decision policy based on the output of the early exit and the channel state information. We note that this problem is multi-faceted, as one has to consider not only the expected performances of early and final exits, but also the cost of transmitting the intermediate features, as well as their quality at the receiver, given the instantaneous quality of the wireless channel between the edge device and the edge server.

5.2.2 Channel model

We consider additive white Gaussian noise (AWGN) channel between the edge device and the edge server, modeled as $\mathbf{y} = \mathbf{x} + \mathbf{z}$, where $\mathbf{x} \in \mathbb{R}^B$ is the channel input vector and $\mathbf{z}$ is the additive noise vector with each component independently drawn from the zero-mean Gaussian distribution with variance σ^2. Here, B denotes the available channel bandwidth to transmit the intermediate features to the edge server. An average power constraint is imposed such that $P = \frac{1}{B}\sum_{i=1}^{B} x_i^2 \leq 1$; thus, the channel signal-to-noise ratio (SNR) can be calculated as $\text{SNR} = \frac{P}{\sigma^2}$.

5.2.3 Training strategy

The training strategy applied to train each part of the proposed system is as follows. In the first step, we initialize the DNN and the early exit layers. The DNN is trained with the sum of the cross-entropy losses between the ground truth classification labels and the predictions of both early exit F_e at the edge and final exit F_f at the server. Please note that splitting of the DNN is not yet performed at this step, as we pre-train the entire DNN for the image classification task first. In the second step, we split the DNN, and introduce a JSCC autoencoder to communicate over the wireless channel model at the splitting point. We then train the resulting architecture by first freezing the image classification DNN and training only the JSCC autoencoder. Once the JSCC autoencoder converges, we unfreeze the DNN and train the whole architecture end-to-end with the same loss function as in the previous step. Finally, we introduce the TD NN and train it until convergence with each of the strategies detailed in Section 5.2.4. Alongside the TD NN, we study alternative TD mechanisms, which do not require training.

5.2.4 Transmission-decision neural network (TD NN)

The TD NN $D : \mathbb{R}^n \rightarrow [0, 1]$, where n is the number of features input to the TD NN. This number may vary based on the selection of the TD NN inputs, which may include outputs of the early exit, statistics of these outputs, or channel SNR. The choice of these inputs is analyzed in Section 5.3.3. TD NN is trained to make a decision on whether the edge device should transmit the intermediate features to the edge server for further processing, or output the prediction made by the early exit. Overall, defining a proper training criterion for the TD NN is a non-trivial task. On one hand, the goal is to maximize the classification accuracy; on the other hand, the system should also minimize the communication costs, that is, should transmit only when the expected performance gain given by the final exit is sufficiently large. To account for all these criteria, we consider multiple training objectives for the TD NN and carefully study their impact on the final performance in terms of the trade-off between accuracy and communication savings. The final decision on whether to transmit the features or keep the early exit predictions should be a binary choice; therefore, during inference, we apply rounding operation to the TD NN outputs. The architecture of the TD NN is relatively simple, consisting of 3 fully-connected (FC) layers with ReLU activations and 256-dimensional hidden features, as shown in Fig. 5.2.

Joint early-final cross-entropy criterion. The first training criterion we consider uses the weighted combination of the predictions made by the early exit and the final exits of the DNN, where the TD NN output is used as the weights. Given the decision network outputs $D(x)$, the early exit outputs $F_e(x)$, and the final exit outputs $F_f(x)$, we define the joint early-final exit

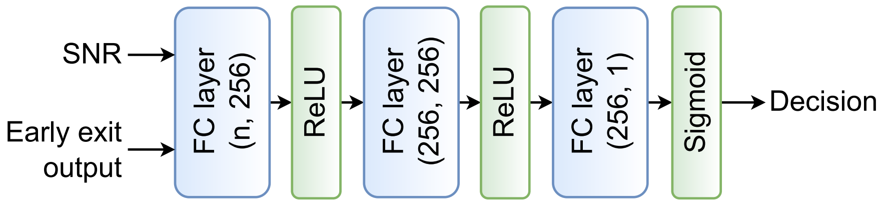


Figure 5.2: The architecture of the TD NN. The DNN considers the outputs of the early exit and the CSI to obtain the decision on whether to accept the early exit predictions or transmit the intermediate feature maps to the edge server for further processing. The numbers in parenthesis indicate the number of input and output features for each FC layer.

prediction as follows:

$$F(x) \triangleq D(x) \cdot F_e(x) + (1 - D(x)) \cdot F_f(x), \tag{5.1}$$

where x is the input sample. Then, the loss function is given by:

$$L_{joint} = L_{ce}(F(x), y_{gt}) + \beta \cdot |1 - D(x)|, \tag{5.2}$$

where L_{ce} is the cross-entropy loss, y_{gt} is the classification ground truth one-hot vector for the input sample x, and β is a weighting parameter used to balance the accuracy with the communication cost. Note that, in (5.2), the first term is responsible for the accuracy of the joint early-final exit prediction, while the second is a penalty term that is aimed at prioritizing early exit and avoiding transmission.

We note that the decision is a binary choice, i.e., $D(x) : \mathbb{R}^n \rightarrow \{0, 1\}$, while the outputs of the TD NN can be any value between 0 and 1. Therefore, during the training we calculate the decision as $D(x) = \frac{1}{1+e^{-T \cdot \hat{D}(x)}}$, where $\hat{D}(x)$ are the raw outputs of the TD NNs, and T is the temperature parameter used to push the decisions away from ambiguous values close to 0.5.

Binary cross-entropy criterion. The second criterion used for training TD NN is based on the assumption that the optimal decision is to transmit intermediate features only when the early exit prediction is likely to be incorrect, while the final exit prediction is likely to be correct. Based on this assumption, we generate a set of ground truth decisions d_{gt}, such that $d_{gt} = 1$ if and only if the early exit is incorrect and the final exit is correct, and $d_{gt} = 0$

otherwise. We then train the TD NN with the following loss function:

$$L_{gt} = L_{bce}(D(x), d_{gt}) + \beta \cdot |1 - D(x)|, \tag{5.3}$$

where L_{bce} is the binary cross-entropy function, and the second term is for penalizing excessive transmissions as before.

Mixed criterion. The final criterion we use to train the TD NN is a mixed approach. It includes a combination of the two aforementioned criteria to obtain the following loss function:

$$L_{mixed} = L_{ce}(F(x), y_{gt}) + \alpha L_{bce}(D(x), d_{gt}) + \beta \cdot |1 - D(x)|. \tag{5.4}$$

5.3 Results

5.3.1 Experimental setup

We evaluate the proposed approach on the CIFAR100 dataset [23], which consists of 60000 images of 100 distinct classes. Each class is represented with 500 samples in the training set and 100 samples in the test set. We employ the VGG16 architecture [27]. To avoid excessive computations on the edge device, the early exit consists of an average pooling operation, a FC layer with ReLU activation, followed by another FC layer with softmax activation, which outputs 100 class predictions.

The training of the DNN follows the strategy presented in Section 5.2.3. We utilize the stochastic gradient descent (SGD) optimizer with a batch size of 128, and run 90, 30, and 30 epochs for each training step, respectively. In the first training step, we set the learning rate to 0.1, and decay it by a factor of 10 every 30 epochs. In the second and third training steps, we also start from a learning rate of 0.1, but the decay occurs every 10 epochs. At each training iteration, starting from the second training step, we set the channel bandwidth B to 64 channel uses and determine the channel SNR according to the "sandwich rule" [143], that is, we set

SNR = -10 dB in the first iteration, SNR = 10 dB in the second iteration, and, in the third iteration, SNR = $\mathcal{U}(-10\,\text{dB}, 10\,\text{dB})$, where $\mathcal{U}(\cdot,\cdot)$ represents the uniform distribution. This cycle is then repeated throughout the whole training. This ensures that the system learns to handle diverse channel conditions by training a single set of DNNs exposed to a large SNR variability throughout training. An alternative approach would be to train a separate set of DNNs for each channel SNR, which imposes a significant memory overhead and is not considered in this chapter. For the experiments presented in this section, we placed the splitting point after the third pooling layer of the VGG16 architecture.

TD mechanisms. In addition to NN-based approaches explained in Section 5.2.4, we consider the following alternatives:

- We define the *confidence* of the early exit classifier as the maximum value within the softmax output of the classifier. We transmit the features to the edge server only if the confidence is lower than a predefined threshold. We denote this approach as *Confidence threshold*. This mechanism is closely related to the method proposed in [142], which also utilizes confidence thresholds for making the decision.

- We first calculate the entropy of the softmax output of the classifier as:

$$H(x) = -F_e(x) \bullet \log_2(F_e(x)), \tag{5.5}$$

  where $\bullet$ indicates the dot product operation. We then define an entropy threshold and transmit only if the entropy for a given sample is lower than the threshold. A similar criterion has been utilized in [138]. We denote this approach as *Entropy threshold*.

- We first use the training split of the dataset to calculate the average expected accuracy and communication savings for each class under different confidence thresholds, as defined above, and multiple values of SNR. During inference, we find the set of per-class thresholds that maximizes the objective criterion defined as a weighted sum of expected accuracy and communication savings. We denote this approach as *Per-class confidence threshold*.

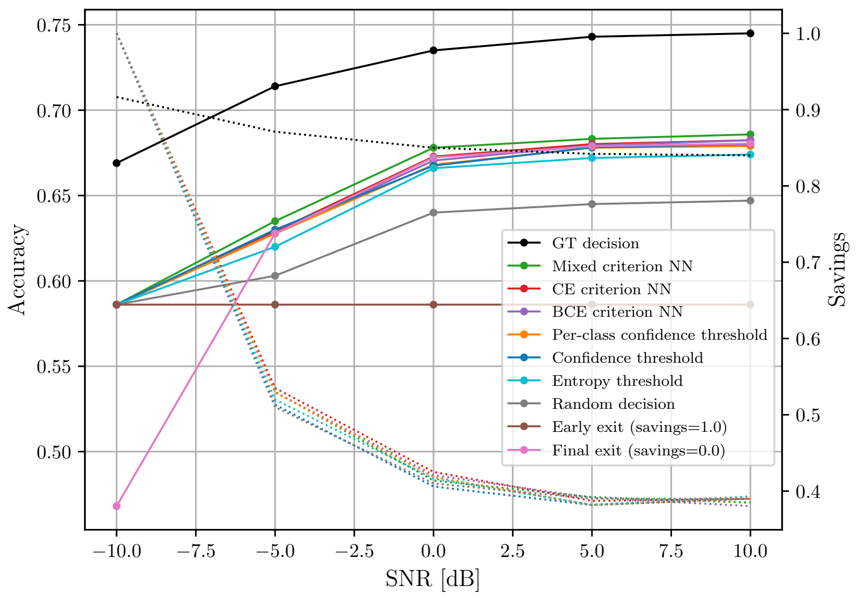


Figure 5.3: Comparison of the accuracy (solid lines) achieved by different models for comparable communication savings (dotted lines) as a function of channel SNR.

- *Early exit*, which corresponds to a scheme, where the features are never transmitted to the edge server, and only the predictions made by the early exit are utilized. This corresponds to a case, where $D(x) = 1$. The opposite of this scheme is *Final exit*, which assumes the features are always transmitted to the edge server, and only the predictions made by the final exit are considered, which corresponds to $D(x) = 0$.
- *Random decision*, where the transmission of each sample is decided randomly using a uniform distribution.

5.3.2 Performance comparison

In this section, we compare the classification accuracy achieved by the TD methods described in Section 5.3.1 and the TD NN trained with each of the strategies described in Section 5.2.4.

In Fig. 5.3, we plot the accuracy achieved by each of the methods as a function of the channel SNR. We tune the hyperparameters of each method to achieve similar communication savings for a given SNR. The *Mixed criterion NN* approach, which is based on a TD NN trained with the loss function in Eq. (5.4), for $\alpha = 0.1$, slightly outperforms other approaches, most of which

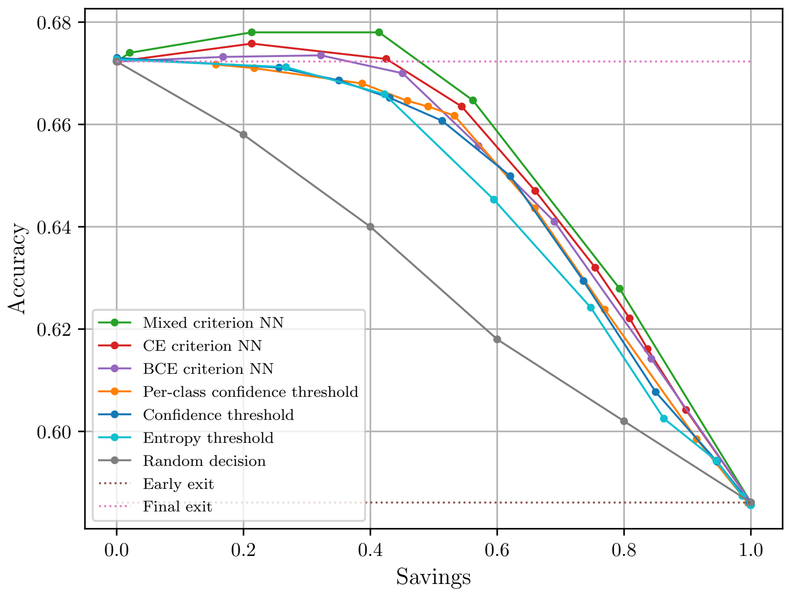


Figure 5.4: Comparison of the accuracy (solid lines) achieved by different models for fixed channel SNR = 0 dB as a function of communication savings.

achieve similar results (which will be further analyzed in Section 5.3.3), and the entropy threshold approach performs slightly worse than the confidence-based approaches. More importantly, significant communication savings (almost 45% for SNR = 0 dB) can be achieved by applying the majority of TD mechanisms studied in this chapter. We note that proper utilization of these mechanisms leads to achieving better or on-par performance compared to using only early exit or final exit, which validates the need for implementing dynamic early exiting mechanisms in the context of edge collaborative inference systems. Another observation is that the random decision, as expected, achieved significantly lower accuracy than the other approaches. The *GT decision* approach, which refers to the ground truth, and only transmits when the early exit is incorrect and the final exit is correct, serves as an upper bound. It consistently allows more than 85% savings at an accuracy significantly higher than the other approaches. One can observe that savings decrease as the SNR increases since less noise leads to better accuracy at the final exit, making it more beneficial to transmit the intermediate features to the edge server. The TD mechanisms studied in this work can take that information into consideration, and produce decisions that can accurately assess whether additional processing at the edge server is required.

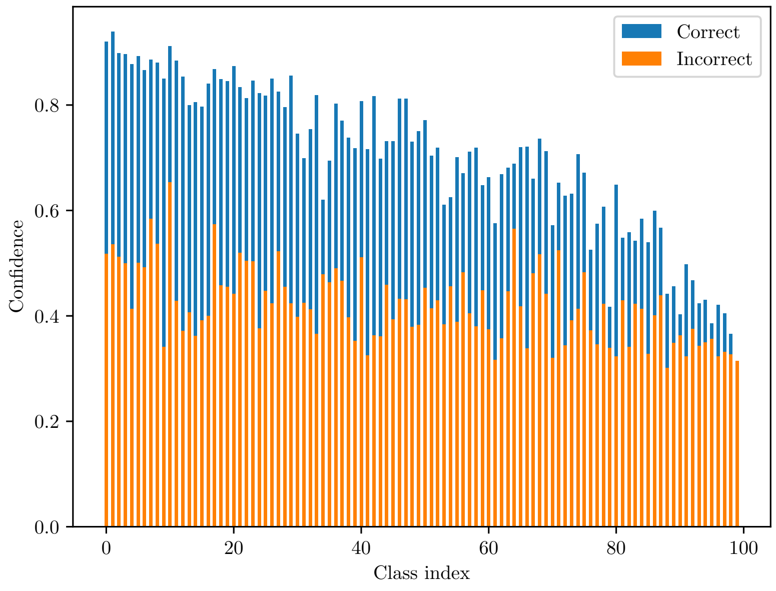


Figure 5.5: Average per-class confidences of the early exit predictions for the correct and incorrect predictions for all the classes in the CIFAR100 dataset. In general, the confidences are higher for the correct predictions; however, the gap may differ significantly between classes.

In Fig. 5.4 we first fixed SNR to 0 dB and varied the hyperparameters of each of the methods to achieve different values of communication savings. Again, slightly improved accuracy was achieved by Mixed criterion NN, closely followed by *CE criterion NN*, where the TD NN was trained with the loss function defined in Eq. (5.2). The other methods achieved similar accuracy for comparable communication savings levels. Early exit and Final exit are visualized as dotted lines on the figure (Early exit savings are always equal to 1.0). We can see that at savings of approximately 0.4, the Mixed scheme is able to outperform Final exit by 0.6% in accuracy, which proves that avoiding the transmission and utilizing a less powerful early exit can bring communication savings and provide better quality predictions. Another important observation is the fact that the Mixed approach NN is effectively a combination of the CE criterion NN and *BCE criterion NN* approaches, yet it outperforms both of them. We hypothesize that the reason for this behavior is that the CE criterion NN effectively learns to put more weight on the output that is more likely to produce the correct prediction in given circumstances. A small addition of the binary cross-entropy term from the BCE criterion NN further avoids transmission when both early and final exits are not likely to produce a correct prediction. This leads to further communication savings without imposing any accuracy loss.

