Deep Learning for Next-Generation Communication Technologies

ISBN: 978-94-6473-892-6

Research institute for mathematics & computer science in the Netherlands

Typeset by LaTeX.
Printed by: Ipskamp
Cover design by: Arwin Gansekoele and Midjourney v7

VRIJE UNIVERSITEIT

Deep Learning for Next-Generation Communication Technologies

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor of Philosophy aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. J.J.G. Geurts,
volgens besluit van de decaan
van de Faculteit der Bètawetenschappen
in het openbaar te verdedigen
op woensdag 17 september 2025 om 9.45 uur
in de universiteit

door

Adriaan Winand Gansekoele

geboren te Almere

Voorwoord

Aan het einde van een PhD is het gebruikelijk (lees: vereist) om het werk te bundelen in een boek. Toch is het een gek gevoel om 4 jaar aan werk samengevat te zien in wat vaak liefkozend 'het boekje' wordt genoemd. Op een gegeven moment was er genoeg werk af en lag het boekje daar; hij was af. Degene die dit boekje nu in zijn handen houdt, wil ik alvast heel veel leesplezier meegeven. Eerst wil ik nog de mensen bedanken die instrumenteel waren in het tot stand brengen van dit boekje.

Degenen die mij goed kennen, weten dat ik nogal koppig kan zijn. Zeker mijn begeleiders, Mark, Rob en Sandjai, hebben daar hun handen vol mee gehad. Toch zijn ze altijd geduldig geweest en hebben ze mij de ruimte gegeven om te groeien. Die groei was duidelijk zichtbaar over de jaren heen: de artikelen werden kwalitatief beter en kwamen steeds sneller achter elkaar. Ik heb mijzelf enorm kunnen ontwikkelen met hun hulp. Ik wil jullie van harte bedanken voor vier mooie jaren en ik wens Rob alvast veel genot van het gratis bier.

Ik heb het geluk gehad dat ik veel samen heb mogen werken met getalenteerde onderzoekers. Tom en Tycho wil ik bedanken voor hun inhoudelijke kennis. Alexios I would like to thank for his expert guidance in an area of research I was still unfamiliar with at the time. Yura en Hilde wil ik bedanken voor de fijne samenwerkingen met interessante inhoudelijke discussies. Emiel wil ik bedanken voor een leuke samenwerking met mooi resultaat. Als laatste wil ik mijn projectgenoten, Britt, Etienne, Jan en Joris bedanken. Ik heb ontzettend genoten van onze tijd samen en hoop jullie nog regelmatig te zien in de toekomst.

Ik wil ook de rest van de groep bedanken. Ik heb ontzettend mooie herinneringen aan onze tijd samen. Ik wil ten eerste mijn mede covid starters bedanken, Rebekka en Robin. Het was geen makkelijk begin, maar het is ons allemaal gelukt uiteindelijk. Guus en Salim vertegenwoordigden samen het 113-project en waren allebei onmisbaar voor de sfeer. Nadat ik begon, kwamen er al snel veel nieuwe aanwinsten bij de groep, waaronder mijn kamergenoten de eerste drie jaar. Ruurd en Joris waren altijd in voor hoogwaardige gesprekken over de prachtige stad Almere. Ruurd was ook altijd in voor een wedje (ja, je krijgt die XXL frikandel nog). Hoewel Jesse en Tim niet altijd blij waren met de geluidsoverlast, was het nooit saai bij ons. Met Berend en Hilde heb ik met veel genoegen een kamer mogen delen in het laatste jaar. Verder wens ik Elisabeth, Ismail, Moos en Simon nog veel plezier in hun traject als (relatief) nieuwe aanwinsten van de groep.

Op de VU heb ik het genoegen gehad om bij nog twee onderzoeksgroepen betrokken te zijn. Van de A&O groep wil ik Rik, Jesper, Anni, Jeroen, Erica, Yoram, Leon, Jasper, Mathijs, Yasmin, Laith en Ritsaart bedanken. De donderdagen waren altijd wat om naar uit te kijken, vooral dankzij jullie. Van de QDA groep wil ik Anne, David, Jacob, Louk, Tariq, Lucas, Rutger, Bob, Vincent, Ameet en Laurens bedanken. Ik kijk met veel genot terug en wens degenen die nog bezig zijn enorm veel succes.

Naast degenen die direct betrokken waren, wil ik vrienden en familie bedanken voor al hun steun over de jaren heen. Ik wil mijn ouders en mijn zusjes even in het licht zetten voor hun steun de afgelopen jaren. Zonder jullie had ik het niet gehaald. Van mijn vrienden wil ik onder andere Dion en Kalvin bedanken. Het was niet altijd makkelijk en jullie waren er altijd voor mij. Ook Joshua en Mark wil ik bedanken voor de mooie tijden die we beleefd hebben de afgelopen jaren. Ik hoop veel van de oude PhD groep nog vaak tegen te komen.

Contents

List of Abbreviations **vii**

1 Introduction **1**
1.1 The Layers of Networking 3
1.2 Physical Layer Challenges 5
1.3 Application Layer Challenges 8
1.4 Overview of the Dissertation 11

I Deep Learning for the Physical Layer **13**

2 Joint Demapping of QAM and APSK Constellations Using Machine Learning **15**
2.1 Introduction 16
2.2 Related Works 18
2.3 Methodology 20
2.4 Experimental Setup 31
2.5 Simulation Results 35
2.6 Discussion and Conclusion 45

3 Relative Phase Equivariant Deep Neural Systems for Physical Layer Communications **47**
3.1 Introduction 48
3.2 Related Works 50
3.3 Methods 51
3.4 Experimental Setup 59
3.5 Results 60
3.6 Discussion and Conclusion 66

Appendices **69**
3.A Additional Figures . . . 70
3.B Circular Convolution Proof . . . 71

4 The Computation of Generalized Embeddings for Underwater Acoustic Target Recognition using Contrastive Learning **73**
4.1 Introduction . . . 74
4.2 Material and Methods . . . 76
4.3 Results . . . 82
4.4 Discussion and Conclusion . . . 88

II Deep Learning for the Application Layer **91**

5 Unveiling the Potential: Harnessing Deep Metric Learning to Circumvent Video Streaming Encryption **93**
5.1 Introduction . . . 94
5.2 Related Work . . . 96
5.3 Methodology . . . 97
5.4 Data Collection . . . 100
5.5 Experimental Setup . . . 101
5.6 Results . . . 103
5.7 Discussion and Conclusion . . . 108

6 Robustly Overfitting Latents for Flexible Neural Image Compression **111**
6.1 Introduction . . . 112
6.2 Preliminaries and Related Work . . . 113
6.3 Methods . . . 116
6.4 Experiments . . . 119
6.5 Analysis and Societal Impact . . . 122
6.6 Discussion and Conclusion . . . 125

Appendices **127**
6.A Derivatives Proof . . . 127
6.B Additional Results . . . 128
6.C Mean-Scale Hyperprior . . . 133

7 Meta-Learning for Federated Face Recognition in Imbalanced Data Regimes **141**
7.1 Introduction . . . 142
7.2 Related Work . . . 143
7.3 Methods . . . 144
7.4 Experimental Setup . . . 148
7.5 Results . . . 150
7.6 Discussion and Conclusion . . . 158

8 Discussion **159**

8.1 Research Questions 160
8.2 Open Problems 163
8.3 Concluding Remarks 164

Publications **165**

Bibliography **167**

Online References **181**

Summary **183**

Samenvatting **187**

About the Author **191**

List of Abbreviations

(A)PSK	(Amplitude and) Phase Shift Keying
3GPP	3rd Generation Partnership Compliant
AE	Autoencoder
AI	Artificial Intelligence
AMR	Automatic Modulation Recognition
ANN	Artificial Neural Network
AWGN	Additive White Gaussian Noise
BASK	Binary Amplitude Shift Keying
BCE	Binary Cross-Entropy
BER	Bit Error Rate
BFSK	Binary Frequency Shift Keying
BPP	Bits Per Pixel
BPSK	Binary Phase Shift Keying
BYOL	Bootstrap Your Own Latent
CDL	Channel Delay Line
CE	Cross-Entropy
CL	Contrastive Learning
CNN	Convolutional Neural Network
CSI	Channel State Information

DASH	Dynamic Adaptive Streaming over HTTP
DFT	Discrete Fourier Transform
DL	Deep Learning
DNN	Deep Neural Network
DNS	Domain Name System
DVB-S2X	Digital Video Broadcasting-Satellite Second Generation Extensions
FFNN	Feed-Forward Neural Network
FFR	Federated Face Recognition
FL	Federated Learning
FR	Face Recognition
FSK	Frequency Shift Keying
GAN	Generative Adversarial Network
GNN	Graph Neural Network
HF-MAML	Hessian-Free Model Agnostic Meta-Learning
HLS	HTTP Live Streaming
HTTP(S)	Hypertext Transfer Protocol (Secure)
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
kNN	k-Nearest Neighbors
LLM	Large Language Model
LLR	Log-Likelihood Ratio
LMMSE	Linear Minimum Mean Square Error
LOS	Line-Of-Sight
LTE	Long-Term Evolution
LTI	Linear Time-Invariant
MIMO	Multiple Input Multiple Output
MITM	Man-in-the-Middle
ML	Machine Learning
MLP	Multi-Layer Perceptron
MoCo	Momentum Contrast

MPEG	Moving Picture Experts Group
OFDM	Orthogonal Frequency-Division Multiplexing
OL	Outlier Leveraging
ONC	Ocean Network Canada
OOD	Out-of-Distribution
OSI	Open Systems Interconnection
PFL	Personalized Federated Learning
PSNR	Peak Signal-to-Noise Ratio
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
QUIC	Quick UDP Internet Connections
R-D	Rate-Distortion
RNN	Recurrent Neural Network
SDR	Software-Defined Radio
SGA	Stochastic Gumbel Annealing
SIMO	Single Input Multiple Output
SNR	Signal-to-Noise Ratio
SSL	Sigmoid Scaled Logit
TCP	Transmissions Control Protocol
TLS	Transport Layer Security
UATR	Underwater Acoustic Target Recognition
UDP	User Datagram Protocol
UMa	Urban Macro
VAE	Variational Autoencoder

CHAPTER 1

Introduction

Deep learning (DL) [103] originated as a subfield of Machine Learning (ML). The field focused on the use of deep neural networks (DNNs) to model complex patterns in data. These were originally inspired by networks of biological neurons; the human brain. By stacking layers of functions, increasingly complex patterns in data can be extracted and used for a variety of tasks. The study of artificial neural networks (ANNs) has been ongoing since the 1960s, with some important foundational works by Rumelhart et al. [143] and Lecun et al. [102]. At this point, it has become clear that the potential of DNNs was limited by the availability of compute and data. The work of Krizhevsky et al. [101] popularized the training of DNNs, more specifically convolutional neural networks (CNNs) , for the use of image classification. Their CNN beat the second-best performing approach on ImageNet [45] by more than 10%.

The years since then have seen remarkable progress in terms of DL. Image classification was previously named, but another nice example is image segmentation [139]. This technology has proven vital in the development of autonomous vehicles. Furthermore, generative modeling techniques such as generative adversarial networks (GANs) [63] have become sufficiently capable to generate photorealistic images. Finally, and arguably the most influential, are the advancements in natural language processing.

At the time of writing, it is safe to characterize the current time period as an artificial intelligence (AI) boom or spring. The release of ChatGPT has introduced AI into the mainstream and made technical terms, such as large language models (LLMs), commonplace. Its introduction started an arms race among large tech companies to develop increasingly capable foundation models and kickstart a new generation of startups building apps based on LLMs. The impact of ChatGPT on many domains and, in particular, the domain of education cannot be understated. Based in part on these successes, the Nobel Prize in Physics was awarded to John J. Hopfield and Geoffrey Hinton for their work laying the foundation for modern DL [202]. In the same year, Demis Hassabis and John Jumper shared the Nobel Prize in Chemistry with David Baker [201]. They were presented the award for their work on building AI-based computational tools, such as AlphaFold [92].

These advancements highlight a key strength of DL, which is the ability to extract useful information from high-dimensional unstructured data. Using DNNs is most effective in domains where human feature engineering is challenging or outright impossible. One domain with a variety of such data-related challenges is telecommunications. Telecommunications has been a key enabling technology in the world today, in particular with the development of the Internet. It has been the most important development towards the age of information that we live in today. The ability to exchange information across the world in seconds has enabled rapid and immense scientific progress and completely transformed the lives of people around the world. In large part, these key technologies have enabled the current success of DL by providing the ability to collect large amounts of data and exchange ideas instantaneously.

A constant within networking and telecommunications is the need for expansion. New users and devices are constantly added to the network, each requiring a greater amount of bandwidth. Furthermore, the functions a network has to support also changes over time. The Internet of Things (IoT), for example, adds a great amount of complexity and demand to existing infrastructure. As the infrastructure grows and old devices are mixed with newer backward compatible devices, the complexity of maintaining this infrastructure drastically increases. The increase in complexity further opens up the risk of new cyber attacks. At a lower level, theoretical bandwidth limits as described by the Shannon-Hartley theorem [152] present issues. Intuitively, no more data can be transmitted over a connection than this limit. Note that many of these issues operate on and/or are caused by data. This data is often high-dimensional and, in case of the actual transmission, concerns actual physical signals. As DL excels in these types of domains, it is natural to ask the question what DL can contribute in ensuring the continued development of networking.