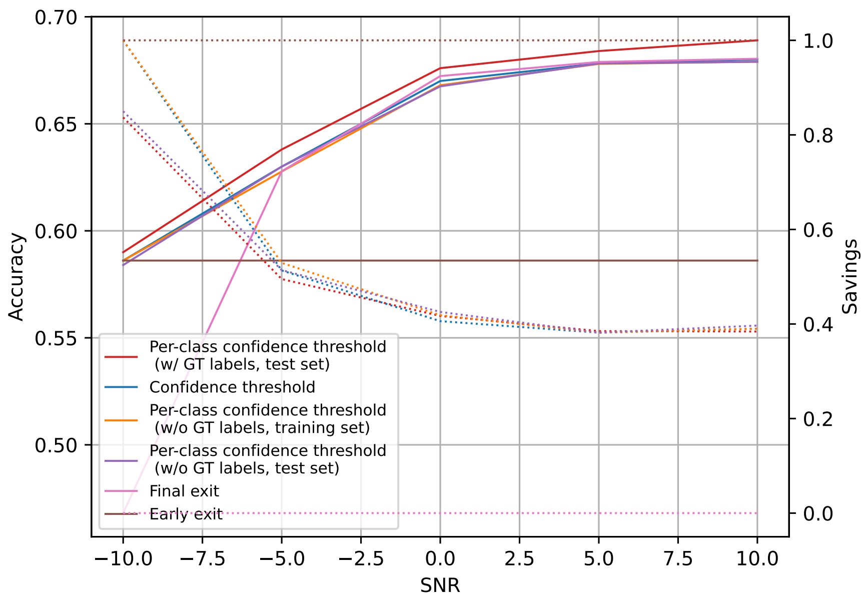


Figure 5.6: Comparison of the accuracy (solid line) and savings (dotted line) of the methods based on early exit predictions confidence. Hyperparameters of each method were tuned to provide comparable savings at each SNR.

5.3.3 Ablation study

Confidence analysis. In this section we analyze the distribution of the per-class confidences of the early exit outputs separately for the correct and incorrect predictions (please see Fig. 5.5). We note that the confidences of the correct predictions are generally higher than the confidences of the false predictions, yet the exact ratio between the two may vary significantly from as high as 2 : 1 to 1 : 1 for different classes. This leads to the conclusion that information about the classes has to be taken into consideration when making the decision. Surprisingly, when analyzing Fig. 5.3, one can quickly notice, that the Per-class confidence threshold and Confidence threshold schemes exhibit significant overlap, despite the latter utilizing only a single threshold value for all the classes. The reason for that behavior will be analyzed next.

Performance of the confidence threshold methods. In Fig. 5.6 we compare the accuracy achieved by various confidence-based threshold methods for different values of channel SNR and comparable savings. We note that the accuracy of the method that sets separate thresholds for each class, based on the training set performance, is similar to that achieved by the method that applies a single threshold to all the classes. To further investigate the reason for this behavior,

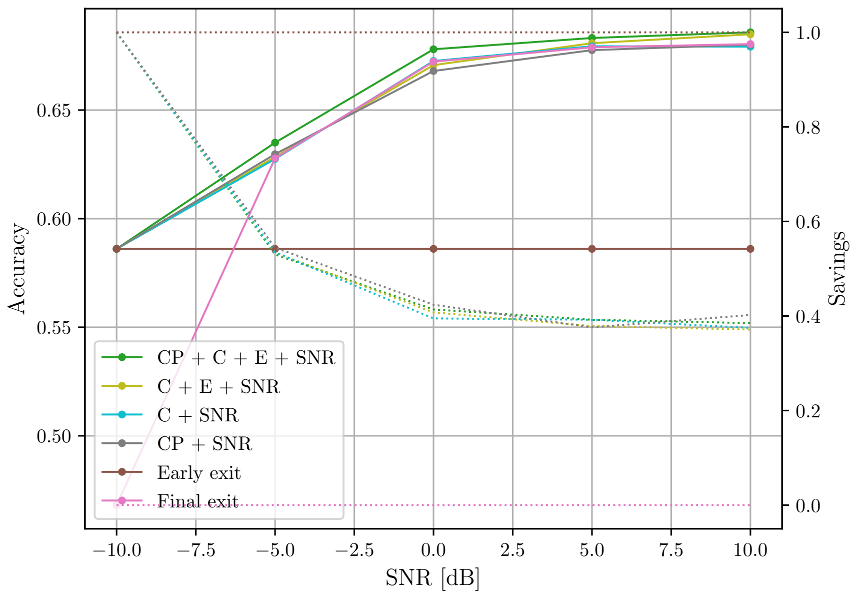


Figure 5.7: Comparison of the accuracy (solid line) and savings (dotted line) of the Mixed criterion NN, given different inputs to the TD NN, as a function of SNR. Hyperparameters of each method were tuned to provide comparable savings for each SNR. Early exit class probabilities (CP), confidences (C), entropy (E) of the class predictions, and SNR were used as inputs. The best-performing combination includes all of the inputs.

we obtained a new set of per-class confidence thresholds. As opposed to the original method, the new thresholds were found based on the test set of the CIFAR100 dataset (indicated as *test set* in the label). We further noted that the class with the highest softmax score assigned by the early exit is not necessarily the correct one. Therefore, for the sake of comparison, we consider another scheme, which selects the proper per-class threshold based on the ground truth label. This ensures the correct per-class threshold is selected each time during inference. We see that only in the case of utilizing the test set to calculate the thresholds and ground truth labels to select the correct threshold during inference, the per-class confidence method was able to achieve improvement over the single threshold method. This indicates that using the training set to find optimal thresholds, as well as the error incurred by selecting thresholds based on the most confident class may mislead the TD mechanism and cause a significant accuracy drop of the per-class confidence threshold method.

Optimal TD NN inputs. The above analysis (Fig. 5.5) about the necessity of considering the entire distribution of class predictions rather than some high-level statistical information about

Table 5.1: Complexity of the TD NN compared to the other parts of the DNN.

Model	Complexity
TD NN	0.094 MFLOPs
Early exit classifier head	0.025 MFLOPs
DNN part at the edge device	97.990 MFLOPs
Full DNN	320.717 MFLOPs

it has been experimentally validated. In Fig. 5.7 we show the comparison of the accuracy of the models, where the TD NN was employed and trained with the Mixed criterion. The architecture of the TD NN and other hyperparameters remain unchanged, and we only modify the inputs to the network to find their impact on the accuracy. We see from the results that providing only the channel SNR and the class probability (CP) distribution of the early exit is not sufficient to achieve satisfactory performance. The behavior is similar if we consider SNR and the statistics that summarize the CP distribution, including confidence (C) and entropy (E). The best result is achieved by combining all the inputs, i.e., CP distribution, C, E, and SNR. We note that the information about the entropy and the confidence is already implicitly included in the CP distribution, but we hypothesize, that the TD NN operates best when both high- and low-level information is provided to it as input.

Complexity analysis. In Table 5.1 we compare the complexity of the TD NN to the other parts of the DNN. We note that in the scenario studied in this chapter, i.e., for VGG16 DNN with a partitioning point selected after the third average pooling operation, the complexity of TD NN is small. When we compare the complexity of the part of the DNN deployed at the edge device with the complexity of the TD NN, we can see that the latter is responsible for approximately 0.01% of the floating-point operations (FLOPs) performed at the edge device. We further argue, that the computational burden of 94000 FLOPs imposed by the TD NN, is acceptable given the modern hardware standards.

5.4 Conclusions

In this chapter, we presented an early exit mechanism for collaborative inference systems deployed at the wireless edge, where an image classification DNN is split into two parts and deployed between the edge device and the edge server. We introduced a TD NN, which utilizes the information provided by the early exit mechanism and the channel state to make a binary decision indicating whether the early exit output should be accepted, or the data should be offloaded to the edge server for further processing. We showed that, through careful design of the TD mechanism, either learnable or not, it is possible to achieve significant communication savings, while also outperforming strategies based on utilizing only early exit or final exit, yet the differences between the accuracy achieved by various TD mechanisms are not significant. Through further experimentation, we also illustrated that the entire early exit output distribution, combined with summarizing statistics should be used as an input to the TD mechanism in order to achieve satisfactory performance.

The systems studied in this chapter and chapters 3 and 4 assume uploading the features from edge devices to edge servers for obtaining the results of image classification or image retrieval task. We note, however, that the wireless uplink channels are usually significantly more constrained than the downlink channels. Moreover, transmitting users' data to a centralized entity for processing induces privacy constraints. Therefore, in the next chapter, we consider a scenario, where a DNN trained for a specific task requested by the user is transmitted from the edge server to the edge device, and the user can utilize the trained network locally on their own data, without incurring any privacy risks.

Chapter 6

Neural Network Transmission over the Air

This chapter considers the problem of transmitting deep neural network (DNNs) parameters over wireless channels [38, 144, 145]. We note, that chapters 3, 4, and 5 mainly focused on the uplink transmission from the edge device to the edge server. However, in practice, such uplink channels are heavily constrained, therefore it is usually a preferred choice to utilize the downlink connection instead [146]. As opposed to the previous chapters, here we do not consider transmitting intermediate features obtained from the DNNs. Instead, we focus on transmitting the parameters of a DNN to an edge device, which can use the DNN to perform inference on its own data. The proposed scheme maps the DNN parameters directly into channel input symbols. The delivered parameters are not the perfect reconstruction of the original DNN parameters, therefore, the proposed approach first trains the DNN such that it is robust to channel imperfections expected during the transmission. This includes exploiting methods such as noise injection [147] and knowledge distillation [148]. To further improve the protection of the most significant DNN parameters, the unequal error protection scheme is proposed, which first compresses the parameters and then expands their size with Shannon-Kotelnikov mappings [149], depending on their importance measured by Hessian-based characteristics [150]. Finally, to achieve better SNR adaptivity, an interpolation scheme is implemented by collectively training a family of DNNs [151], each targeting a single SNR. The performance of all the proposed schemes is extensively evaluated in the experimental section.

6.1 Introduction

In recent years, deep learning (DL) has been shown to provide very promising solutions to many practical tasks within computer vision, natural language processing, robotics, autonomous driving, communications, and other fields. Developments within the area of DL have been made possible mainly thanks to the rapid growth of the computational power and memory available for both the researchers and the potential users of various DL-based algorithms. This resulted in the development of increasingly complex deep neural networks (DNNs) with millions and even billions of parameters trained on massive datasets, achieving impressive accuracy and performance in a wide variety of applications. On the other hand, the memory required to store a single modern DNN model can easily go from a few megabytes up to hundreds of gigabytes.

We typically evaluate the performance of a DNN architecture with the accuracy it achieves on new samples. This assumes the availability of the model at the end user. However, given the increasing prominence of DNNs employed for a large number and variety of tasks, we cannot expect every user to have all possible DNN parameters always available locally. Moreover, even a locally available model may need to be updated occasionally, either because the model at the server has been improved in the meantime through further training, or the task at hand has changed, e.g., due to variations in the statistics or size of the samples, or the system requirements. An alternative may be for the user to send its data samples to an edge server, where an up-to-date model is available for inference [125, 63, 37, 58]. However, in many scenarios, the user may not want to send its data due to privacy constraints. Also, the user may want to infer many samples, which may create increased traffic. Moreover, the uplink capacity may be limited compared to the downlink. In such scenarios, a reasonable solution is to transmit the DNN parameters to the user over the network, rather than the user sending

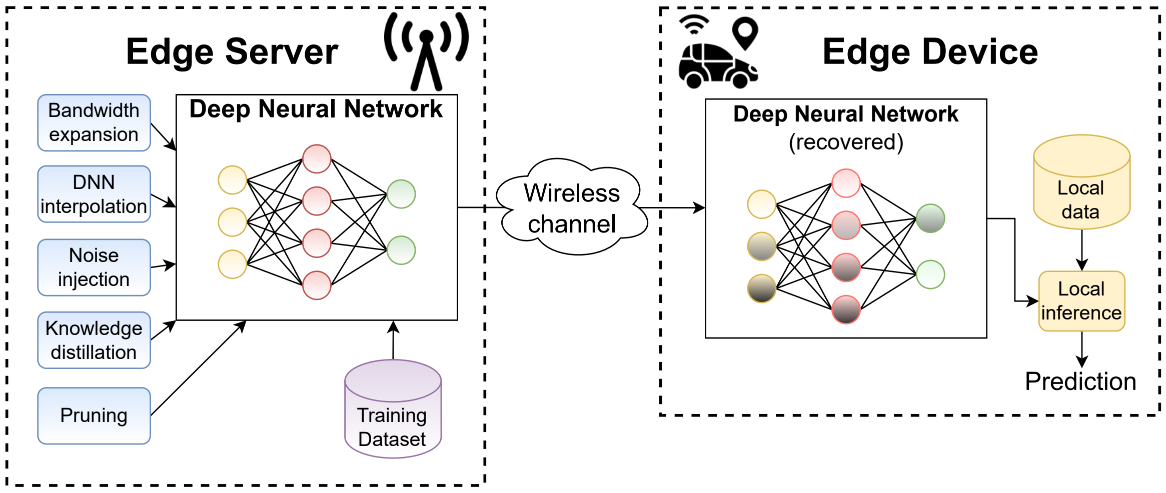


Figure 6.1: System model. In AirNet, DNN is transmitted over a wireless channel in an uncoded fashion, and it employs various training techniques aimed at bandwidth reduction and enabling robustness against channel noise.

the data samples to the edge server. However, given the growing size of modern DNNs, and stringent latency requirements of edge intelligence applications, the transmission of the DNN parameters to an edge user may be infeasible. This problem can be further amplified in the future by the adoption of very specialized DNNs, that either solve very specific tasks adapted to a specific geographic location, or are frequently updated due to the non-stationarity of the underlying tasks. In such scenarios, it is necessary to develop methods, which allow for fast and reliable delivery of DNN parameters over the wireless channel.

A fundamental ambition of the sixth generation (6G) of mobile wireless networks will be to enable seamless and ambient edge intelligence. Therefore, it is expected that the efficient storage and delivery of DNN parameters will constitute a significant amount of traffic. Indeed, model distribution and sharing for machine learning applications at the edge is already being considered by 3GPP as part of the next generation of mobile networks [152]. Our goal in this chapter is to develop efficient DNN delivery techniques at the wireless network edge, such that the highest performance can be achieved by the user despite wireless channel imperfections.

We consider the system model illustrated in Fig. 6.1, where an *edge server*, e.g., a base station (BS), should enable an *edge device*, e.g., a mobile phone, an autonomous car, a drone, a medical device, to carry out inference on local data samples. We impose a strict latency constraint as well as the usual resource constraints on the wireless channel between the server and device. The edge device reconstructs a local model to be used for inference on local samples.

Consider, for example, vision- or LIDAR-aided channel estimation or beam selection, where an autonomous car aims at establishing a high-rate millimeter wave connection with a BS in the non-line-of-sight setting, based on the input from its cameras or LIDAR sensors [153, 154]. The best approach would be to provide the car with a DNN, optimized specifically for the coverage area of the particular BS. However, we cannot expect each car to store DNNs trained for every possible cell area that it may go through. Instead, it is much more reasonable to assume that locally-optimized DNNs would be delivered to vehicles as they move around, and come into a coverage area of a particular cell. Another important application of model delivery is federated/distributed learning over wireless channels [9, 155]. In such problems, a locally trained/updated model is shared with a parameter server or neighboring devices at each iteration of the training process, and highly efficient delivery is essential considering that training a large DNN model can require thousands of iterations. The same would hold for many other DNN-aided edge applications that may require localized optimization of DNN parameters, e.g., various localization services. On the other hand, sending even a relatively simple VGG16 [27] network requires transmission of roughly 15×10^6 32-bit floating-point parameters. Assuming a standard LTE connection at a channel signal-to-noise ratio (SNR) of 5dB, and capacity-achieving channel codes, such a transmission would require roughly 30 seconds to complete, which is unacceptable for most time-sensitive edge applications.