However, incorporating DL into the networking stack is an open problem. Networking systems have a variety of requirements difficult to merge with DL systems. These systems have to operate in a fast and energy-efficient manner. It can also be difficult to obtain sufficient high-quality labeled data. Furthermore, DL systems are black-box, making them difficult to interpret when implemented. It is thus imperative to look at improving DL systems for networking, or identifying areas in networking where these issues are sufficiently mitigated. It is important to note that much work has already been done on integrating DL into the networking stack [17, 135, 146, 170, 179, 192].

That is why the first question asked in this thesis is:

RQ1: *How can deep learning techniques be effectively applied across different parts of the networking stack?*

Here, 'effectively' does not simply apply to system performance. To make DNN network components usable, it is also necessary that they are as modularity, robust to changing environments, and small enough to operate within system requirements. These challenges differ based on where they are applied in the system.

1.1 The Layers of Networking

Before being able to effectively answer **RQ1**, it is necessary to specify the challenges that arise of DL for networking and telecommunications. Networking and telecommunications is an incredibly broad domain that is often abstracted into multiple separate domains. Each of these abstractions has its own tasks and corresponding challenges. The tasks for each abstraction are performed largely independently to ensure a simpler implementation and scalability. Thus, it is much more meaningful to discuss challenges and ways to address them per abstraction level. Doing so allows us to focus on specific solution avenues promising for each of the abstraction levels.

When discussing abstractions, it is common to use some form of conceptual model. The conceptual model with which the Internet was created is the Transmissions Control Protocol/Internet Protocol (TCP/IP) model. This model divides the Internet into four different layers, where each layer corresponds to a level of abstraction. These layers define different steps in the transmission of data from one point to the other. While the TCP/IP model is used in practice, a model that more clearly defines the separation between different functionalities is the Open Systems Interconnection (OSI) model [158, 200]. This model defines seven layers of abstraction where each layer is clearly defined in terms of its functionalities. As such, this model allows for clear explanations of the various functionalities within networking. A figure of the OSI model for transmission and reception is given in Figure 1.1. The scope of each layer is as follows:

7. The **Application Layer** describes the top-level protocols such as the Hypertext Transfer Protocol (HTTP) and Dynamic Adaptive Streaming over HTTP (MPEG-DASH, where MPEG refers to the Moving Picture Experts Group). It is arguably the most diverse layer, as it describes the protocol used by applications to communicate with each other. Thus, many implementations of a wide variety of functionalities exist in this layer.

6. The **Presentation Layer** describes the translation of application data to a format that is presentable on the Internet. This includes functions such as character coding, (de)compression, and de/encryption. Challenges often directly relate to the aforementioned topics and essentially entail how to ensure data are formatted properly, safely, and efficiently.

5. The **Session Layer** discusses the approaches used to establish, maintain, and terminate connections independent of addresses. The most common implementation of session layer functionality is through the use of cookies.

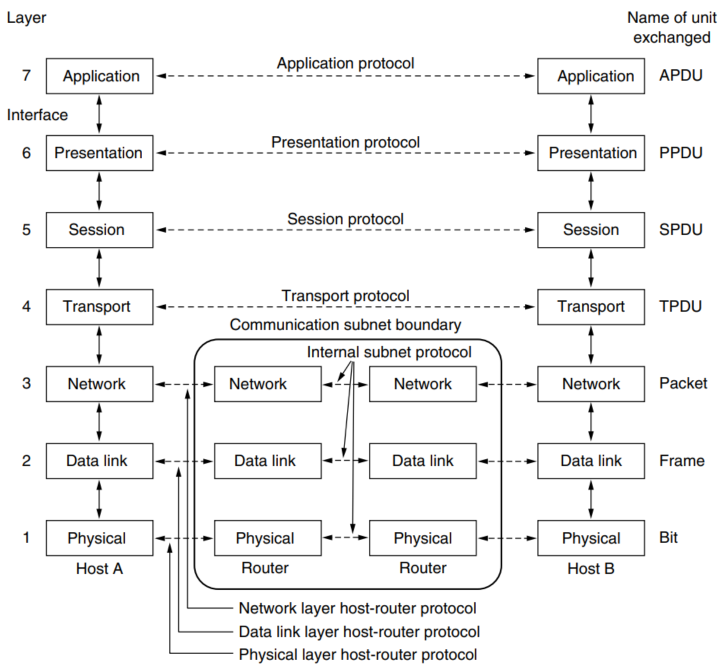


Figure 1.1: The OSI model for transmitting and receiving data [158].

These allow storing user information locally to enable quick resumal of browsing. Key challenges in this area often relate to security.

4. The **Transport Layer** manages the transmission of data from one point to the other. Its key protocols are User Datagram Protocol (UDP) (stateless transmission) and TCP (stateful transmission). The core tasks of the transport layer are to ensure that data arrives at the right destination, in a timely fashion, while maintaining data integrity. Opportunities of DL for the transport layer lie in creating smart algorithms to more efficiently transport data.

3. The **Network Layer** manages the process of routing a packet through a network. The most important in this layer is the IP protocol, which governs, for example, routing packets through Internet. These protocols determine the path a packet should take to get from one end of the network to the other. Note that routers within the IP protocol do not have to be physical routers. The key opportunities for DL lie in creating intelligent routing algorithms to improve the flow of packets.

2. The **Link Layer** describes how to transport data between two ends of a physical link. Note that a physical link may not be reliable or congested. This layer deals with managing what data gets sent first and ensuring that data eventually arrives. The Ethernet protocol is one of the most well-known

algorithms in this layer. A key challenge again lies in estimating which links are congested and adapting intelligently.

1. The **Physical Layer** prescribes at the hardware level how to perform a transmission given that a link exists. It deals with the physical medium, e.g. how to correct distortions that occur when sending a signal over the air or over a link. Many of the components in this layer are hardware-based. Key opportunities lie in the intelligent use of software-based components to more efficiently perform transmissions.

Even within this model, it often occurs that some of the layers overlap. For example, it can sometimes be unclear in practice where the separation between the application layer, the presentation layer, and the session layer lies. The same often goes for the link layer and the physical layer. These overlaps can result in vulnerabilities, as will be discussed later.

In general, these layers define a set of largely separate technologies that enable sending packets from point A to point B. For brevity's sake, the layers will be discussed in three sets of layers. Of these three sets of layers, only two have been considered in-depth. The part not discussed is the combination of the network layer and the transport layer. These layers are highly relevant within the context of DL for networking but outside the scope of this thesis. The first part discussed will be the combination of the physical layer and the link layer. The second part discussed will be the combination of the application layer, the session layer, and the presentation layer. We note that these sets of layers are logical choices to combine, given their lack of separation in the TCP/IP model. We will therefore refer to this combination as simply the application layer.

Although challenges and technologies differ substantially across layers, the modeling concepts within DL are often similar.

RQ2: *What are the advantages and challenges of implementing deep learning in the physical layer versus the application layer?*

To get an overview of how this thesis plans to address both questions, we introduce the challenges and approaches of the two different parts identified.