In this work, we consider two fundamental approaches to this problem [38]. As in most wireless lossy source delivery problems, the two approaches follow the separation approach and the joint source-channel coding (JSCC) approach, respectively. In the separation-based approach, we train a model of a certain size, whose parameters are then transmitted over the channel using an error correction code to provide reliability in the presence of noise and fading. We can either train a sufficiently small-size network that can be delivered over the channel, or a pre-trained network can be compressed to be reduced to the required size. In the alternative JSCC approach, the parameters of the network are transmitted directly over the channel. In the latter approach, called AirNet, the training is carried out taking into account the effect of channel imperfections. We remark that, in both approaches, we assume the availability of the data at the edge server, which allows to optimize the DNN in terms of accuracy, size, and robustness. The problem we face can be treated as a lossy source delivery problem over a wireless channel; more specifically a remote source delivery problem, where the goal is to deliver the underlying true inference function, i.e., the model, to the edge user. But, the edge server does not have access to the true model, and it can only estimate it through its local dataset.

Our major contributions can be summarized as follows:

- We introduce a novel communication problem that requires the delivery of machine learning models over wireless links under strict bandwidth and transmission power constraints for reliable inference at the receiver.
- We propose a novel JSCC approach to this problem, called AirNet, that can achieve reliable edge inference at very low channel SNR and bandwidth, and is robust to channel variations, as opposed to separation-based techniques, which break down abruptly when the channel SNR cannot support the adopted channel code rate.
- We use network pruning to meet the channel bandwidth constraint, and knowledge distillation (KD) to increase the accuracy of inference at the receiver. To further increase AirNet's robustness to adverse wireless channel conditions, we employ noise injection during training and carefully study its effect on performance.
- In order to provide unequal error protection (UEP) to different network layers, we employ bandwidth expansion; that is, we prune the network to a size smaller than the available bandwidth, and expand some of the layers to provide extra protection against channel noise. We choose the layers to be expanded by their *sensitivity*, measured through the Hessian matrix. We show that UEP through bandwidth expansion provides significant gains in terms of the final accuracy at the receiver.
- Above algorithms are trained for a specific channel SNR to obtain the best accuracy. This would require training and storing a different set of network parameters for different channel conditions, which is not practical. To resolve this critical limitation, we propose

an ensemble learning approach, where we obtain a spectrum of networks simultaneously for a whole range of channel SNRs.

- We present extensive evaluations of AirNet, including different datasets, channel models, training and pruning strategies, channel conditions, and power allocation methods. We show that the proposed AirNet architecture and training strategies achieve superior accuracy compared to separation-based methods, which employ DNN compression followed by separate channel coding. AirNet allows for a significant reduction in bandwidth requirements while sustaining satisfactory levels of accuracy for the delivered DNNs.

The remainder of this chapter is organized as follows. We present relevant works on DNN compression and JSCC literature in Section 6.2. In Section 6.3, we present our system model, followed by the introduction of the AirNet architecture in Section 6.4 with all the details regarding training, pruning, and noise injection methods used. Section 6.5 presents bandwidth expansion methods, alongside an UEP scheme, and Section 6.6 introduces an ensemble learning scheme, which trains a spectrum of networks aimed at different values of channels SNRs. This is followed by Section 6.7, which evaluates AirNet on various datasets, channel conditions and choices of training parameters. Finally, Section 6.8 concludes the chapter.

6.2 Related Work

6.2.1 Network quantization

Alongside pruning, which has been described in detail in Chapter 2, another method of reducing the complexity of DNNs is quantization. Instead of using a full, 32-bit precision for storing the network weights and activations, low-bit precision can be used, resulting in significant gains in terms of both the computations and the memory footprint. Many works have studied network quantization in recent years [156, 157, 150, 158, 159, 160, 161]. These works study different aspects of quantization, e.g., evaluation of the sensitivity of the DNN parameters, training strategies that benefit quantization, etc. Authors of [150] estimate the statistics of the

Hessian matrix corresponding to each layer of the network, in order to derive a layer-dependent sensitivity metric for a mixed-precision quantization process. In the DeepCABAC method [144], quantized DNN parameters are further compressed by utilizing context-adaptive binary arithmetic coding.

Analog storage of network parameters is studied in [162], where the authors also consider applying channel noise to DNN parameters during training, pruning, and KD. We note that despite some techniques can be effectively used for both analog storage and wireless transmission of DNNs parameters, there are many differences. The fundamental difference between these two applications is that for analog storage, the channel noise variance does not change with time, but rather with the magnitude of the stored value. For wireless transmission, however, we have to ensure that the network adapts well to a variety of channel SNRs. To this end, our approach requires either storage of multiple DNNs, each trained for a specific value of channel SNR, or a training method, which ensures that the network can adapt to various SNRs. Another novelty of our work is that we take into consideration the sensitivity of each layer with respect to channel noise, and assign more channel bandwidth to the most sensitive layers.

6.3 System Model

We consider an edge server, which has access to a labeled dataset that can be used to train a machine learning model. However, this trained model is to be employed at an edge device to predict labels of new data samples. The edge server is capable of training a model locally, but it is connected to the edge device through a bandwidth and power-limited noisy channel. The goal is to minimize the performance loss, measured by the accuracy of the model at the receiver end, due to the channel imperfections, while meeting the prescribed bandwidth and power constraints at the transmitter. This is different from conventional channel coding problems, which aim at minimizing the probability of error, or the conventional JSCC problems, which minimize an additive distortion measure defined on source samples. The optimal performance of such separation-based or JSCC approaches is usually achieved under the presence of multiple samples from the source distribution, which allows generating a model of a source used for compression and reconstruction with minimal error/distortion. In our case, only a few, or a single DNN is stored at the edge server, which cannot obtain such a source model. AirNet is still a JSCC problem, but with an unconventional distortion measure, that requires not only training but also delivery of a model over a noisy channel with good generalizability properties.

While the above formulation is general enough to be applied to any learning model, given their state-of-the-art performance and large size that require significant communication resources, we focus on DNNs. More specifically, we consider a DNN with parameters $\mathbf{w} \in \mathbb{R}^d$, trained at the edge server. The DNN is then transmitted to the edge device over the wireless channel. The specific channel models used in our work are described in Section 6.3.1. After the transmission, the edge device reconstruct another network $\tilde{\mathbf{w}} \in \mathbb{R}^d$ based on the signal it receives over the channel. This network is then employed at the device to obtain predictions $p_{\tilde{\mathbf{w}}} = f_{\tilde{\mathbf{w}}}(I)$, where I is a sample from the edge device's local dataset, and $f_{\tilde{\mathbf{w}}}(\cdot)$ represents the forward pass of the DNN parameterized by the decoded weights $\tilde{\mathbf{w}}$.

6.3.1 Channel model

We model the channel between the edge device and the edge server as an additive white Gaussian noise (AWGN) channel. We consider static as well as slow fading channels. For the static AWGN channel, we have $\mathbf{y} = \mathbf{x} + \mathbf{z}$, where $\mathbf{x} \in \mathbb{C}^b$ is the channel input with the channel bandwidth b, defined as the number of channel uses, $\mathbf{y} \in \mathbb{C}^b$ is the channel output, and $\mathbf{z} \in \mathbb{C}^b$ is a vector containing independent and identically distributed (i.i.d.) noise samples drawn from circularly-symmetric complex Gaussian distribution $\mathcal{CN}(0, \sigma^2)$ with variance σ^2. An average power constraint is imposed on the channel input, i.e., $\frac{1}{b}\sum_{i=1}^{b} \|x_i\|^2 \le P$. We set $P = 1$ without loss of generality, which corresponds to an SNR of $\mathrm{SNR} = 10\log_{10}\left(\frac{1}{\sigma^2}\right)$.

In the slow fading scenario, the channel output $\mathbf{y} \in \mathbb{C}^b$ is given by $\mathbf{y} = h\mathbf{x} + \mathbf{z}$, where $h \sim \mathcal{CN}(0, \sigma_h^2)$ is the channel gain. We assume that the channel remains constant for the duration of a block of b channel symbols, but takes i.i.d. values drawn from $\mathcal{CN}(0, \sigma_h^2)$ across different blocks. As in the static channel scenario, we impose the average power constraint of $P = 1$, and the average SNR is given by $\mathrm{SNR} = 10\log_{10}\left(\frac{\sigma_h^2}{\sigma^2}\right)$. In all our experiments we set σ_h to 1. We also assume perfect channel state information (CSI) to be available at the receiver, thus, to get rid of the multiplicative component h, the receiver scales received signal $\mathbf{y}$ by $\frac{h^*}{\|h\|}$. The resulting signal is given by $\mathbf{x} + \frac{h^*\mathbf{z}}{\|h\|^2}$, which is equivalent to AWGN channel with a random SNR value. Therefore, in the case of communication over fading channels, we need to come up with a transmission scheme that will perform well over a range of SNRs with a given average variance.

6.4 AirNet: JSCC of DNN Parameters for Reliable Edge Inference

The conventional approach to the problem presented above would be to train a DNN of limited size, and quantize, compress and deliver its parameters over the channel using state-of-the-art channel codes. While we will also consider this "separation-based" approach as a baseline, our main contribution is a JSCC approach, where the DNN parameters are directly mapped to channel inputs. Next, we present the details of this approach.

6.4.1 Training strategy

Our performance measure is the average accuracy of the model reconstructed at the edge device on new samples. Here, the randomness stems from not only the random and previously unseen samples encountered at the receiver, but also from the channel noise and fading.

We first train a DNN with the data available at the edge server. In the proposed AirNet approach, each DNN parameter will be mapped to a channel input symbol. This has two consequences: first, the number of DNN parameters that can be delivered is limited by the channel bandwidth b; and second, transmitted parameters are received with random noise at the receiver. To increase network's robustness against noise we inject a certain amount of noise to the network's weights during training. While we initially train a large DNN with more than b parameters, in the second training step, we prune them by removing redundant parameters in order to satisfy the bandwidth constraint. Alongside pruning and noise injection, we also apply KD to prevent accuracy drop due to pruning. We provide the details of each training step in the rest of this section.

6.4.2 Pruning

To reduce the bandwidth required to transmit the network parameters we apply a simple pruning strategy [97]. Pruning removes redundant parameters while maintaining satisfactory performance, which effectively reduces the bandwidth requirement. In this work, we apply repetitive pruning and fine-tuning steps. At each pruning iteration, we remove a fraction of parameters that have the lowest l_1-norm. In order to avoid the transmission of meta-data containing the network's structure after pruning, we only consider structured pruning, which removes either entire convolutional filters or entire neurons, depending on the layer type. During fine-tuning, we simply re-train the network to recover the performance lost due to pruning. We additionally apply noise injection and KD to further increase robustness to channel noise and reduce the performance loss imposed by pruning.

6.4.3 Noise injection

Noise injection has been originally proposed as a regularization method to prevent overfitting in DNNs [147]. In this work, however, we utilize noise injection as a method to increase the DNN's robustness to channel noise. We hypothesize that, if the network experiences a certain amount of noise injected to its weights during training, it will effectively learn to achieve good performance even after its weights are transmitted over a noisy channel. At each iteration of the training, we apply the same noise components as imposed by the channel on the current network parameters, and calculate the loss function using these noisy network parameters. Please note we only inject the noise during network training to mimic the channel noise experienced by the network during the transmission.

6.4.4 Knowledge distillation (KD)

KD has been proposed as an effective method to boost up the performance of various DNN models trained for classification task [148]. In KD, a large DNN, called the *teacher*, which achieves high accuracy in the task, distills some knowledge into a smaller DNN, called the *student*. The loss function in KD is defined as a sum of two terms:

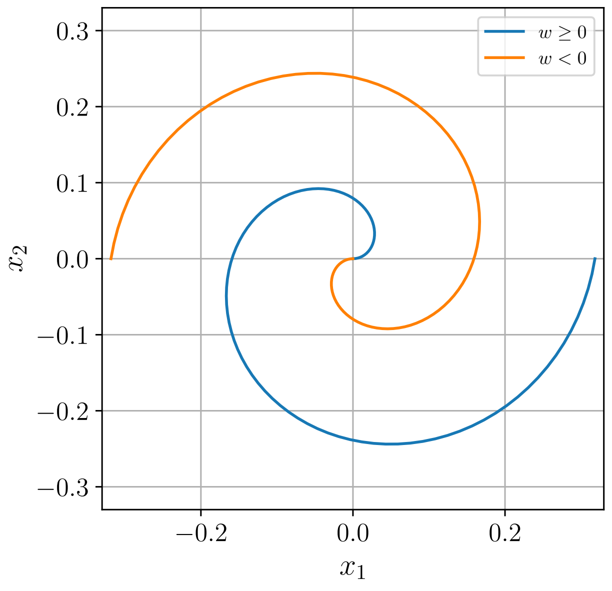
$$L_{total} = -t^2 \sum_{i}^{N} \hat{p}_i \log p_i - \sum_{i}^{N} \bar{p}_i \log p_i, \text{ where } \hat{p}_i \triangleq \frac{e^{\frac{\tilde{p}_i}{t}}}{\sum_j^N e^{\frac{\tilde{p}_j}{t}}}. \tag{6.1}$$

In Eq. (6.1), the first term is responsible for distilling the knowledge between the teacher and student, and the second is a standard cross-entropy loss, where $\hat{p}_i$ represents the softmax predictions of the teacher model, t is the temperature parameter, set to 2 in all our experiments, $\bar{p}_i$ denote the ground truth, and p_i are the student's predictions.

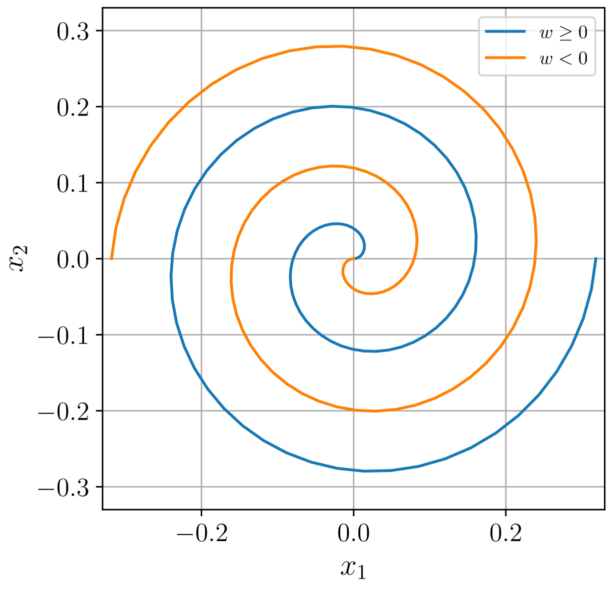
6.5 AirNet with Unequal Error Protection (UEP)

The network trained with noise injection has a certain level of robustness against channel noise. However, the performance will degrade inevitably as the SNR decreases. Here, we first propose trading-off the pruned network size with robustness against noise. The main idea is to use bandwidth expansion to better protect the DNN parameters against noise. For example, instead of pruning the network down to b parameters, we can prune it to, say, $b/2$ parameters, and use two channel symbols to transmit each network parameter. We consider two bandwidth expansion methods: Shannon-Kotelnikov (SK) mapping [149] and simple *layer repetition*.

SK mapping has been successfully used in JSCC in [163], where the authors use Archimedes' spiral as a codebook, and show its benefits for both bandwidth compression and expansion tasks. In this work, we employ a similar approach. Specifically, we map the DNN parameters onto Archimedes' spirals defined as:



(a) $\gamma = 2\pi$



(b) $\gamma = 4\pi$

Figure 6.2: Examples of Archimedes' spirals used in this work for the SK mapping scheme. Parameter γ controls the robustness of the DNN parameters against channel noise.

$$x_1 = \frac{\Delta}{\pi} w \cos(\gamma w), \; x_2 = \frac{\Delta}{\pi} w \sin(\gamma w), w \geq 0 \tag{6.2}$$

$$x_1 = -\frac{\Delta}{\pi} w \cos(-\gamma w + \pi), \; x_2 = -\frac{\Delta}{\pi} w \sin(-\gamma w + \pi), w < 0, \tag{6.3}$$

where Δ is a scaling factor, which we set to 1, γ controls the length of the spirals without changing the radius of the disc occupied by the spiral, and w is a DNN parameter.

Spirals generated by different γ parameters are shown in Fig. 6.2. Each network parameter $w \in \mathbb{R}$ is mapped to a point (x_1, x_2) on the spiral. Sign of the parameter is encoded by mapping positive-valued DNN parameters to the spiral defined by Eq. (6.2), and the negative-valued ones to the spiral defined by Eq. (6.3). We note that parameter γ can effectively control the length of the spirals, and the distance between the negative and positive spirals, which impacts the robustness of this coding technique. At low SNR, a high γ value may lead to the two spirals being too close to each other, resulting in sign errors in decoding. However, at high SNR, one can increase γ in order to better allocate the 2D space spanned by the spirals.

After the transmission, we decode the original parameters by mapping the received symbols (y_1, y_2) back to the values of network parameters by finding the nearest point on either of the spirals, as shown below:

$$\hat{w} = \pm\underset{w}{\mathrm{argmin}} \left((y_1 - \theta(w))^2 + (y_2 - \theta(w))^2 \right), \tag{6.4}$$

where $\theta(\cdot)$ represents the union of the spirals defined by Eq. (6.2), and (6.3).

We note that the SK expansion as defined above only allows a $1:2$ bandwidth expansion ratio. In order to achieve higher orders of expansion, one may consider re-applying the same expansion to x_1 and x_2, by simply replacing w in Eq. (6.2) and Eq. (6.3) by x_1 and x_2. With this, we can implement a bandwidth ratio of $1:2^n$, where n is the number of expansion steps applied. In order to achieve more flexibility in the overall expansion rates, we propose two methods, which allow to achieve intermediate expansion levels by applying different expansion rates to each layer of the network, depending on the available bandwidth. For a detailed explanation of the methods, please see Section 6.5.1.

An alternative, much simpler method is *layer repetition*. In this method, we simply transmit each network parameter multiple times and average the outputs at the receiver in order to reduce the variance of the noise component. We note that channel repetition can effectively achieve rates of expansion of $1:n$; thus, it is inherently more flexible than SK expansion; however, it does not exploit the higher-dimensional space as effectively as SK expansion, which leads to a sub-optimal performance as we will observe in Section 6.7.

6.5.1 UEP of DNN parameters against channel noise

The main limitation of the methods explained above is that they only consider uniform expansion of the entire network. In this section, we aim at providing methods that achieve intermediate levels of network expansion, and better accommodate the available bandwidth. It is known that the impact of different DNN layers on the overall performance varies [150]. Therefore, to exploit this inhomogeneity, we will look at methods that apply different expansion ratios to each layer of a DNN. A proper selection of the layers that should be expanded is extremely important in order to achieve satisfactory performance, thus a sensitivity metric is

Algorithm 1: Von Mises iteration for calculating the largest eigenvalue of the Hessian matrix associated with layer $\mathbf{w}_i$.

```
Input: $i$-th layer $\mathbf{w}_i$ of a DNN $\mathbf{w}$, training dataset $\mathcal{D}$.
Calculate the loss $L = \sum_{I_j \in \mathcal{D}} l(\mathbf{w}, I_j)$;
Calculate the gradient $\mathbf{g}_i = \frac{\partial L}{\partial \mathbf{w}_i}$ of the loss w.r.t. $\mathbf{w}_i$;
Draw a random vector $\mathbf{v}_i$ of the same dimension as $\mathbf{w}_i$;
Normalize $\mathbf{v}_i$: $\mathbf{v}_i \leftarrow \frac{\mathbf{v}_i}{\|\mathbf{v}_i\|_2}$.
repeat
    Calculate inner product $\mathbf{g}_i^T \mathbf{v}_i$;
    Calculate the Hessian and $\mathbf{v}_i$ product by $H_i \mathbf{v}_i = \frac{\partial(\mathbf{g}_i^T \mathbf{v}_i)}{\partial \mathbf{w}_i}$;
    $\mathbf{v}_i^{(prev)} \leftarrow \mathbf{v}_i$;
    Update $\mathbf{v}_i$: $\mathbf{v}_i \leftarrow \frac{H_i \mathbf{v}_i}{\|H_i \mathbf{v}_i\|_2}$;
    Calculate $\lambda_i = \frac{\mathbf{v}_i^T H_i \mathbf{v}_i}{\mathbf{v}_i^T \mathbf{v}_i}$.
until $|\mathbf{v}_i - \mathbf{v}_i^{(prev)}| < \epsilon$;
Output: Largest eigenvalue $\lambda_i$ of the Hessian matrix associated with layer $\mathbf{w}_i$.
```

necessary to specify which layers should be protected more than the others.

The first sensitivity metric we propose is based on the variation in the loss function imposed by injecting a certain amount of noise into a DNN, one layer at a time. We hypothesize that the layers that lead to a higher increase in the loss function when perturbed are more sensitive, and hence, should be protected more. Let $\tilde{\mathbf{w}}^{(i)}$ denote the network $\mathbf{w}$ with certain amount of noise injected into its i-th layer. Our sensitivity measure is based on the squared difference between the loss function $l(\mathbf{w}, I)$ of the original DNN and the same network when the i-th layer is perturbed: $l(\tilde{\mathbf{w}}^{(i)}, I)$. The sensitivity of the i-th layer can be expressed as:

$$s_1^{(i)} = \sum_{I_j \in \mathcal{D}} \left(l(\mathbf{w}, I_j) - l(\tilde{\mathbf{w}}^{(i)}, I_j) \right)^2, \tag{6.5}$$

where $\mathcal{D}$ denotes the training set available at the edge server.

Next, we consider a Hessian-based sensitivity metric [150], where the largest eigenvalue λ_i of the Hessian matrix associated with a given layer i is treated as the sensitivity metric of this layer. Since the computation of the Hessian matrix is not feasible for large DNNs, we follow the Von Mises iteration [164] to estimate it, as shown in Algorithm 1. Once the eigenvalue λ_i is calculated for the i-th layer, we simply set its sensitivity as $s_2^{(i)} = \lambda_i$.

Expanding the layer with the highest sensitivity may not always result in the best performance. This is because some layers contain more parameters than others, and expanding them requires more bandwidth than some less sensitive, yet already compact layers. Similarly, some layers may have higher variance than others, and expanding them may lead to a significant increase in the average power. To better allocate the available power and bandwidth resources, we normalize each sensitivity parameter by the total energy of the corresponding layer, $\tilde{s}_j^{(i)} \triangleq s_j^{(i)}/\|\mathbf{w}_i\|_2^2, j = 1, 2$. Once the sensitivities are calculated for all the layers, we iteratively expand the layers with the highest sensitivity until the available bandwidth is exhausted, as shown in Algorithm 2.

Algorithm 2: Uneven bandwidth expansion based on layer sensitivities.

```
Input: Layers $\mathbf{w}_i \in \mathbb{R}^{d_i}$ of a DNN and their sensitivities $s^{(i)}$, bandwidth $b$.
Initialize $r_i = 1, \ \forall_i$.
while $\sum_i r_i d_i \leq b$ do
    Calculate normalized sensitivities $\tilde{s}^{(i)} = \frac{s^{(i)}}{r_i \|\mathbf{w}_i\|_2^2}$;
    Find $i^* = \underset{i}{\text{argmax}} \ \tilde{s}^{(i)}$.
    $r_i \leftarrow r_i + 1$ (or $r_i \leftarrow 2r_i$ if SK expansion is considered).
end
Output: Number of repetitions $\{r_i\}$ for the selected sensitivity metric.
```

6.6 SNR Robustness

So far, training has been done targeting a particular channel SNR. With this approach, AirNet performs best if the mismatch between training and test SNRs is minimal. However, such a solution is not practical as it requires the edge server to store multiple sets of DNN parameters, each trained for a specific SNR value. In this section, we will present two methods to significantly reduce memory requirements.

In the first scheme, we train a single set of DNN parameters to be used over a range of SNR values from the interval $[\text{SNR}_{min}, \text{SNR}_{max}]$. Instead of sampling a noise vector from a fixed SNR target, we consider a different noise variance at every training iteration. SNR values over the iterations are chosen according to the "sandwich rule", which is frequently used in the efficient

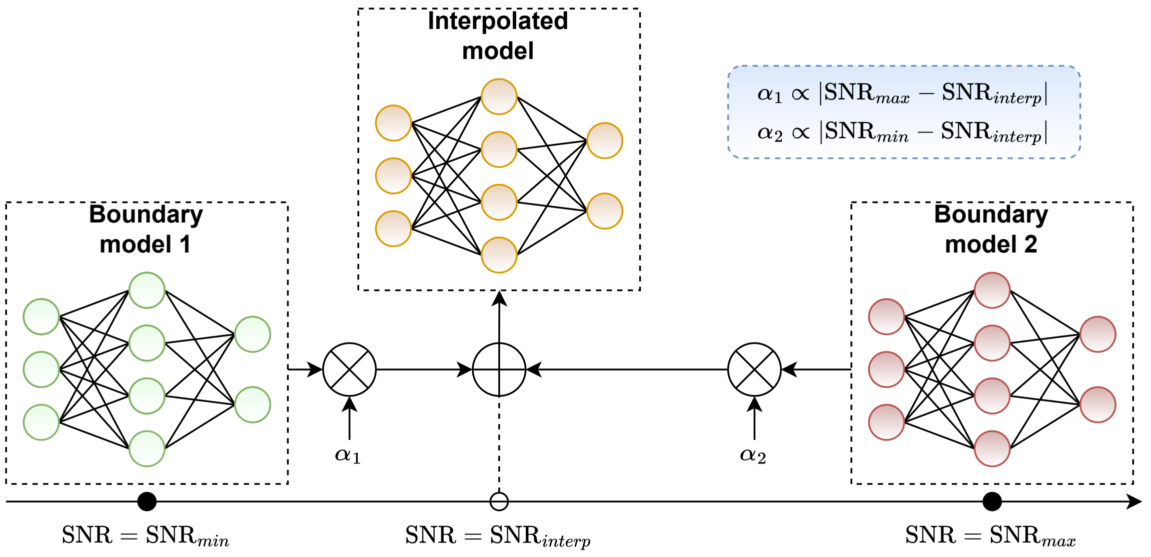

Interpolated model
$\alpha_1 \propto |\mathrm{SNR}_{max} - \mathrm{SNR}_{interp}|$
$\alpha_2 \propto |\mathrm{SNR}_{min} - \mathrm{SNR}_{interp}|$
Boundary model 1
Boundary model 2
α_1
α_2
$\mathrm{SNR} = \mathrm{SNR}_{min}$
$\mathrm{SNR} = \mathrm{SNR}_{interp}$
$\mathrm{SNR} = \mathrm{SNR}_{max}$

Figure 6.3: Proposed interpolation scheme. Given a channel SNR, the interpolated network is obtained as a weighted sum of the boundary networks with weights α_1 and α_2.

DNN design literature [143]. In the first training iteration, we target SNR_{min}, in the second iteration SNR_{max}, and finally, in the third iteration, we target an SNR value randomly sampled from $[\mathrm{SNR}_{min}, \mathrm{SNR}_{max}]$, i.e., $\mathrm{SNR} = \mathrm{SNR}_{interp} \sim \mathcal{U}(\mathrm{SNR}_{min}, \mathrm{SNR}_{max})$. We repeat these three iterations throughout the entire training process to make sure the final DNN can adapt to a variety of SNR values it can experience during testing.

Despite its simplicity, the above method is limited, as it still relies on a single set of parameters, which cannot perfectly adapt to every SNR. To overcome this limitation, we consider an ensemble training approach inspired by [151], which stores only two sets of DNN parameters at the edge server. In this *interpolation scheme* (see Fig. 6.3), the two sets of DNN parameters, $\mathbf{w}_{min}$ and $\mathbf{w}_{max}$, called the *boundary networks*, are trained targeting channel SNR values SNR_{min} and SNR_{max}. In the first iteration, we set the SNR target to SNR_{min}, and train only the boundary network $\mathbf{w}_{min}$. In the second iteration, we repeat this process for the boundary network $\mathbf{w}_{max}$ by setting $\mathrm{SNR} = \mathrm{SNR}_{max}$. Finally, in the third training iteration, we consider a random SNR value from $[\mathrm{SNR}_{min}, \mathrm{SNR}_{max}]$, i.e., $\mathrm{SNR} = \mathrm{SNR}_{interp} \sim \mathcal{U}(\mathrm{SNR}_{min}, \mathrm{SNR}_{max})$, and train a DNN with parameters $\mathbf{w}_{interp}$ equal to the weighted sum of the boundary networks' parameters. The exact values of the interpolated model parameters are calculated as:

$$\mathbf{w}_{interp} = \alpha_1 \mathbf{w}_{min} + \alpha_2 \mathbf{w}_{max}, \tag{6.6}$$

where $\alpha_1 = \frac{|\text{SNR}_{max} - \text{SNR}_{interp}|}{|\text{SNR}_{max} - \text{SNR}_{min}|}$ and we set $\alpha_2 = 1 - \alpha_1$.

This strategy allows us to train a family of interpolated networks that achieve satisfactory performance on a range of SNR values between SNR_{min} and SNR_{max}. In this work, we initialize both boundary networks with the same set of weights, and they naturally converge to different optima as their parameters are updated with different SNRs. Once the networks are trained, during the test phase, depending on the experienced channel SNR we can sample an interpolated network, and also apply SK mapping or repetition schemes to it as desired, in order to further increase the performance.

6.7 Results

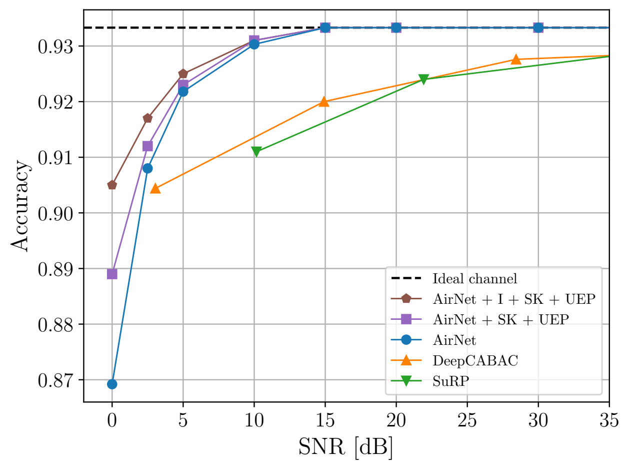
6.7.1 Experimental setup

In this work, we focus on transmitting parameters of DNNs trained for the image classification task. For evaluation, we utilize two distinct datasets. The first dataset we consider is CIFAR10 [23], which consists of 60000 RGB images of 32×32 resolution. The images represent 10 different classes. Following the standard protocol, we utilize 50000 images for training and 10000 for testing, and employ the top-1 classification accuracy as our primary accuracy metric. For a fair comparison with other approaches, we consider Small-VGG16 [27] as our baseline DNN. The Small-VGG16 follows a similar structure to the standard VGG16 network, but replaces the standard classifier head with a new one that consists of three fully-connected layers, the first two containing 512 neurons with ReLU activation, and the third containing 10 output neurons for class prediction. During training we use cross-entropy loss, stochastic gradient descent (SGD) optimizer with a learning rate of 0.01 and momentum of 0.9 for 30 epochs, reduce the learning rate to 0.001 and train for a further 30 epochs. The same procedure applies to both initial training and fine-tuning after every pruning step.

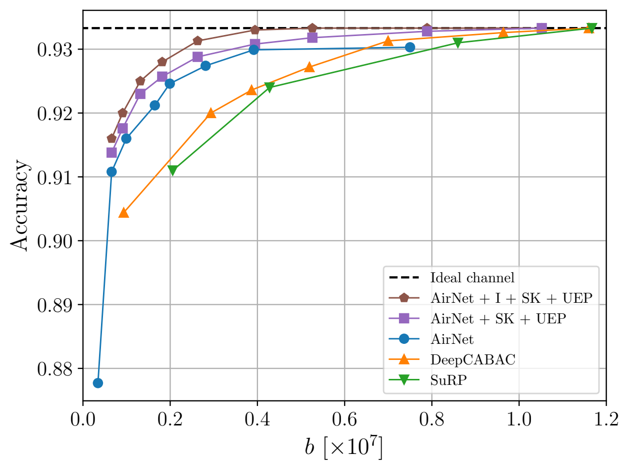
The second dataset we utilize in this work is Tiny ImageNet [165]. The dataset consists of 100000 64×64 training images of 200 classes, where each class is uniformly represented with 500 image samples. The test set consists of 10000 images, and, as before, we use top-1 classification accuracy as our performance metric. We adopt ResNet-34 [93] as our DNN architecture for this task. During training, we utilize cross-entropy loss, and SGD optimizer with a learning rate of 0.001 and momentum of 0.9, reduce the learning rate to 0.0001 after 60 epochs and train for a further 30 epochs. In order to sustain the fixed channel depth after every block of ResNet, during the pruning phase, we only prune the output channels of the first convolutional layer of each ResNet block. Before feeding Tiny ImageNet images into ResNet, we resize them to the resolution of 256×256 and crop the central 224×224 pixels from each image.

We perform multiple training runs of the networks for different SNR values, available bandwidth constraints $\boldsymbol{b}$, channel models, and training strategies. Unless specified differently, in the fading channel scenario, we assume the CSI is available only at the receiver.