1.2 Physical Layer Challenges

As introduced previously, the first part of this thesis considers the physical layer of telecommunications. The physical layer of communications deals with transmitting information across a physical link. It assumes a stream of bits that need to be transmitted between two neighboring points in a network. As these two points are usually not in the same place, the most important task is to encode digital data onto a physical medium. Such a medium can be many things, with common mediums being cable or over-the-air. A signal passing through a medium always arrives differently than when it is transmitted. Thus, a key implication of the transmission task is that the physical layer has to ensure that the bits arrive correctly at the receiver.

Physical layer functionalities are often implemented using dedicated hardware circuits. The hardware implementation allows these components to be extremely fast and energy-efficient. However, this limits the flexibility and adaptability of components of the physical layer. That is why, similar to the networking layer, a software-based alternative has been proposed. This is often referred to as a software-defined radio (SDR). By implementing physical layer functionalities in software, arbitrarily complex functionalities can be implemented. The main concern is that an SDR is unlikely to match the speeds and energy efficiency of dedicated hardware. The added flexibility may address challenges that are difficult to handle with hardware-based components.

There are various challenges that remain within the physical layer. As the scope of this thesis mainly focuses on wireless systems, the challenges discussed are directly related to wireless mediums. An important challenge in wireless networks lies in the ability to scale up the speeds of the network. Applications such as, for example, the transmission of holograms potentially require terabit per second speeds. The speed of a wireless connection is directly related to the amount of bandwidth. However, bandwidth within an area is a limited resource. With current hardware, there is a fixed amount of bandwidth to divide among all parties. If multiple parties naively broadcast within the same bandwidth, all transmissions are distorted. A more efficient use of existing bandwidth would alleviate this issue. This can be done through a concept called spectrum sensing. Using DL to perform spectrum detection would allow devices to check for unused bandwidth and use it adaptively [62].

However, the combination of software and DL can be considered in a wider context. An important question to ask is whether it is sufficient to implement functionalities as single models. A traditional result in DL is that performance improves as more parts of the pipeline are added to the model and trained end-to-end. The work of [84] provides a more comprehensive view of the purpose of DL in 6G and beyond telecommunications. They envision various stages of the sender-receiver pipeline being replaced with AI. The early stages involve replacing only parts of the pipeline with AI. The spectrum sensing mentioned earlier would fit into this framework. Later stages could involve replacing the entire transmission and reception process with AI components, such as DNNs. This would enable fully leveraging end-to-end learning with potential performance benefits as a result.

To progress towards this goal and assess its benefits, it is first necessary to develop foundational DL techniques. These techniques should demonstrate whether full end-to-end communication is viable or beneficial. This question is of interest and worked on by the broader community of DL for the physical layer [84]. To that end, we contribute in three problems towards progressing the state of DL for the physical layer. These three chapters each address one of many open questions in how to implement DL for the physical layer. We have divided this into three chapters. The first addresses a flexibility issue that results from modeling the receiver as a single DNN. The second leverages the phase properties of transmissions to reduce the parameter count of deep receivers. The third examines the use of contrastive learning for modeling underwater acoustics.

1.2.1 Overview of the Physical Layer Chapters

The first three chapters examine research on deep learning applications in the physical layer of communication systems. These chapters offer a physical layer perspective that aligns with our primary research questions. Specifically, they identify key challenges within the physical layer, addressing **RQ1**, and facilitate comparisons from the physical layer perspective, as outlined in **RQ2**.

In **Chapter 2**, we discuss our first work on deep neural receivers. To optimally use a given spectrum, a multitude of constellations is usually used. A constellation in this case refers to a set of phase and amplitude shifts that each translate to a different set of bits. By encoding elements of this constellation onto a carrier wave, an analog wave can be encoded to transmit digital data. Including more combinations of phase and amplitude shifts results in a higher spectral efficiency but a higher sensitivity in terms of distortions.

However, previous work, such as the work by [82], considers a separate neural receiver for each type of constellation. The core complexity of a receiver is often in performing channel corrections; demapping is trivial when the distortions are known. Thus, sharing parameters for different demapping settings could be highly beneficial.

In [82] a scheme was proposed to efficiently model the family of 4^n-Quadrature Amplitude Modulation (QAM). We extend their work by introducing a module that decouples the configuration of the constellation from the neural network weights. We prove that the inclusion of our module allows for strictly more general neural receivers. This is achieved by decoupling the constellation characteristics from the bit mapping through the use of an intermediate representation. We also provide a method for modeling the representation for QAM and Amplitude and phase-shift keying (APSK) constellations. Through this representation, weights can be shared across similar constellations. Finally, we demonstrate that our approach can provide benefits to the training of automatic modulation recognition (AMR) models by including a demapping loss function.

Through our work, we make DNN receivers more flexible by reintroducing modularity while provably maintaining performance benefits.

In **Chapter 3**, we continue our work on deep neural receivers, but address a different angle. A lot of data is processed in digital communications, so receivers generally need to work fast and in an energy-efficient fashion. A common concern with DNN-based receivers is that they are too big, too slow and consume too much energy to be viable in such a high-throughput environment. One avenue to address this issue is through reducing the size of receiver models.

We opted to look into creating more efficient DNN architectures through the use of inductive biases. The ground work for this approach is the work of [42] on group equivariant DNNs. They were the first to propose a DNN that incorporates known symmetries directly into the DNN construction. This removes the need to learn these effects from the data and allows the network to use patterns across different group elements.

We identified that deep neural receivers should be equivariant with respect to the phase-of-arrival or the initial phase of a receiver. The initial phase of a signal is commonly assumed to be 0 or completely omitted in the analysis; the main information carrier is the phase shift between two sequential moments within a signal. We propose a method to construct equivariant DNN receivers and use previous results to prove that our equivariant approach is the most general solution possible. In addition, we propose an improvement to DeepRx [82] in terms of parameter efficiency based on advancements in CV and natural language processing (NLP). We demonstrate that the inclusion of relative phase equivariance allows significant improvements in bit error rate (BER) at similar numbers of parameters.

Through this work, we laid the groundwork for including inductive biases in terms of the channel estimation component of the physical layer. This allows for reductions in the sizes of the deep receivers without impacting performance.

In **Chapter 4**, we consider the issue of artificially induced sounds in an underwater environment. The increased levels of noise could negatively affect the underwater environment, and thus it is prudent to monitor the causes of noise pollutions. To do so, sounds are recorded through the use of passive acoustic monitoring. Afterward, they can be classified by hand or by using audio-based DNNs specifically built for underwater acoustics.

However, the limited availability of labeled data limits the use of supervised techniques. We note that large amounts of unlabeled data exist and thus propose the use of contrastive learning to build a backbone model for underwater acoustics. We demonstrate that our prototype can learn features that result in competitive performance to those generated by supervised contrastive learning (CL). This provides a valuable step for underwater acoustic target recognition (UATR). It also potentially provides a step for communication in an underwater setting. Simulating realistic environments is challenging in an underwater setting. The ability to generate backbone models from underwater acoustic data may be valuable in the generation of realistic underwater channel coefficients.