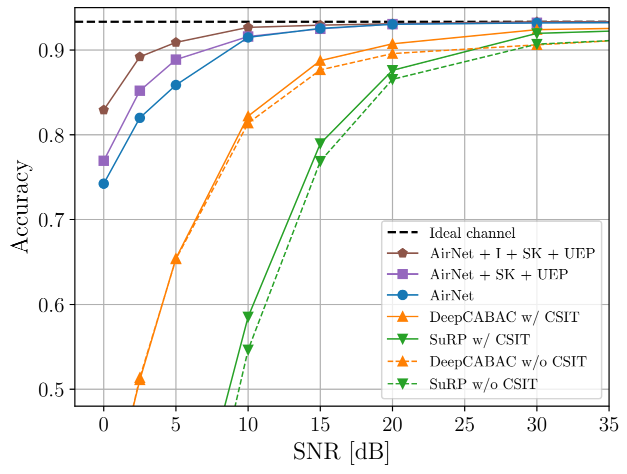
We compare our scheme against two separation-based schemes: DeepCABAC [144] and SuRP [145]. Both methods first perform network pruning to obtain a sparse structure, which is followed by network quantization and compression via either arithmetic or Huffman coding. For both methods, we performed multiple runs of pruning and compression with various hyper-parameters, and considered only the best-performing ones in our evaluations. When digital schemes are used over the fading channel, we consider two scenarios regarding CSI availability. In the first scenario, we consider the CSI is only available at the receiver, and the transmitter is assumed to transmit at a fixed rate. If the channel capacity is below this rate, we assume the transmission fails, i.e., an outage occurs. We then calculate the fraction of successful transmissions and multiply it by the accuracy achieved by the transmitted DNN. In the second scenario, we consider the availability of CSI at both the transmitter and the receiver. This will be denoted by CSIT in the simulations. In this scenario, the transmitter always transmits at the capacity of the channel, which serves as an upper-bound on the performance of separation-based DNN delivery schemes.



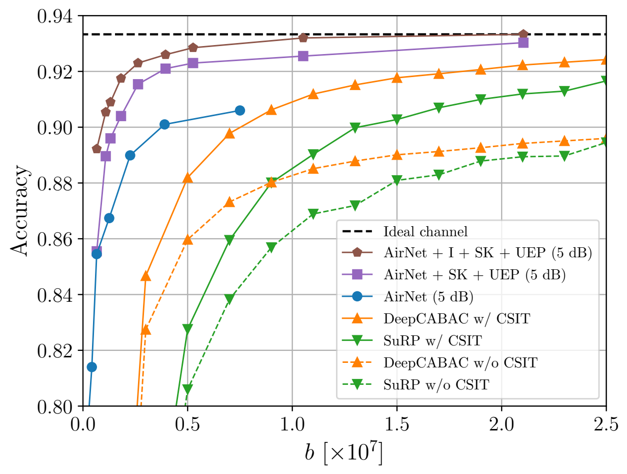
(a) AWGN, $b \approx 1.2 \times 10^6$



(b) AWGN, SNR = 5dB



(c) Fading, $b \approx 1.2 \times 10^6$



(d) Fading, SNR = 5dB

Figure 6.4: Performance comparison between the proposed AirNet approaches, and the alternative digital and analog schemes over AWGN and slow fading channels for a range of channel SNRs and bandwidths in image classification task with Small-VGG16 network and CIFAR10 dataset.

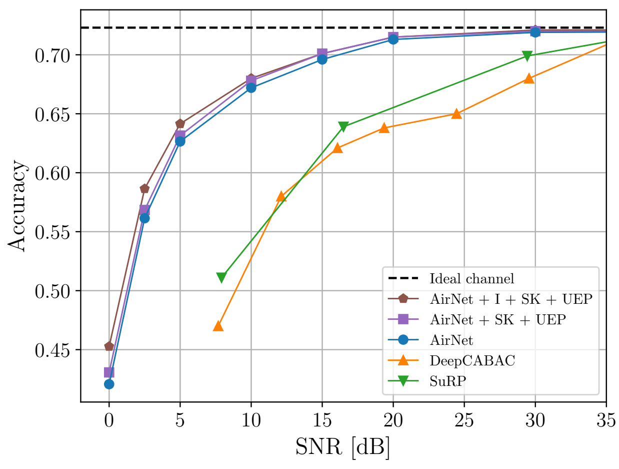
6.7.2 Performance comparison

In this section, we present the comparison between AirNet and alternative separation-based schemes. For AirNet, we consider three alternatives - vanilla AirNet, denoted simply as AirNet, which employs noise injection, pruning, and KD, AirNet with SK bandwidth expansion and UEP (denoted as AirNet + SK + UEP) presented in Section 6.5.1 with Hessian-based sensitivity, and AirNet trained with the interpolation scheme (presented in Section 6.6), with SK scheme and uneven error protection (denoted as AirNet + I + SK + UEP).

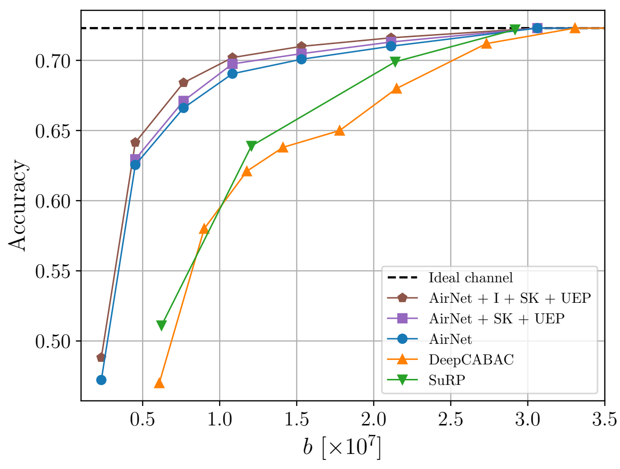
Small-VGG16. In Fig. 6.4, we consider the image classification task with the Small-VGG16 network. In Fig. 6.4a we fix the bandwidth b to approximately 1.2×10^6 channel uses and vary the SNR of an AWGN channel. For the AirNet and AirNet + SK + UEP curves, each point is achieved by a separate model trained specifically for the corresponding SNR value used for testing. On the other hand, for AirNet + I + SK + UEP, each point corresponds to a DNN obtained as the weighted sum of two boundary models, as explained in Section 6.6. For the separation-based DeepCABAC and SuRP approaches, the DNN is compressed to the level allowed by channel capacity. We see that AirNet is able to outperform separation-based alternatives for every value of the SNR by a large margin. This is despite the fact that we assumed capacity-achieving channel coding for the separation-based approaches. More strikingly, AirNet is able to achieve satisfactory accuracy even at extremely low values of SNR. This accuracy is further improved by the use of SK mapping with the UEP scheme. This is particularly beneficial in the low SNR regime. The proposed interpolation scheme not only removes the requirement of training a separate model for every target SNR, but also brings further performance improvement, especially at low SNR values. Our scheme is able to recover the original accuracy of the network at a moderately low SNR value of 15dB, whereas the digital alternatives achieve significantly lower accuracy even at the SNR of 35dB.

Similar trends can be observed when we fix the SNR value to 5dB and vary the available bandwidth b. AirNet is able to outperform digital alternatives at every bandwidth value considered. We note, that SK bandwidth expansion improves the network performance, which indicates that it is better to first prune the network below the available bandwidth, and to further expand it for better protection of the remaining DNN parameters. The vanilla AirNet without SK bandwidth expansion is not able to recover the original DNN accuracy, whereas, with bandwidth expansion, which can exploit the available channel bandwidth for better protection against noise, the original accuracy of the DNN can be recovered at $b \approx 8 \times 10^6$. The digital alternatives are consistently outperformed by the proposed schemes, as they require much larger bandwidth to achieve similar levels of accuracy.

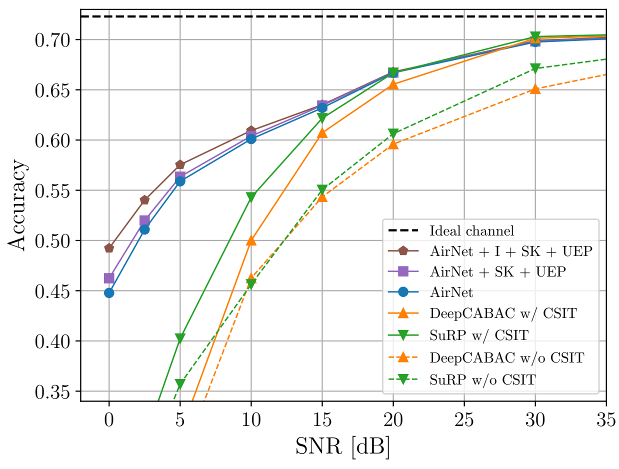
We consider fading channels with different average SNR values, and a fixed bandwidth of $b = 1.2 \times 10^6$ in Fig. 6.4c. Here, each point of the AirNet curves is obtained by training a different DNN to be used over the fading channels with that particular average SNR value.



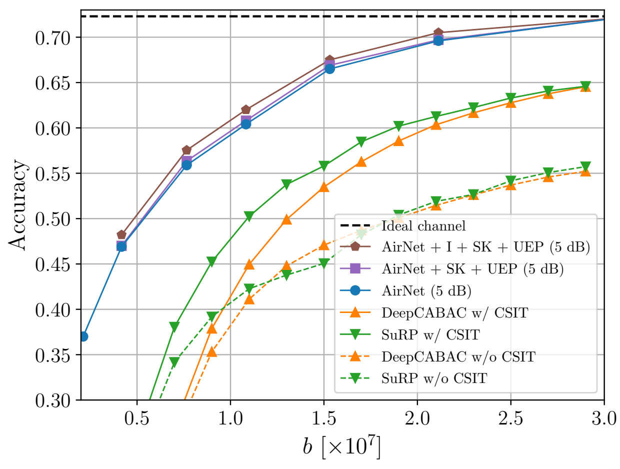
(a) AWGN, $b \approx 4.5 \times 10^6$



(b) AWGN, SNR = 5dB



(c) Fading, $b \approx 7.5 \times 10^6$



(d) Fading, SNR = 5dB

Figure 6.5: Performance comparison between the proposed AirNet approaches, and the alternative digital and analog schemes over AWGN and slow fading channels for a range of channel SNRs and bandwidths in classification task with ResNet-34 network and Tiny ImageNet dataset. Our training strategy generalizes well to different network architectures and datasets.

Again, AirNet is able to achieve better accuracy than the separation-based methods, while also being able to recover the original DNN accuracy at an average SNR of 20dB. The improvement of AirNet is even more evident against digital schemes without CSI. We remind that the AirNet scheme does not assume CSI at the transmitter. These schemes experience sharp accuracy drop whenever the SNR drops below $\sim$ 15dB. As already observed in the AWGN case, SK bandwidth expansion helps to increase the robustness of the network, especially at the low values of SNR, which can be further improved by the interpolation scheme, which also enjoys low memory requirements.

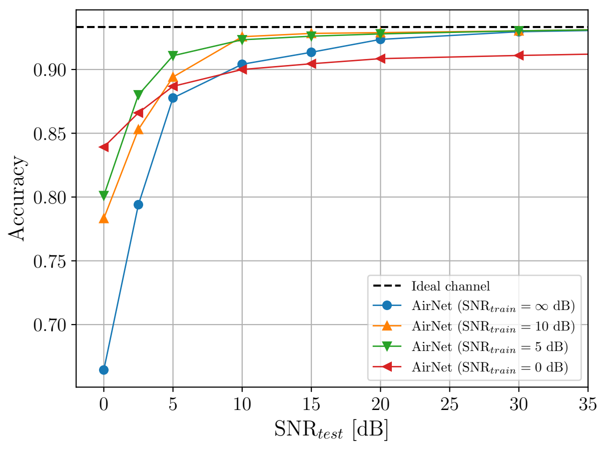
In Fig. 6.4d, we fix the average SNR value to 5dB in the fading regime, while varying the channel bandwidth b. We observe that both vanilla AirNet and AirNet with SK bandwidth expansion are able to achieve satisfactory accuracy at a wide range of channel bandwidths. Separation-based schemes tend to outperform AirNet without SK expansion when they have access to CSIT. The results here further motivate the use of bandwidth expansion as its benefit is clear over fading channels in the low SNR regime. It can be observed that at SNR of 5dB, the interpolation scheme still yields performance improvements over a wide range of bandwidths tested.

ResNet-34. In Fig. 6.5, we present the results for the classification task on the Tiny ImageNet dataset with ResNet-34 network. ResNet-34 architecture differs significantly from Small-VGG16 as it is a much deeper architecture with residual connections and batch normalization layers.

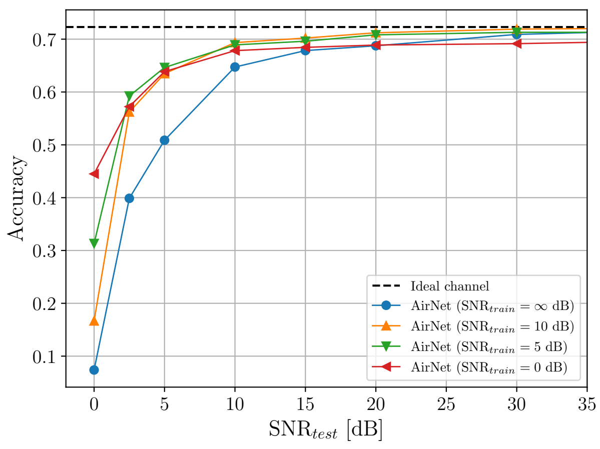
In Fig. 6.5a, we first fix the available bandwidth b to approximately 4.5×10^6 channel uses and train a separate network for a variety of different SNR values. Our observations are consistent with the previous experiments with Small-VGG16. Networks trained with AirNet outperform separation-based counterparts by a large margin, being able to recover the original network accuracy at SNR of 30dB in a significantly more challenging classification task. SK mapping combined with UEP scheme is able to bring improvements over the vanilla AirNet, although the gains are more limited in this scenario. Similarly with the previous experiments, the interpolation scheme increases the accuracy within low SNR regime, compared to SK + UEP, while still significantly reducing storage requirements.

Results presented in Fig. 6.5b show that our strategy consistently outperforms digital alternatives at a wide range of channel bandwidths b for a fixed SNR of 5dB. Small improvements are achieved through applying SK expansion with the UEP scheme, which can be pushed further by applying the interpolation scheme.

Our method shows good generalizability to fading channel scenarios as well, and significantly outperforms separation-based alternatives, both for fixed bandwidth b (see Fig. 6.5c), and fixed SNR (see Fig. 6.5d) scenarios. It is further observed that the SK expansion with UEP leads to performance improvements as before, which can be amplified by the use of the interpolation



(a) Small-VGG16, CIFAR10, $b \approx 0.65 \times 10^6$



(b) ResNet-34, Tiny ImageNet, $b \approx 6.5 \times 10^6$

Figure 6.6: Performance comparison of the models trained for different values of SNR$_{train}$ and tested on a range of SNR$_{test}$.

scheme, especially at low SNRs. For SNR values above 5dB, no significant difference is observed between different variations of AirNet, meaning that the proposed extensions are effective mainly in the low SNR regime.