This work thus explores possible extension to challenging environments that may be better solved with DL techniques.

1.3 Application Layer Challenges

In the previous part, we address several challenges in the physical layer that move towards making a complete end-to-end AI communications pipeline. A natural question at that point is why the vision of a fully end-to-end communications pipeline should restrict itself to the physical layer. In fact, many advances in DL are either applicable to the application layer and/or straightforwardly integrated with it. We contribute to the broader domain of DL applications through three works.

First, note that DL has immense potential for automatic analysis of data streams. In that context, it is important to evaluate the current application layer protocols. These protocols may have vulnerabilities that were difficult to exploit without the use of DL. Such studies are important to ensure eventual DL communication systems are secure. To that end, we study one such vulnerability in Chapter 5. To build on the work in security, we also look at the use of DL for compression. In particular, we build on the existing neural image compression literature in Chapter 6.

In the context of fully end-to-end learning, a wider interpretation of the application layer is also looked into. To use DL/ML models, it is necessary to provide access to them to the parties interested. As these models can be too large for small computers, it is necessary to host them from some server and allow users to run the inference remotely. Just serving a model can easily be done with the existing networking infrastructure using HTTP. Expanding this process is important in the context of fully end-to-end communication systems. In this context, it can make sense to think about the development of protocols specific to the requirements of ML in a networking setting.

A key method to perform distributed learning in a secure way is called federated learning (FL). In line with both the work on the physical layer and the security angle, we investigate fairness in FL. We specifically do so under face recognition (FR), which is a highly sensitive domain in terms of data. We present this work in Chapter 7.

1.3.1 Overview of the Application Layer Chapters

In **Chapter 5**, we investigate a security issue that arises due to the interaction between the functionalities of the presentation layer and the application layer. Video streaming on platforms such as YouTube is performed using application layer protocols such as MPEG-DASH. Video is generally a data-intensive format in networking, so these protocols split up videos into segments that can be downloaded as needed. In this way, only the parts that a user is watching are transmitted, resulting in a large gain in efficiency. Ensuring the transmission is private is handled by the presentation layer, where each segment is encrypted.

However, work by Schuster et al. [150] identified that the dynamic nature of MPEG-DASH results in a vulnerability that compromises the confidentiality of the connection. Without touching the encryption, the video ID can be identified using a CNN classifier. We build upon this work using deep-metric learning and demonstrate that the patterns are identifiable to such a degree that a handful of examples are sufficient to identify which video stream it is.

Through this work, we show that DL simplifies exploiting existing vulnerabilities between communication layers that were previously challenging to exploit.

In **Chapter 6**, we continue with the compression session layer component and specifically look at neural image compression. The concept of neural image compression treats a traditional image compression codec as an encoder/decoder structure. The encoder model transforms an image into a quantizable representation that can be restored in a lossy fashion by the decoder. Neural image compression has been shown to achieve a better trade-off between the quality of the restored image (measured by, e.g., the peak signal-to-noise ratio (PSNR)) and the bits per pixel (BPP) required to store the image. The downside of neural image compression is the requirement of maintaining a neural codec and the associated processing time.

The work of Yang et al. [187] proposed a method to perform a form of post-processing on the generated latents. They demonstrate that applying their stochastic Gumbel annealing (SGA) method to individual images to refine the latent allows for an even better trade-off between PSNR and BPP at the cost of increased compute.

However, we demonstrate that the use of the atanh function in weighting the Gumbel probabilities is suboptimal. We propose a set of functions that exhibit analytical properties and dub them SGA+. Among these functions is the sigmoid scaled logit (SSL) function, which allows interpolation between all valid Gumbel weighting functions through the use of the hyperparameter a. We validate our approach empirically and demonstrate through a wide set of experiments that our approach converges faster than SGA and achieves a better PSNR/BPP trade-off earlier.

Overall, effective neural image compression techniques are highly interesting with an increasing capability of edge devices to run neural codecs locally. One of the key benefits of better compression techniques lies in reducing congestion on the Internet by reducing the overall size of traffic being sent around.

In **Chapter 7**, we present work on FL for FR. FL, as a form of distributed private learning, is highly interesting from the perspective of networking. Potentially, the data used to update a model can be significantly larger than the model update itself. It may also be outright infeasible to send the data to a central server due to size constraints and/or privacy issues. To this end, we investigate the use of FL in the field of FR. It should be noted that the identification of faces using DNNs has been solved in a centralized setting.

However, multiple retractions of the datasets that underlie key papers in this domain have raised questions about the future of FR. To that end, it is interesting to look at the use of FL in FR to overcome these privacy issues. An issue in FR is that it is reasonable to assume that, e.g., identities are not shared across parties. It is also reasonable to assume that there is a high degree of data heterogeneity across different clients. That is why we propose the use of federated meta-learning within the domain of face recognition. We demonstrate that, especially under heterogeneous splits, the use of metalearning can improve the overall performance per client. We also show that the benefits of performance improvements mainly benefit the weakest clients, resulting in a smaller variance in performance between clients.

1.4 Overview of the Dissertation

Finally, an overview of the layout of the dissertation is provided. The part, chapter and reference overview can be found in Table 1.1.

Part	Chapter	Reference
I	2	[1, 2]
	3	[4]
	4	[6]
II	5	[5]
	6	[7]
	7	[3]

Table 1.1: The part and chapter layout of the thesis with references.

Ch. 2: Gansekoele et al. [2] I contributed the original idea to build a multi-neural receiver, theory, implementation, ablations, experiments, and writing.

Ch. 3: Gansekoele et al. [4] I contributed the original idea to build a relative phase equivariant receiver, theory, implementation, ablations, experiments, and writing.

Ch. 4: Hummel et al. [6] The research was led by Hilde Hummel. I assisted in conceptualizing the approach, theory, ablations, experiments, and writing.

Ch. 5: Gansekoele et al. [5] I contributed the original idea to apply deep metric learning to video stream fingerprinting, theory, implementation, ablations, experiments, and writing.

Ch. 6: Perugachi-Diaz et al. [7] The research was led by Yura Perugachi-Diaz. We both contributed original ideas, theory, implementation, ablations, experiments, and writing.

Ch. 7: Gansekoele et al. [3] I assisted in developing the original idea to apply federated meta-learning to FR, theory, and implementation. I contributed ablations, experiments, and writing.