6.7.3 Generalizability to different noise SNR

So far, we have assumed that the DNN trained for a specific target SNR is tested at that SNR. In this section, we investigate the effects of the mismatch between the training and test SNRs. In Fig. 6.6a, we show the performance of the networks trained for different SNR$_{train}$ values varying between 0dB and 10dB as well as a model trained without noise, denoted as SNR$_{train}$ $= \infty$ dB, when tested with SNR$_{test}$ values between 0dB and 35dB. We see that the model trained and tested at the same SNR, i.e., SNR $=$ SNR$_{train}$ $=$ SNR$_{test}$, achieves the best performance. However, models trained with moderate values of SNR$_{train}$ seem to achieve reasonable performance for a wide range of different SNR$_{test}$. The model trained without noise injection (SNR$_{train}$ $= \infty$ dB) clearly suffers in the low SNR$_{test}$ regime. This indicates the necessity of the noise injection step throughout training. We note that standard separation-based transmission schemes usually exhibit *cliff effect*, i.e., the accuracy sharply degrades when the SNR$_{test}$ falls below the target SNR value. As observed in Fig. 6.6a, our model, on the contrary, exhibits *graceful degradation*, thus its accuracy slowly degrades as the channel noise

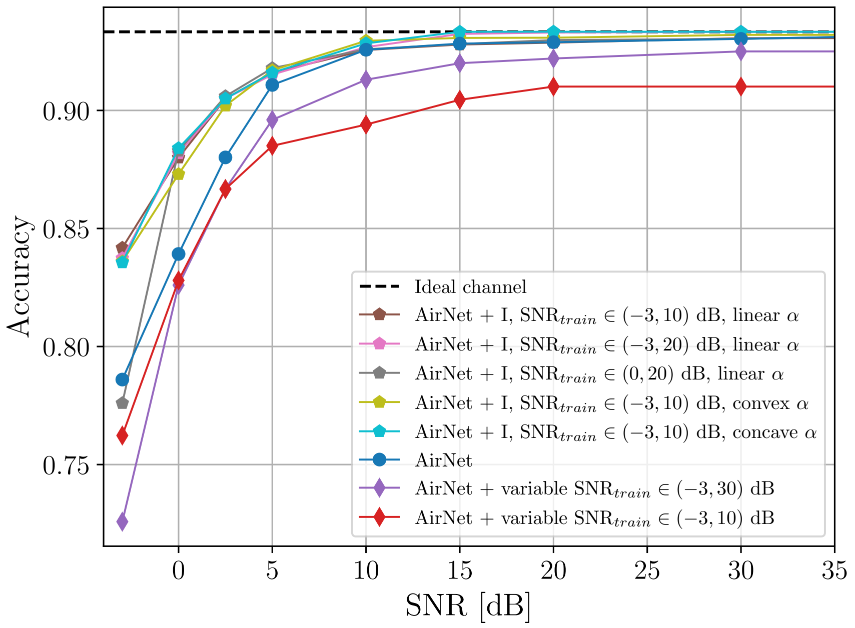
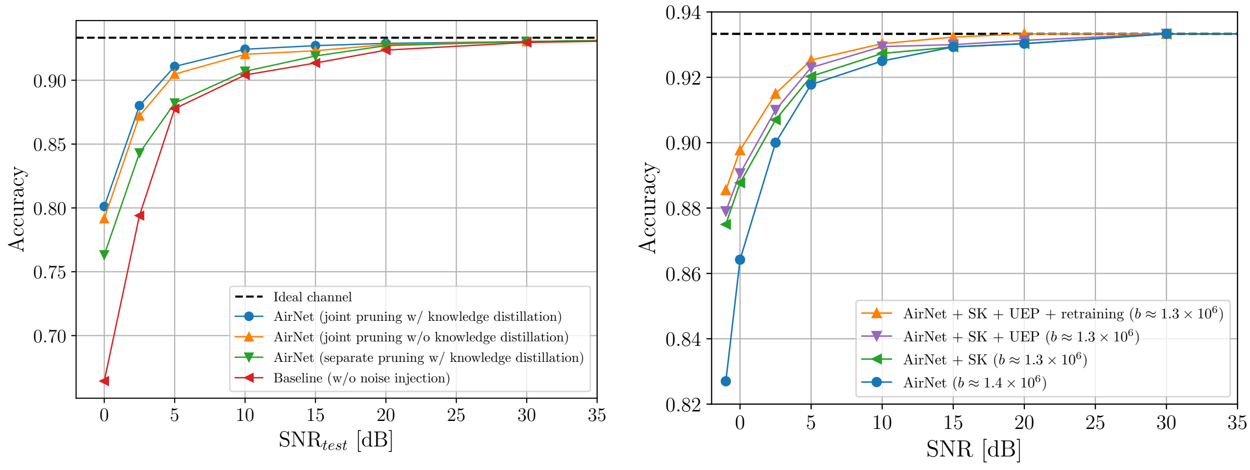


Figure 6.7: Comparison between vanilla AirNet trained with $\text{SNR}_{train} = \text{SNR}_{test}$, interpolation scheme, and AirNet trained with variable SNR. $b \approx 0.65 \times 10^6$.

variance increases, which makes AirNet even more desirable in practical implementations, as it is able to achieve good performance even when accurate channel estimation is not possible. In Fig. 6.6b we show a similar comparison, but with the ResNet-34 baseline network and Tiny ImageNet dataset. We see that the main observations remain consistent between different architectures and datasets.

The above results beg the question of how to train the network when we do not know the channel SNR in advance. The natural approach is to follow the approach used for fading channels, and train the network over a range of SNR values. In Fig. 6.7, we present the accuracy comparison between the three methods as we increase SNR for a fixed bandwidth $b = 0.65 \times 10^6$ channel uses. The blue curve in the figure is the baseline AirNet performance obtained by networks trained for each specific SNR value. One can see that AirNet trained with variable values of SNR (see Section 6.6) generalizes well to a wide range of SNR, but fails to match the performance of separate networks trained for a specific SNR. Moreover, when a very wide range of $\text{SNR}_{train} \in (-3, 30)$ dB is considered, a significant performance degradation is observed for extremely low values of SNR. Training over a more narrow range of SNR values,



(a) Training strategy, $b \approx 0.65 \times 10^6$ (b) Bandwidth expansion

Figure 6.8: Ablation study for our training strategy (a) and various bandwidth expansion strategies (b). Removing any of the steps from the training strategy leads to a drop in the accuracy across all SNRs. We have SNR$_{train}$ = 5dB.

SNR$_{train}$ $\in (-3, 10)$ dB, tends to work better in the low SNR regime, yet saturates to a sub-optimal accuracy value as the SNR increases. We observe that the proposed interpolation scheme not only matches the accuracies achieved by separate networks, but introduces further performance improvements at all SNRs, particularly significant in the low SNR regime. This performance improvement does not depend on a specific choice of the SNR$_{train}$ range, as long as it overlaps with the range of SNR$_{test}$ values experienced during test, or a shape of the function used to calculate the weighting parameter α. In Eq. (6.6), we proposed α(SNR) to be a linear function; however, in Fig. 6.7, one can clearly see that similar results can be obtained when the function is replaced with a simple strictly increasing convex or concave function parameterized by a Bézier curve. Another important observation is that all the schemes should be trained over an SNR range that includes the SNR values expected to be encountered during transmission. Interpolation scheme suffers from sharp performance degradation when trained at SNR$_{train}$ $\in (0, 20)$ dB and tested with SNR$_{test}$ $= -3$dB. Furthermore, it is worth noting that, on top of the performance improvements, the interpolation scheme allows for a significant reduction in the number of parameters stored at the edge server, as it only requires storing two sets of DNN parameters for each bandwidth, as opposed to storing a separate set of DNN parameters for each target SNR value at each bandwidth.

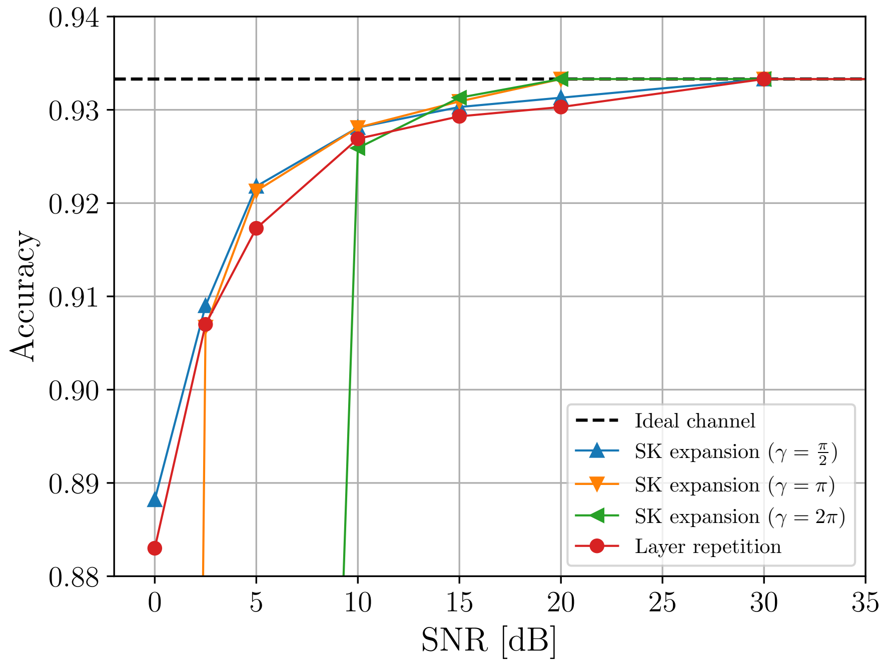


Figure 6.9: Comparison between SK mapping and layer repetition scheme for bandwidth expansion ($\text{SNR}_{train} = 5\text{dB}$, $b \approx 1.3 \times 10^6$).

6.7.4 Evaluation of training steps

In this section, we evaluate the impact of every training step utilized in the AirNet method. A comparison between networks trained with different training strategies is shown in Fig. 6.8a. One can see that each step is crucial to achieve the best possible final accuracy. The best-performing network utilizes KD and *joint pruning*, meaning that the noise injection is performed jointly with fine-tuning after each pruning iteration. Lack of KD results in a slight decrease in the performance, visible especially at $\text{SNR}_{test} < 15\text{dB}$. When we separate pruning from noise injection by first performing pruning, and re-train the network with noise injection afterward, denoted as *separate pruning*, we observe a large drop in the performance for every SNR_{test} value below 20dB. This illustrates that, when a certain amount of noise is injected into the DNN parameters, it effectively learns to prioritize only a few most significant convolutional filters within each layer, whereas the remaining ones get discarded during pruning. Once the pruning process is performed without noise injection, it is impossible for the network to learn this amount of robustness against channel noise. Finally, the model trained without noise injection performs the worst across the entire range.

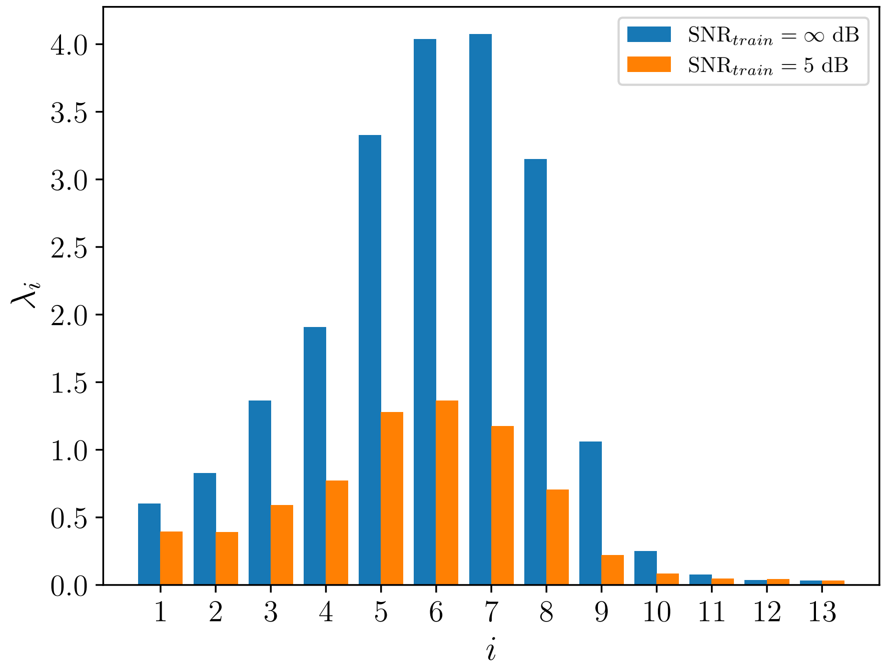


Figure 6.10: Comparison of per-layer sensitivity measured by the largest eigenvalue λ_i of the Hessian matrix corresponding to layer i, $i = 1, \ldots, 13$. Training with noise injection significantly reduces the sensitivity of DNN layers (SNR$_{train}$ = 5dB, $b \approx 0.65 \times 10^6$).

6.7.5 Comparison of bandwidth expansion methods

In this section, we compare the different bandwidth expansion methods described in detail in Section 6.5. In Fig. 6.8b, we show the impact of each bandwidth expansion step on the performance of AirNet. For this comparison, we use vanilla AirNet with noise injection at SNR$_{train}$ = 5dB and $b \approx 1.4 \times 10^6$. For bandwidth expansion, however, we first prune a DNN to $b \approx 0.65 \times 10^6$ with the same SNR$_{train}$ and then expand it to $b \approx 1.3 \times 10^6$. We see that the best accuracy is achieved when we perform Hessian-based UEP with the SK bandwidth expansion scheme, which is followed by DNN retraining during which the variance of noise injected into each layer is scaled according to the number of repetitions calculated. We see that the network without retraining (AirNet + SK + UEP) is able to achieve satisfactory accuracy; however, retraining brings further improvement of up to 0.4% accuracy. AirNet without UEP achieves slightly worse performance, but is still superior to the vanilla AirNet, which uniformly protects all the weights, especially at low SNR values.

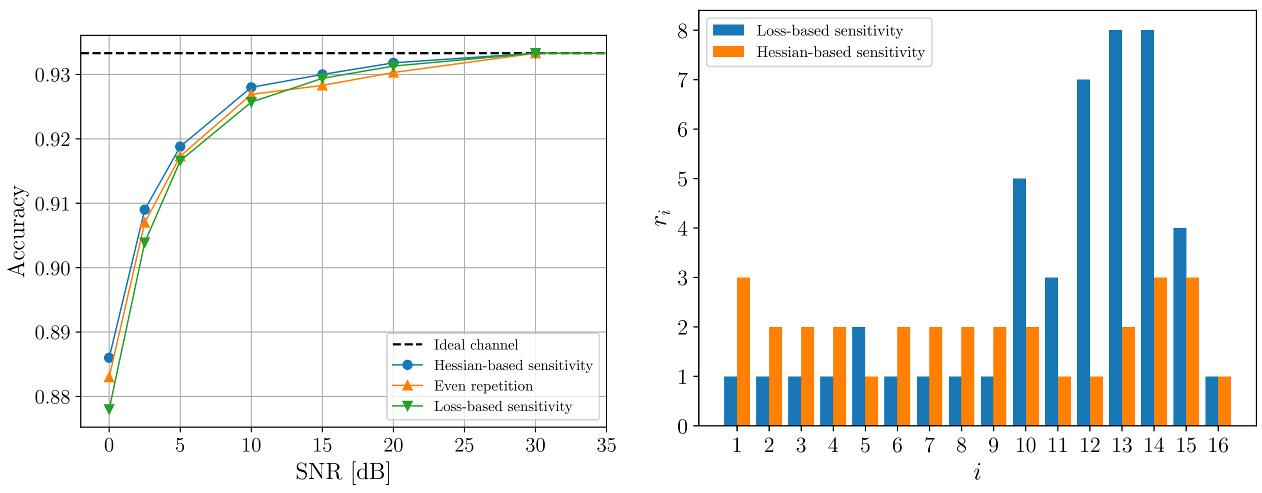
In Fig. 6.9, we consider the SK bandwidth expansion scheme with different values of the γ

parameter, and the simple layer repetition scheme. We see that even for a relatively low value of γ, SK expansion outperforms layer repetition for a wide range of SNR values. The performance of the SK expansion scheme can be further improved in the high SNR regime by increasing the γ value. However, this results in a sharp accuracy drop in the low SNR regime. This is caused by the fact that large γ causes the distance between positive and negative spirals to be small, which results in large positive DNN parameters being decoded as negative values, and vice versa. This means that γ is another hyperparameter of the proposed AirNet scheme with SK bandwidth expansion, which can benefit from the availability of CSI at the transmitter.

6.7.6 Comparison of different sensitivity estimation criteria

In this section, we analyze the different sensitivity criteria used in adaptive layer expansion as introduced in Section 6.5.1. In Fig. 6.10, we compare the Hessian-based sensitivity for a network, when trained with and without noise injection. We observe that training with noise injection significantly reduces the sensitivity of the layers, which is extremely beneficial for further wireless transmission of the network.

Fig. 6.11a compares different sensitivity measuring criteria (Hessian-based and loss-based) in terms of the accuracy they achieve. The number of repetitions per layer of the VGG16 network is presented in Fig. 6.11b. In this comparison, we utilize the layer repetition scheme as it allows any integer number of repetitions, unlike the SK expansion method which is limited to per-layer repetition factor r_i of the form 2^n. We can observe that the Hessian-based strategy performs much better than the alternative schemes for every SNR value considered. Surprisingly, a large difference can be observed between the loss-based and Hessian-based sensitivity metrics in both the performance and the number of repetitions per layer. Despite being intuitively similar, the two methods prioritize different layers; the loss-based method prioritizes the later layers close to the output, while the Hessian-based method distributes the repetitions more evenly across the network. We can further notice that the layer repetition scheme with loss-based sensitivity is outperformed by the even repetition scheme in the low SNR regime, which illustrates that the protection of the early layers is more important than the later layers of the DNN.



(a) Performance comparison (b) Number of repetitions

Figure 6.11: Comparison between UEP schemes with loss-based and Hessian-based sensitivity metrics ($\text{SNR}_{train} = 5\text{dB}$, $b \approx 1.3 \times 10^6$).

6.8 Conclusions

In this work, we studied the important problem of the transmission of the DNN parameters over wireless channels, which is expected to become a significant traffic load for future networks given the increasing adoption of machine learning applications in edge devices. We have proposed training the DNN with noise injection to enable robustness against channel impairments, and network pruning to meet the bandwidth constraint. We have then shown that performance can be improved further, particularly in the low SNR regime by pruning the network to a size below the channel bandwidth and applying bandwidth expansion, which can be considered as an analog error correction technique. We then exploited the fact that not all DNN layers are equally significant for its final accuracy, and introduced an UEP technique by applying selective bandwidth expansion only to the most important layers of the network. Finally, to reduce the memory requirements caused by training a separate network targeting different channel conditions, we developed a novel ensemble training approach that allowed us to simultaneously train a whole spectrum of networks that can be adaptively used in different channel conditions. We believe this method lay the foundations for "on demand" delivery of DNNs in the future mobile networks.

Chapter 7

Conclusion

In the preceding chapters, we have studied problems related to vision-based machine learning over the wireless edge and proposed potential solutions. Thanks to the rapid developments in both deep learning and wireless communication fields, similar solutions are likely to be deployed in real systems in the near future. This would enable the end users to access powerful machine learning algorithms conveniently, without having to own specific specialized equipment. Below we give an overview of the main observations of the works presented in this thesis.

Deep joint source-channel coding. The main observation of this thesis is that joint source-channel coding can be considered a reasonable alternative to separation-based, digital methods for the wireless transmission of various sources related to deep learning, specifically feature vectors and DNN parameters. We have shown that JSCC can not only outperform the separation-based alternatives in task-related metrics, including accuracy or precision, but also provides a number of additional benefits. Firstly, we show that deep JSCC exhibits graceful degradation in various remote inference applications. This is an extremely desirable feature, as it effectively removes the necessity for accurate channel estimation. Whenever the channel conditions change, one can utilize the same deep JSCC algorithm and expect it to provide reasonable performance given the current channel conditions. On the contrary, digital alternatives require very accurate channel estimation to operate optimally. If the transmission rate used for these separation-based schemes is above the current channel capacity, the transmission is very likely to fail, which means the task-related performance will become equivalent to a random guess, due to the outage. On the contrary, if the transmission rate is significantly below the capacity, we do not utilize the channel efficiently, which also leads to obtaining suboptimal performance.

Another benefit of deep JSCC compared to deep learning based digital communication schemes is that the former can be easily modeled as an end-to-end trainable autoencoder, which, given enough data, adapts to source and channel distributions easily, yielding satisfactory performance which simplifies the engineering process leading to the development of new deep JSCC algorithms. Digital-based deep learning algorithms for compression, on the contrary, require significantly more effort to design, as they utilize quantization, which is a non-differentiable operation. To properly address this problem, the quantization operation has to be accurately modeled in a differentiable manner, meaning that an approximate function shall be used in the backward pass of the training loop, leading to lower-quality gradients that do not match the true distribution of the underlying process. Another important caveat of these methods is that they require an accurate estimate of the underlying distribution of the quantized latent, which is necessary for the entropy coding engine to operate as close as possible to the optimal coding rate. JSCC effectively avoids this problem as the entire compression step happens within the trainable autoencoder, which learns both encoding and decoding functions jointly, therefore the latent distribution does not have to be explicitly known or learned for efficient compression.

Edge inference systems. Another key finding of this work relates to the design principles of edge inference systems. Such systems usually include multiple constraints on computational complexity, latency, channel bandwidth, etc., all of which have to be carefully considered in the design.

For example, different approaches to the computational complexity constraint of an edge device were presented in Chapter 3 and Chapter 4. The former assumes the edge device is provided with some additional, high-performance hardware elements capable of running inference for deep learning models deployed at the edge device. This allows the entire forward pass of the

DNN to be performed at the edge device, which significantly reduces the number of computations to be performed at the edge server side, which is desirable, because ideally the edge server would be connected to a large number of edge devices and performing the inference centrally, simultaneously for all the connected edge devices, would require even more powerful hardware available at the edge server. Moreover, such an approach reduces the amount of information to be transmitted over the channel, as the DNN deployed at the transmitter already extracts only the essential, task-related information from the high-dimensional input source. As a result, a bandwidth of a few hundred channel uses was enough to convey the information necessary for successful retrieval in the scenario considered in Chapter 3.

A different system design was used in Chapter 4, where computational or memory constraints were imposed on the edge device, which could not run the full forward pass of the DNN locally. Such a scenario is more realistic, as edge devices are not usually equipped with dedicated hardware to utilize complex deep learning algorithms locally, because of their significant cost or memory requirements. In such scenarios, one has to consider various ways to reduce the computational or memory requirements of running DNNs. One of the approaches is to utilize pruning or quantization of a DNN. Both pruning and quantization were shown to be able to significantly reduce the complexity of the DNNs at a small to no loss in performance. Another method to meet the computational complexity constraints imposed by the edge device in the edge inference scenario is to split a DNN into two parts and only run the forward pass of the first part at the edge device while executing the forward pass of the other part of the DNN at the edge server. This approach has been extensively studied in Chapter 4. It was shown, that the design of such a split system requires the optimization of multiple factors. These include the splitting point of the DNN, which determines how many layers are executed at the edge device, but also the compression algorithm for the intermediate features. In downstream deep learning tasks, the DNN effectively learns how to preserve only the information necessary for the task. This means that, as we progress with the forward pass of the network, less information is stored within every subsequent feature map. This behavior imposes a significant trade-off in the edge inference scenario, since the more layers the DNN is capable of executing locally, at the edge device, the less amount of information has to be transmitted over the wireless network to the edge server for finalizing the forward pass and obtaining the prediction. Therefore, in order to find the optimal splitting point, one has to carefully study the rate-complexity curve that suits their specific performance requirements. The work presented in Chapter 4 studies both DNN splitting and pruning, and shows, that a combination of these two methods leads to significant improvements in the accuracy at a given constraint on the on-device computations, compared to other methods.

The main observation made in Chapter 4 is that more local processing leads to communication savings as the data processed by the layers of a DNN contains progressively less information with each layer. In Chapter 5, we noted that there exist scenarios when it is no longer beneficial to transmit the intermediate feature over the wireless link to the edge server to complete the forward pass of the DNN. Many recent works in the area of deep learning studied early exits, which are mechanisms for obtaining early predictions without the need to utilize the full depth of the network. In early exits, intermediate layers of a DNN are equipped with simple linear classifiers that are trained to map early intermediate features directly to the class predictions. These predictions can be obtained at significant computational savings, but also suffer from decreased accuracy. The findings provided in Chapter 5 show, that wireless edge collaborative inference systems are perfectly suited to adapt such early exit mechanisms. In the case of extremely challenging channel conditions, the noise may prevent the edge server from completing the task at the desired accuracy. Moreover, certain data samples may be easier to recognize than others, thus early exits can be effective enough to succeed in the task. Our evaluations show, that the decision-making mechanism used to decide whether the early exit prediction can be accepted or not, should depend on both outputs from the early exit, and the current channel SNR. As long as both inputs are provided, the exact choice of the decision-making mechanism is not a crucial factor. It can be as complex as a neural network or as simple as a threshold on the early exit confidence. The difference between the accuracy and communication savings imposed by selecting different decision-making mechanisms is negligible. Surprisingly, it was shown that each method can achieve significant savings of approximately 40% at SNR = 0dB without any drop in classification accuracy compared to a scenario presented in Chapter 4, where the intermediate feature was always transmitted to the edge server for obtaining the prediction.

7.1 Future Work

Bandwidth and SNR adaptive JSCC for edge inference. We note, that the majority of the JSCC approaches presented in this thesis produce DNNs that target a specific, predefined bandwidth and SNR. Despite their undeniable performance, such approaches are not desirable, since wireless channel conditions can constantly change due to variations in the local environment of the system. Such changes would lead to sub-optimal performance or even require re-training of the algorithms proposed in Chapter 3 and Chapter 4.

The concept of bandwidth- or SNR-adaptive methods for split inference in the deep JSCC regime has not been extensively studied yet, but some concepts from the classical deep JSCC literature can be easily transferred to the split learning framework. For example, findings of [166] indicate, that it is possible to train a deep JSCC scheme in such a way, that ensures satisfactory image transmission performance for a variety of different bandwidth values. Of course, this imposes a trade-off between the rate and the reconstruction accuracy, since the more channel uses can be utilized for transmitting the features, the better reconstruction quality is expected. A similar concept can be directly transferred to training pipelines and DNN architectures proposed in Chapter 3 and Chapter 4. This is expected that a similar, accuracy-rate trade-off will also exist in the remote inference setting for image classification and retrieval tasks.

The SNR adaptivity problem for the methods proposed in Chapter 3 and Chapter 4 can be effectively tackled with the use of the same methods that were applied in the work proposed in Chapter 6. We note, that in the case of AirNet, it was possible to avoid training a separate DNN for every channel SNR. Instead, we proposed the interpolation scheme and variable SNR scheme, which allowed the network to perform well at a wide SNR range, without the necessity to perform multiple training runs. We expect, that it is possible to train edge inference systems, such that they are treated with channel noises of different SNR values at each iteration, and learn to adapt to frequently changing wireless channel conditions. Implementation of such an approach has to be performed very carefully, with proper training pipelines, to ensure that the accuracy achievable for the adaptive models is close to, or even on par with the accuracy of the separate models, each targeting a separate SNR regime. Success in creating such schemes would lead to a significant reduction in the number of training runs required for edge inference JSCC schemes, effectively removing the need to train multiple models. This would lead to a significant reduction in not only training times but also memory requirements for edge devices, which would be required to store only one DNN model, which can adapt to a variety of channel conditions.

7.2 Epilogue

In this thesis, we studied a variety of problems related to deep JSCC applied to different computer vision tasks, including wireless image retrieval, classification, and transmission of DNN parameters. Despite the existing design standards for modern wireless networks being oriented towards digital communications, we believe that the application of the JSCC might become extremely beneficial. This might eventually lead to the implementation of analog JSCC-based transmission schemes, especially for scenarios with stringent latency requirements, under extreme wireless channel conditions. This would include networks, where it is impossible to estimate the SNR accurately, or where the SNR is rapidly changing. Under such conditions, separation-based methods often fail to deliver any result, whereas JSCC-based approaches can smoothly adapt to channel quality, and provide a result corresponding to current channel conditions. Such behavior is extremely desirable for some of the recently emerging time-dependent applications, including autonomous driving, the defense industry, or fault detection systems, which can hugely benefit from the increased performance of JSCC systems.

Bibliography

[1] M. Chui, M. Issler, R. Roberts, and L. Yee, "McKinsey technology trends outlook 2023," https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-top-trends-in-tech, McKinsey & Company, July 2023, accessed: 25.11.2022.

[2] "ITU DataHub," https://datahub.itu.int/, International Telecommunication Union, accessed: 25.11.2023.

[3] M. Chui, B. Hall, H. Mayhew, A. Singla, and A. Sukharevsky, "The state of AI in 2022—and a half decade in review," https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2022-and-a-half-decade-in-review, McKinsey & Company, August 2022, accessed: 25.11.2022.

[4] M. Wang, W. Fu, X. He, S. Hao, and X. Wu, "A survey on large-scale machine learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 6, pp. 2574–2594, 2020.

[5] R. Torfason, F. Mentzer, E. Ágústsson, M. Tschannen, R. Timofte, and L. V. Gool, "Towards image understanding from deep compression without decoding," in *International Conference on Learning Representations*, 2018.

[6] J. Upadhyay, P. Rida, S. Gupta, and N. Siddique, "Smart doorbell system based on face recognition," *International Research Journal of Engineering and Technology (IRJET)*, vol. 4, no. 03, 2017.

[7] A. Al-Kaff, D. Martin, F. Garcia, A. de la Escalera, and J. M. Armingol, “Survey of computer vision algorithms and applications for unmanned aerial vehicles,” *Expert Systems with Applications*, vol. 92, pp. 447–463, 2018.

[8] M. N. Ahangar, Q. Z. Ahmed, F. A. Khan, and M. Hafeez, “A survey of autonomous vehicles: Enabling communication technologies and challenges,” *Sensors*, vol. 21, no. 3, 2021.

[9] D. Gündüz, D. B. Kurka, M. Jankowski, M. M. Amiri, E. Ozfatura, and S. Sreekumar, “Communicate to learn at the edge,” *IEEE Communications Magazine*, vol. 58, no. 12, pp. 14–19, 2020.

[10] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, “Toward an intelligent edge: Wireless communication meets machine learning,” *IEEE Communications Magazine*, vol. 58, no. 1, pp. 19–25, 2020.

[11] T. Rappaport, “The wireless revolution,” *IEEE Communications Magazine*, vol. 29, no. 11, pp. 52–71, 1991.

[12] D. H. Ring, “Mobile telephony-wide area coverage,” Bell Labs, Tech. Rep., 1947.

[13] H. Viswanathan and M. Weldon, “The past, present, and future of mobile communications,” *Bell Labs Technical Journal*, vol. 19, pp. 8–21, 2014.

[14] J. R. Bhat and S. A. Alqahtani, “6G ecosystem: Current status and future perspective,” *IEEE Access*, vol. 9, pp. 43 134–43 167, 2021.

[15] W. Jiang, B. Han, M. A. Habibi, and H. D. Schotten, “The road towards 6G: A comprehensive survey,” *IEEE Open Journal of the Communications Society*, vol. 2, pp. 334–366, 2021.

[16] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[17] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[18] H. J. Kelley, “Gradient theory of optimal flight paths,” *ARS Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[19] K. Fukushima and S. Miyake, “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition,” in *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[20] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[21] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv:1412.6980*, 2015.

[22] N. Qian, “On the momentum term in gradient descent learning algorithms,” *Neural Networks*, vol. 12, no. 1, pp. 145 – 151, 1999.

[23] A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,” 2009.

[24] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[25] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *Computer Vision – ECCV 2014*. Springer International Publishing, 2014, pp. 740–755.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012.

[27] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *arXiv:1409.1556*, 2014.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison,

A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[29] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/

[30] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[31] W. Weaver, "Recent contributions to the mathematical theory of communication," *ETC: a review of general semantics*, pp. 261–281, 1953.

[32] D. Gündüz, Z. Qin, I. E. Aguerri, H. S. Dhillon, Z. Yang, A. Yener, K. K. Wong, and C.-B. Chae, "Beyond transmitting bits: Context, semantics, and task-oriented communications," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 1, pp. 5–41, 2023.

[33] E. Bourtsoulatze, D. Burth Kurka, and D. Gündüz, "Deep Joint Source-Channel Coding for Wireless Image Transmission," *IEEE Transactions on Cognitive Communications and Networking*, vol. 5, no. 3, pp. 567–579, 2019.

[34] T.-Y. Tung and D. Gündüz, "Deepwive: Deep-learning-aided wireless video transmission," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 9, pp. 2570–2583, 2022.

[35] M. Jankowski, D. Gündüz, and K. Mikolajczyk, "Wireless Image Retrieval at the Edge," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 89–100, 2021.

[36] M. Jankowski, D. Gunduz, and K. Mikolajczyk, “Deep joint source-channel coding for wireless image retrieval,” in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2020, pp. 5070–5074.

[37] M. Jankowski, D. Gündüz, and K. Mikolajczyk, “Joint Device-Edge inference over wireless links with pruning,” in *IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, May 2020.

[38] M. Jankowski, D. Gündüz, and K. Mikolajczyk, “Airnet: Neural network transmission over the air,” in *IEEE International Symposium on Information Theory (ISIT)*, 2022, pp. 2451–2456.

[39] C. E. Shannon, “A Mathematical Theory of Communication,” *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, July 1948.

[40] R. G. Gallager, “Low-density parity-check codes,” *IRE Transactions on Information Theory*, vol. 8, pp. 21–28, 1962.

[41] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes,” in *Proceedings of IEEE International Conference on Communications*, vol. 2, 1993, pp. 1064–1070 vol.2.

[42] E. Arikan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[43] A. Goldsmith, “Joint source/channel coding for wireless channels,” in *IEEE 45th Vehicular Technology Conference*, vol. 2, July 1995, pp. 614–618.

[44] F. Zhai, Y. Eisenberg, and A. K. Katsaggelos, “Joint source-channel coding for video communications,” in *Handbook of Image and Video Processing*, 2005.

[45] G. M. Davis and J. M. Danskin, “Joint source and channel coding for image transmission over lossy packet networks,” *Proceedings of SPIE - The International Society for Optical Engineering*, May 1999.

[46] A. Goldsmith, "Joint source/channel coding for wireless channels," in *1995 IEEE 45th Vehicular Technology Conference*, vol. 2, 1995, pp. 614–618 vol.2.

[47] F. Zhai and A. Katsaggelos, *Joint source-channel video transmission.* Springer Cham, 2007, vol. 10.

[48] A. Ortega and K. Ramchandran, "Rate-distortion methods for image and video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 23–50, 1998.

[49] M. Ruf and J. Modestino, "Operational rate-distortion performance for joint source and channel coding of images," *IEEE Transactions on Image Processing*, vol. 8, no. 3, pp. 305–320, 1999.

[50] I. Kozintsev and K. Ramchandran, "Robust image transmission over energy-constrained time-varying channels using multiresolution joint source-channel coding," *IEEE Transactions on Signal Processing*, vol. 46, no. 4, pp. 1012–1026, 1998.

[51] V. Stankovic, R. Hamzaoui, Y. Charfi, and Z. Xiong, "Real-time unequal error protection algorithms for progressive image transmission," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 10, pp. 1526–1535, 2003.

[52] D. B. Kurka and D. Gündüz, "Deepjscc-f: Deep joint source-channel coding of images with feedback," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 178–193, 2020.