Part I

Deep Learning for the Physical Layer

CHAPTER 2

Joint Demapping of QAM and APSK Constellations Using Machine Learning

Based on [2] which extends [1]:
Arwin Gansekoele, Alexios Balatsoukas-Stimming, Tom Brusse, Mark Hoogendoorn, Sandjai Bhulai, and Rob van der Mei
Joint Demapping of QAM and APSK Constellations Using Machine Learning
Conference version published in the 2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)
Journal extension published in the 6th issue of the IEEE Open Journal of the Communications Society (OJCOMS)

Abstract

As telecommunication systems evolve to meet increasing demands, integrating DNNs has shown promise in enhancing performance. However, the trade-off between accuracy and flexibility remains challenging when replacing traditional receivers with DNNs. This paper introduces a novel probabilistic framework that allows a single DNN demapper to demap multiple QAM and APSK constellations simultaneously. It is demonstrated that the framework can exploit hierarchical relationships in families of constellations. The consequence is that we need fewer neural network outputs to encode the same function without an increase in BER. The simulation results confirm that the framework approaches the optimal demodulation error bound under an additive white Gaussian noise (AWGN) channel for multiple constellations. Under 3rd Generation Partnership Project (3GPP)-compliant orthogonal frequency-division multiplexing (OFDM) fading channels, it is as accurate as a neural receiver operating on just one modulation type. Thereby, the framework addresses multiple important issues in practical neural receiver design. These include improvements in computational efficiency, a reduction in memory overhead, and an improved adaptability in dynamic environments.

2.1 Introduction

Digital communications has proven to be an essential technology in the era of information. A common communications pipeline consists of many components that need to match on both the sender and receiver sides. These include components such as filters, coding, and modulators. Recently, the use of ML has been investigated and has already been included in the 5G standard. DNN approaches, specifically, replace (parts of) the communications pipeline with a neural network. One approach is to replace the entire sender-receiver pipeline with a DNN [13, 14, 47, 56, 57, 99, 133, 180, 182]. Doing so allows the DNN to adapt the type of constellation used based on the channel conditions seen during training. However, since the entire pipeline is modeled by one DNN, the modularity of the pipeline is destroyed. Modularity can be hugely beneficial when designing practical systems as it simplifies the reuse of components in a setting where space is limited. That is why a second line of work opts to perform only channel estimation and equalization using DNNs [22, 34, 64, 114, 125, 129, 155, 174, 175, 178, 191, 193]. As channel estimation and equalization are traditionally performed with data-driven techniques, replacing these components with a DNN is intuitive. However, assumptions on the output distribution of the DNN have to be made in this case. That is why a third line of work proposes to also add the demodulation component to the DNN, along with equalization and estimation [33, 82, 100, 136–138, 196].

Digital demodulation techniques generally assume that the received I/Q samples are perturbed by white Gaussian noise. This assumption becomes unnecessary by including the demodulator as part of the receiver design. This inclusion can potentially improve performance. The work of Honkala et al. [82] demonstrated that this approach can reduce the number of pilots necessary, while achieving a performance similar to that when the state channel information is known. However, this approach introduced a new problem. Traditional demappers for digital communications are flexible in the settings they can use. For example, if needed, the constellation type or bit mapping can be adjusted for every individual data block with traditional demappers. Flexibility in bit mappings allows receivers to operate with senders from previous generations, while supporting multiple constellation types is imperative for adaptive bandwidths.

Without further adjustments, it becomes necessary to train a new DNN receiver for every mapping and constellation type. If the implementation choice is made to keep all sets of parameters in memory, the memory usage scales linearly with the number of mappings and constellation types. On the other hand, loading parameter sets as required results in significant computational delays. Both options are impractical and unnecessary, as the main receiver task of channel estimation is shared between constellations. What makes these inefficiencies even more problematic is that naively applying DNNs already does not meet the low-power and high-throughput requirements of modern communication systems. While Honkala et al. [82] propose a scheme to model all variants of the 4^n-QAM, their scheme does not support different bit mappings. It also achieves worse performance after combining multiple types of different 4^n-QAM constellations.

To address these issues, this work proposes a framework for building deep neural receivers that can demap multiple types of constellation. The modularity of the approach makes it applicable to any neural receiver that operates on some form of frequency and/or amplitude keying. The framework enables features present in traditional receivers, such as flexible symbol-to-bit mappings and multiple demodulation types. It does so while maintaining the gain enabled by neural receivers. The core contributions of the work as follows can be characterized as follows:

1. An extension to existing deep receivers is proposed that allows them to have flexible symbol-to-bit mappings without loss of performance.

2. The hierarchical relationship from Honkala et al. [82] is generalized to arbitrary constellations. In addition, a hierarchical approach for APSK is proposed that can reduce the memory requirement to model families of constellations.

3. To the best of our knowledge, this work is the first to construct a deep neural receiver that can demap different constellations simultaneously with (close to) no loss in performance. This claim is empirically validated.

This chapter is based on an extended version of Gansekoele et al. [1], which was originally presented at the ICMLCN 2024 conference and was included in the proceedings. The current version extends the conference version as follows. First, a proof is included for a claim in the original work. A proof was included that the framework is strictly more general than a neural receiver that directly predicts bit log-likelihood ratios (LLRs). Second, a scaling factor was proposed to correct for an issue that occurs when demapping multiple different 4^n-QAM over fading channels. Third, a link with AMR is presented. The extended version demonstrates how the framework can improve the performance of existing AMR systems. Finally, the evaluation suite is expanded to include 3GPP-compliant OFDM receivers.

2.2 Related Works

The idea of using DNNs to perform channel estimation and demapping has been studied over the years [135, 192]. Hoydis et al. [84], You et al. [189] and Zhijin Qin et al. [197] provide a vision on the role that AI could play in 6G. Hoydis et al. [84] specifically identified three steps: AI replacing individual components in the sender-receiver pipeline, AI replacing multiple components at once, and finally, AI replacing the entire pipeline and/or designing the pipeline itself. The core benefit of AI they envision is the ability to design systems in a data-driven manner. A lot of development time goes into building systems that are modular and effective for different cases. The use of artificial intelligence could alleviate these issues by automating (parts of) pipeline design. This paper contributes to their vision by introducing modularity into a pipeline that is entirely data-driven.