[53] D. B. Kurka and D. Gündüz, "Bandwidth-agile image transmission with deep joint source-channel coding," *IEEE Transactions on Wireless Communications*, vol. 20, no. 12, pp. 8081–8095, 2021.

[54] M. Yang, C. Bian, and H.-S. Kim, "Deep joint source channel coding for wireless image transmission with OFDM," in *IEEE International Conference on Communications*, 2021, pp. 1–6.

[55] Z. Weng and Z. Qin, "Semantic communication systems for speech transmission," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2434–2444, 2021.

[56] H. Xie and Z. Qin, “A lite distributed semantic communication system for internet of things,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 142–153, 2021.

[57] N. Farsad, M. Rao, and A. Goldsmith, “Deep Learning for Joint Source-Channel Coding of Text,” in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2018, pp. 2326–2330.

[58] A. E. Kalør, D. Michelsanti, F. Chiariotti, Z.-H. Tan, and P. Popovski, “Remote anomaly detection in industry 4.0 using resource-constrained devices,” in *IEEE 22nd International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2021, pp. 251–255.

[59] D. B. Kurka and D. Gündüz, “Successive refinement of images with deep joint source-channel coding,” in *IEEE International Workshop on Signal Processing and Advances in Wireless Communications (SPAWC)*, July 2019, pp. 1–5.

[60] H. Xie, Z. Qin, G. Y. Li, and B.-H. Juang, “Deep learning enabled semantic communication systems,” *IEEE Transactions on Signal Processing*, vol. 69, pp. 2663–2675, 2021.

[61] Z. Weng and Z. Qin, “Semantic communication systems for speech transmission,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2434–2444, 2021.

[62] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, “Distributed learning in wireless networks: Recent progress and future challenges,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3579–3605, 2021.

[63] J. Shao and J. Zhang, “Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems,” in *IEEE International Conference on Communications Workshops (ICC Workshops)*, 2020, pp. 1–6.

[64] D. Wen, X. Jiao, P. Liu, G. Zhu, Y. Shi, and K. Huang, “Task-oriented over-the-air computation for multi-device edge AI,” *arXiv:2112.07244*, 2022.

[65] S. F. Yilmaz, B. Hasırcıoğlu, and D. Gündüz, "Over-the-air ensemble inference with model privacy," in *IEEE International Symposium on Information Theory (ISIT)*, 2022, pp. 1265–1270.

[66] Q. Lan, Q. Zeng, P. Popovski, D. Gündüz, and K. Huang, "Progressive feature transmission for split classification at the wireless edge," *IEEE Transactions on Wireless Communications*, vol. 22, no. 6, pp. 3837–3852, 2023.

[67] G. Zhang, Q. Hu, Z. Qin, Y. Cai, and G. Yu, "A unified multi-task semantic communication system with domain adaptation," in *IEEE Global Communications Conference (GLOBECOM)*, 2022, pp. 3971–3976.

[68] M. Wang, Z. Zhang, J. Li, M. Ma, and X. Fan, "Deep joint source-channel coding for multi-task network," *IEEE Signal Processing Letters*, vol. 28, pp. 1973–1977, 2021.

[69] J. S. Assine, J. Santos Filho, and E. Valle, "Collaborative object detectors adaptive to bandwidth and computation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 2839–2843.

[70] G. K. Wallace, "The JPEG still picture compression standard," *Communications of the ACM*, vol. 34, no. 4, pp. 30–44, Apr. 1991.

[71] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Transactions on Computers*, vol. C-23, no. 1, pp. 90–93, Jan 1974.

[72] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 36–58, Sep. 2001.

[73] F. Bellard, "Better portable graphics," https://bellard.org/bpg/, 2014, accessed: 25.11.2022.

[74] "WebP image format," https://developers.google.com/speed/webp, accessed: 25.11.2022.

[75] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," in *International Conference on Learning Representations*, 2017.

[76] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy image compression with compressive autoencoders," *arXiv:1703.00395*, 2017.

[77] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," in *International Conference on Learning Representations*, 2018.

[78] J. Ballé, V. Laparra, and E. P. Simoncelli, "Density modeling of images using a generalized normalization transformation," *arXiv:1511.06281*, 2015.

[79] D. Minnen, J. Ballé, and G. Toderici, "Joint autoregressive and hierarchical priors for learned image compression," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 10 771–10 780.

[80] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2325–2383, Oct 1998.

[81] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[82] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680.

[83] E. Agustsson, M. Tschannen, F. Mentzer, R. Timofte, and L. V. Gool, "Generative adversarial networks for extreme learned image compression," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 221–231.

[84] S. Santurkar, D. Budden, and N. Shavit, "Generative compression," in *Picture Coding Symposium (PCS)*, 2018, pp. 258–262.

[85] "Portable network graphics (PNG)," http://www.libpng.org/pub/png, accessed: 25.11.2022.

[86] J. Sneyers and P. Wuille, "FLIF: Free lossless image format based on maniac compression," in *IEEE International Conference on Image Processing (ICIP)*, Sep. 2016, pp. 66–70.

[87] L. Theis, A. van den Oord, and M. Bethge, "A note on the evaluation of generative models," *arXiv:1511.01844*, 2016.

[88] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool, "Practical full resolution learned lossless image compression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[89] F. Mentzer, L. V. Gool, and M. Tschannen, "Learning better lossless compression using lossy compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[90] M. C. Anderson, M. Atkins, and J. Vaisey, "Task-oriented lossy compression of magnetic resonance images," *Proceedings of SPIE - The International Society for Optical Engineering*, 1997.

[91] L. Pu, M. W. Marcellin, A. Bilgin, and A. Ashok, "Image compression based on task-specific information," in *IEEE International Conference on Image Processing (ICIP)*, Oct 2014, pp. 4817–4821.

[92] L. Pu, M. W. Marcellin, A. Bilgin, and A. Ashok, "Compression based on a joint task-specific information metric," in *Data Compression Conference*, April 2015, pp. 467–467.

[93] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[94] N. Bian, F. Liang, H. Fu, and B. Lei, "A deep image compression framework for face recognition," in *2nd China Symposium on Cognitive Computing and Hybrid Intelligence (CCHI)*, 2019, pp. 99–104.

[95] S. Singh, S. Abu-El-Haija, N. Johnston, J. Ballé, A. Shrivastava, and G. Toderici, "End-to-end learning of compressible features," in *IEEE International Conference on Image Processing (ICIP)*, 2020, pp. 3349–3353.

[96] Y. LeCun, J. Denker, and S. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing Systems*, 1990.

[97] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Graf, "Pruning Filters for Efficient Convnets," in *International Conference on Learning Representations (ICLR)*, 2017.

[98] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning Convolutional Neural Networks for Resource Efficient Inference," in *International Conference on Learning Representations (ICLR)*, 2016.

[99] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[100] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1398–1406.

[101] N. Lee, T. Ajanthan, and P. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," in *International Conference on Learning Representations (ICLR)*, 2019.

[102] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "Hrank: Filter pruning using high-rank feature map," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[103] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression," in *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[104] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations (ICLR)*, 2019.

[105] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted Residuals and Linear Bottlenecks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[106] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[107] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[108] L. Zheng, Y. Yang, and A. G. Hauptmann, "Person re-identification: Past, present and future," *arXiv:1610.02984*, 2016.

[109] B. He, J. Li, Y. Zhao, and Y. Tian, "Part-regularized near-duplicate vehicle re-identification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3997–4005.

[110] L. Zheng, Y. Huang, H. Lu, and Y. Yang, "Pose-invariant embedding for deep person re-identification," *IEEE Transactions on Image Processing*, vol. 28, no. 9, pp. 4500–4509, 2019.

[111] R. Kuma, E. Weill, F. Aghdasi, and P. Sriram, "Vehicle re-identification: an efficient baseline using triplet embedding," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–9.

[112] T. Chen, S. Ding, J. Xie, Y. Yuan, W. Chen, Y. Yang, Z. Ren, and Z. Wang, "Abd-net: Attentive but diverse person re-identification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 8351–8361.

[113] Y. Fu, Y. Wei, Y. Zhou, H. Shi, G. Huang, X. Wang, Z. Yao, and T. Huang, "Horizontal pyramid matching for person re-identification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 8295–8302.

[114] Y. Zhou and L. Shao, "Aware attentive multi-view inference for vehicle re-identification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6489–6498.

[115] G. Wang, Y. Yuan, X. Chen, J. Li, and X. Zhou, "Learning discriminative features with multiple granularities for person re-identification," in *2018 ACM Multimedia Conference on Multimedia Conference.* ACM, 2018, pp. 274–282.

[116] X. Chang, T. M. Hospedales, and T. Xiang, "Multi-level factorisation net for person re-identification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2109–2118.

[117] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *arXiv:1703.07737*, 2017.

[118] F. Mentzer, L. V. Gool, and M. Tschannen, "Learning better lossless compression using lossy compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[119] W. Li, R. Zhao, T. Xiao, and X. Wang, "DeepReID: Deep filter pairing neural network for person re-identification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[120] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, "Scalable person re-identification: A benchmark," in *IEEE International Conference on Computer Vision (ICCV)*, Dec 2015, pp. 1116–1124.

[121] X. Liu, W. Liu, T. Mei, and H. Ma, "A deep learning-based approach to progressive vehicle re-identification for urban surveillance," in *Computer Vision – ECCV 2016.* Cham: Springer International Publishing, 2016, pp. 869–884.

[122] X. Liu, W. Liu, T. Mei, and H. Ma, “Provid: Progressive and multimodal vehicle reidentification for large-scale urban surveillance,” *IEEE Transactions on Multimedia*, vol. 20, no. 3, pp. 645–658, 2018.

[123] R. G. Gallager, *Information Theory and Reliable Communication.* USA: John Wiley & Sons, Inc., 1968.

[124] D. Gündüz, P. de Kerret, N. D. Sidiropoulos, D. Gesbert, C. R. Murthy, and M. Schaar, “Machine learning in the air,” *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 10, pp. 2184–2199, 2019.

[125] A. E. Eshratifar, A. Esmaili, and M. Pedram, “Bottlenet: A deep learning architecture for intelligent mobile cloud computing services,” in *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2019, pp. 1–6.

[126] S. Lucero, “IoT platforms: enabling the internet of things,” March 2016, accessed: 25.11.2022. [Online]. Available: https://cdn.ihs.com/www/pdf/enabling-IOT.pdf

[127] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, “Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution,” in *IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS).* IEEE, 2018, pp. 671–678.

[128] C. Lee, J. Lin, P. Chen, and Y. Chang, “Deep learning-constructed joint transmission-recognition for internet of things,” *IEEE Access*, vol. 7, pp. 76 547–76 561, 2019.

[129] W. Shi, Y. Hou, S. Zhou, Z. Niu, Y. Zhang, and L. Geng, “Improving device-edge cooperative inference of deep learning via 2-step pruning,” in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019.

[130] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in *International Conference on Machine Learning (ICML)*, vol. 30, no. 1, 2013, p. 3.

[131] H. Robbins and S. Monro, “A stochastic approximation method,” *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, September 1951.

[132] S. Scardapane, M. Scarpiniti, E. Baccarelli, and A. Uncini, “Why should we add early exits to neural networks?” *Cognitive Computation*, 2020.

[133] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[134] F. Ilhan, L. Liu, K.-H. Chow, W. Wei, Y. Wu, M. Lee, R. Kompella, H. Latapie, and G. Liu, “EENet: Learning to early exit for adaptive inference,” *arXiv:2301.07099*, 2023.

[135] X. Dai, X. Kong, and T. Guo, “EPNet: Learning to exit with flexible multi-branch network,” in *ACM International Conference on Information and Knowledge Management*, 2020.

[136] W. Ren, Y. Qu, C. Dong, Y. Jing, H. Sun, Q. Wu, and S. Guo, “A survey on collaborative DNN inference for edge intelligence,” *Machine Intelligence Research*, vol. 20, 2023.

[137] E. Samikwa, A. Di Maio, and T. Braun, “Adaptive early exit of computation for energy-efficient and low-latency machine learning over iot networks,” in *IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, 2022.

[138] S. Teerapittayanon, B. McDanel, and H. Kung, “BranchyNet: Fast inference via early exiting from deep neural networks,” in *International Conference on Pattern Recognition (ICPR)*, 2016.

[139] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, “Adaptive neural networks for efficient inference,” in *International Conference on Machine Learning (ICML)*, 2017.

[140] M. Bullo, S. Jardak, P. Carnelli, and D. Gündüz, “Energy-aware early exiting for dynamic inference in devices with energy harvesting capabilities,” in *IEEE International Workshop on Machine Learning for Signal Processing*, 2023.

[141] Y. Matsubara, M. Levorato, and F. Restuccia, “Split computing and early exiting for deep learning applications: Survey and research challenges,” *ACM Computing Surveys*, 2022.

[142] R. G. Pacheco, R. S. Couto, and O. Simeone, "Calibration-aided edge inference offloading via adaptive model partitioning of deep neural networks," in *IEEE International Conference on Communications*, 2021.

[143] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1803–1811.

[144] S. Wiedemann et al., "DeepCABAC: A Universal Compression Algorithm for Deep Neural Networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 700–714, 2020.

[145] B. Isik, T. Weissman, and A. No, "An information-theoretic justification for model pruning," in *International Conference on Artificial Intelligence and Statistics*, vol. 151. PMLR, 2022, pp. 3821–3846.

[146] "Uplink and slow time-to-content: Extract from the Ericsson mobility report," https://www.3gpp.org/ftp/Specs/archive/22_series/22.874/, Ericsson, Tech. Rep., 2016.

[147] R. Zur, Y. Jiang, L. Pesce, and K. Drukker, "Noise Injection for Training Artificial Neural Networks: A Comparison With Weight Decay and Early Stopping," *Medical physics*, vol. 36, pp. 4810–8, 10 2009.

[148] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[149] V. A. Kotelnikov, *The Theory of Optimum Noise Immunity.* McGraw-Hill Book Company, Inc., 1959.

[150] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "HAWQ: Hessian aware quantization of neural networks with mixed-precision," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 293–302.

[151] T. Garipov, P. Izmailov, D. Podoprikhin, D. P. Vetrov, and A. G. Wilson, "Loss surfaces, mode connectivity, and fast ensembling of dnns," in *Advances in Neural Information*

Processing Systems, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018.

[152] "5G system (5GS); study on traffic characteristics and performance requirements for AI/ML model transfer," https://www.3gpp.org/ftp/Specs/archive/22_series/22.874/, 3GPP, Tech. Rep., 2021.

[153] M. B. Mashhadi, M. Jankowski, T.-Y. Tung, S. Kobus, and D. Gündüz, "Federated mmWave beam selection utilizing LIDAR data," *IEEE Wireless Communications Letters*, vol. 10, no. 10, pp. 2269–2273, 2021.

[154] M. Zecchin, M. B. Mashhadi, M. Jankowski, D. Gündüz, M. Kountouris, and D. Gesbert, "LIDAR and position-aided mmWave beam selection with non-local CNNs and curriculum training," *IEEE Transactions on Vehicular Technology*, 2022.

[155] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. Feljan, and V. Poor, "Distributed learning in wireless networks: Recent progress and future challenges," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3579–3605, 2021.

[156] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-S. Hua, "Quantization networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[157] F. Zhu, R. Gong, F. Yu, X. Liu, Y. Wang, Z. Li, X. Yang, and J. Yan, "Towards unified int8 training for convolutional neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[158] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer, "Zeroq: A novel zero shot quantization framework," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 169–13 178.

[159] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-Nets: Learned quantization for highly accurate and compact deep neural networks," in *European Conference on Computer Vision (ECCV)*, 2018.

[160] Q. Jin, L. Yang, and Z. Liao, “Adabits: Neural network quantization with adaptive bit-widths,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[161] R. Banner, Y. Nahshan, E. Hoffer, and D. Soudry, “ACIQ: analytical clipping for integer quantization of neural networks,” *arXiv:1810.05723*, 2018.

[162] B. Isik, K. Choi, X. Zheng, T. Weissman, S. Ermon, H.-S. P. Wong, and A. Alaghi, “Neural network compression for noisy storage devices,” *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 3, May 2023.

[163] F. Hekland, P. A. Floor, and T. A. Ramstad, “Shannon-Kotelnikov Mappings in Joint Source-Channel Coding,” *IEEE Transactions on Communications*, vol. 57, no. 1, pp. 94–105, 2009.

[164] R. V. Mises and H. Pollaczek-Geiringer, “Praktische verfahren der gleichungsauflösung,” *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 9, no. 2, pp. 152–164, 1929.

[165] F.-F. Li, A. Karpathy, and J. Johnson. Tiny imagenet visual recognition challenge. Accessed: 25.11.2022. [Online]. Available: https://www.kaggle.com/c/tiny-imagenet

[166] M. Yang and H.-S. Kim, “Deep joint source-channel coding for wireless image transmission with adaptive rate control,” *arXiv:2110.04456*, 2021.