Many preliminary works followed the first approach laid out in this vision. In the work of Lin et al. [114], the authors built a neural network demodulator for binary phase shift keying (BPSK), binary amplitude shift keying (BASK), and binary frequency shift keying (BFSK) and demonstrated its potential compared to classical approaches. In similar work, Zhang et al. [193] built a BPSK demodulator using a 1D-CNN. Furthermore, in the work by Mohammad et al. [125], they used a CNN to demodulate frequency shift keying (FSK) under AWGN and Rayleigh fading. These works were still fairly simple and only addressed relatively simple constellations. In work by Neev Samuel et al. [129], the authors proposed DetNet for detection in OFDM systems. This is one of the first works to apply a DNN to OFDM system detection. The work of Wang et al. [174] was among the first to perform over-the-air demodulation. They used DBN-SVM and AdaBoost-based models to do so. [178] introduced the use of a CNN + recurrent neural network (RNN)-based network for demodulation. The work of Soltani et al. [155] and Yue Hao et al. [191] was among the first to perform channel estimation and equalization using a DNN. They used a super-resolution-based DNN to do so. The work of Balevi et al. [22] proposed following up the DNN denoiser by conventional LS for improved demodulation. The work of Cammerer et al. [34] investigated the use of graph neural networks for channel decoding. Work by Goutay et al. [64] built on top of traditional linear minimum mean square error (LMMSE) systems using a 1D-CNN to predict second-order statistics and the channel aging effect. He et al. [76] used approximate message passing with graph neural networks for multiple input multiple output (MIMO) detection. All of these works introduced various components of the signal demodulation pipeline, such as improved modeling in increasingly complex and realistic channel models. They provide essential steps towards the development of fully end-to-end neural receiver systems. The development of the framework relied on these works and/or the framework can be directly applied to these works. For example, DetNet can be extended with this framework to incorporate the properties discussed in this work. Furthermore, work by Hanna et al. [74] proposed a dual-path network that takes into account the characteristics of the channel to improve classification. Their approach is similar to the effect of this work on automatic modulation recognition.

The work of Xuanxuan Gao et al. [185] and Yue Hao et al. [191] was among the first to approach both channel estimation and demodulation with a neural OFDM receiver pipeline to improve performance. These lines of work share similarities with the detection networks described before. The work of Honkala et al. [82] proposed DeepRX, which is a deep receiver that is able to approach the correction performance of correcting with the ground truth channel state information in single input multiple output (SIMO) OFDM systems. They achieved this by incorporating knowledge of modern neural network design to improve receiver capacity. The work of Korpi et al. [100] extended DeepRX to a MIMO setting and the work of Huttunen et al. [87] investigated training parts of the sender to learn beamforming jointly with the channel estimation. Further work on MIMO Cammerer et al. [33] proposed the use of a graph neural network (GNN) to extend works such as DeepRX to a MIMO setting with arbitrary sets of users. Further improvements by Raviv and Shlezinger [137] proposed data augmentation strategies to improve the training of deep receivers. Other approaches include [136] who investigated power-efficient deep OFDM receivers, work by Zhao et al. [196] that built a neural receiver on time-domain OFDM samples, and work by Raviv et al. [138], who used a metalearning approach to quickly adapt a classifier to new channels. The presented framework builds on these works specifically by making these deep receivers more flexible. Our approach can be added as a module to all of these works. Doing so adds to the properties that are discussed in this work.

Many authors have also already begun investigating the possibility of replacing the entire communication pipeline with learnable components. The preliminary work by O'Shea and Hoydis [133] first proposed envisioning the sender-receiver pipeline as an autoencoder (AE). By doing so, they showed it possible to perform geometric constellation shaping as a side-effect of optimizing channel capacity directly. The work of Dorner et al. [48] validated this approach in an over-the-air setting using two SDRs. They achieved BER scores close to conventional systems and showed that the constellations were realistic. The work of Felix et al. [57] extended these systems to modulation in the frequency domain in the form of OFDM. The work of Xisuo Ma et al. [182] adds to this by jointly learning optimal pilot patterns and a channel estimator for MIMO. Although all of these works assume a differentiable channel model, the work by Fayçal Ait Aoudia et al. [56] proposed a method that allows model-free end-to-end learning that also works with non-differentiable channel models. The work of Ait Aoudia and Hoydis [13] built on this by demonstrating that end-to-end learning of the sender and receiver allows for pilotless communication without loss of performance. Korpi et al. [99] arrived at similar conclusions in their work. In a similar work, Ait Aoudia and Hoydis [14] learned a transmit and receive filter, constellation geometry, and associated end-to-end bit labeling. They achieved rates similar to those of an OFDM system under 3GPP-compliant channel models. An important step that still needs to be taken is the inclusion of different modes in these end-to-end pipelines. Adding the module presented in this work to these pipelines could enable constellation switching in a manner that respects predefined hierarchical constraints.

2.3 Methodology

The symbol demapping or demodulation process is the process of extracting information from some information-carrying signal. Given a sequence of received IQ samples $\mathbf{y}$, extraction of the underlying set of information bits $\mathbf{b}$ is desired. This is done by generating a set of bit estimates $\hat{\mathbf{b}}$ based on $\mathbf{y}$. These estimates can be hard or soft; in the latter case, an LLR is computed for each bit that a decoder can use to refine the decision. This process could be performed with a DNN f with a set of parameters θ, in which case the LLR can be defined as

$$\log \frac{P(b_{ij} = 1 \mid \mathbf{y}, \theta)}{P(b_{ij} = 0 \mid \mathbf{y}, \theta)} = \left(f_\theta^{(LLR)}(\mathbf{y}) \right)_{ij}. \tag{2.1}$$

The probability $P(b_{ij} = 1 \mid \mathbf{y}, \theta)$ refers to the probability that bit ij is a 1. The indices ij refer to the i-th IQ sample in the sequence and the j-th bit in the bit string. This DNN predicts the log-likelihood ratio of all bits in the information sequence. This formulation can be optimized by using a sigmoid function followed by a binary cross-entropy loss function. It can then be trained by applying some form of gradient descent on this loss quantity. Note that minimal assumptions have been made on the underlying data distribution. This allows for the finding of an optimal strategy for channel estimation, equalization, and demodulation.

A core issue with this approach is the lack of flexibility. All complexity is abstracted within the architecture and parameters defined by the set of parameters θ. As a DNN is a highly non-linear function, there is no straightforward way to change, for example, the bit mapping or the constellation used in demodulation.

2.3.1 Mapping Independence

Mapping independence is proposed as the ability to change the association between constellation points and the bit assignment. This concept used to be trivially true in traditional receiver design. Previously, demapping was simply a module in a fully modular pipeline. When directly predicting the bit LLRs with a DNN, it becomes less obvious how to change the constellation and association between constellation points. The constellation and association of points with the bit string is hardcoded in the neural network weights. Other than a few special cases, it is necessary to build a new DNN receiver if a different association between constellation points and bit string is desired. Hence why this work defines (bit) mapping independence as a desirable property for practical DNN receivers.

When demodulation is performed without a DNN, the log-likelihood ratio of each bit can be computed exactly based on the constellation points. In this procedure, half of the constellation points are associated with a bit value of 1 and the other half with a bit value of 0 for each bit in the information string. The LLR can then be computed on the basis of the distances of the received sample to each constellation point and their bit assignment as

$$\log \frac{P(b_{ij} = 1 \mid \mathbf{y})}{P(b_{ij} = 0 \mid \mathbf{y})} = \log \left(\frac{\sum_{\substack{k=1 \\ (\mathcal{M}_b(\mathbf{s}_k))_j = 1}}^{|S|} \exp\left(-\frac{1}{N_o}|y_i - s|^2\right)}{\sum_{\substack{k=1 \\ (\mathcal{M}_b(\mathbf{s}_k))_j = 0}}^{|S|} \exp\left(-\frac{1}{N_o}|y_i - s_k|^2\right)} \right). \tag{2.2}$$

Here, c_{y_i} is the constellation point that corresponds to the i-th I/Q sample of the sequence $\mathbf{y}$. Variable s_k is defined as the k-th constellation point that belongs to a set S of possible constellation points given a certain constellation. In the case of 16-QAM, the set S has 16 constellation points, for example. The mapping $\mathcal{M}_b$ was also defined as a function that maps a symbol to a bit string. The outcome of this function can be indexed, which is done in the form of $\left(\int_{||}\right)_j = 1$. This means that the j-th bit of the bit string to which symbol s_k is mapped should be 1.

This form of demapping is optimal when it is assumed that the I/Q samples have a Gaussian distribution centered at their ground-truth constellation point. This approach also has a bit mapping that can be easily adjusted within the module. When a different bit assignment is needed under this formulation, only one thing needs to be changed. This is the assignment of which constellation points should be 1 and which 0. To achieve complete mapping independence, it is necessary to have an explicit probability measure over all possible constellation points. To achieve this in a DNN-based approach, the constellation point probabilities can be directly modeled as

$$P(c_{y_i} = s_k \mid \mathbf{y}, \theta) = \left(f_\theta^{(\text{symbol})}(\mathbf{y}) \right)_{ik}. \tag{2.3}$$

Given the probability distribution over the constellation points, the association between the constellation points and the bit predictions can be defined. Note that the formulation has been restricted to bit mappings that are bijective for brevity's sake. This means that every constellation point maps to exactly one-bit string and that each bit string that is mapped is unique. The bit probability can then be defined as

$$P(b_{ij} = 1 \mid \mathbf{y}, \theta) = \sum_{\substack{k=1 \\ (\mathcal{M}_b(\mathbf{s}_k))_j = 1}}^{|S|} P\left(c_{y_i} = s_k \mid \mathbf{y}, \theta\right). \tag{2.4}$$

Equation (2.4) utilizes that the probability $P(b_{ij} = 1 \mid \mathbf{y}, \theta)$ corresponds to the probability that $\mathbf{s}_k$ should be demapped as one of the symbols that maps to bit j. Thus, it can be computed by adding the constellation point probabilities of points that map to 1 at the index j in the bit string. An explicit term for the probability of each constellation point has been derived. Thus, a mapping-independent formulation is presented. To demodulate with a different mapping, one can change $\mathcal{M}_b$ to a desired new mapping $\hat{\mathcal{M}}_b$. The appropriate probabilities are then directly obtained without adjusting the DNN weights. The described procedure performs an exact computation of the LLR. However, mapping independence can also be achieved with an approximate LLR. This can be computed, for example, by taking the maximal probabilities of constellation points corresponding to a bit value of 1 and 0. Doing so potentially reduces system performance while allowing higher throughput. These effects would be more noticeable the larger the constellation is.

2.3.2 QAM Representation

Although the property of mapping independence is important, it does not address the hierarchical structures in many of the constellation families. Figure 2.1 depicts the hierarchical structure. Here, a quadrature phase shift keying (QPSK) constellation is overlaid by a 16-QAM constellation. There are two important observations here. First, there is an identical symmetry component for both QPSK and 16-QAM. For QPSK, only two bits are needed to encode the location of a point; one bit is used to determine the sign of the real axis, and the other is used to determine the sign of the imaginary axis. As such, it is not necessary to encode a probability for each individual symbol. Two binary probabilities, one for each sign, are sufficient to compute all symbol probabilities. Second, there are shared decision boundaries between QPSK and 16-QAM. The bit mappings depicted here allow us to define the first two bits of the 16-QAM constellation. These bits correspond to the sign of the real and imaginary components. Thus, these bits can be shared with the QPSK constellation, an observation made previously by [82]. Having a form of binary representation thus makes sense in the context of 4^n-QAM constellations. Therefore, a form of shared binary representation that respects the hierarchical relationship between different QAM constellations is proposed. This representation is referred to as r and a mapping $\mathcal{M}_r$ is defined. This mapping maps a constellation point to the binary representation r. The binary representation can be defined in a manner similar to [82] and Figure 2.1. The neural network is then defined as

$$\log \frac{P(r_{ij} = 1 \mid \mathbf{y}, \theta, r_{i1}, \ldots, r_{i(j-1)})}{P(r_{ij} = 0 \mid \mathbf{y}, \theta, r_{i1}, \ldots, r_{i(j-1)})} = \left(f_\theta^{(\mathrm{repr})}(\hat{\mathbf{x}})\right)_{ij}. \tag{2.5}$$

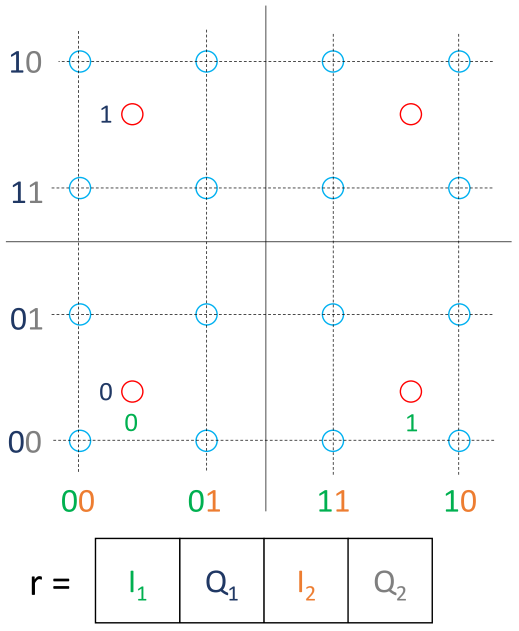


Figure 2.1: An example of QPSK and 16QAM overlaid. The representation is determined by counting from left to right and from bottom to top in Gray code. Afterward, interleave the in-phase and quadrature components to get a hierarchy. The representation values match the position in the representation by color.

Note that no assumption on the independence of each representation bit is made. It is also not assumed that $\mathcal{M}_r$ is bijective. It could be desirable to have a binary representation in which elements of the representations are dependent on prior elements in the representation. The formulation in (2.5) is the final formulation that is used for all experiments. To map the representation bits to a bit string, the symbol probabilities are first computed as

$$P(c_{y_i} = s_k \mid \mathbf{y}, \theta) = \prod_{j=1}^{R} P(r_{ij} = (\mathcal{M}_r(s_k))_j \mid \mathbf{y}, \theta, r_{i1}, \ldots, r_{i(j-1)}). \tag{2.6}$$

Here, R is introduced as the total size of the representation. The symbol probabilities can be computed by sequentially multiplying all the elements in the representation for a given symbol. For the resulting receiver, a sequence of defined formulas can be used. The sequence of (2.4), (2.5), and (2.6) is used. Consequently, all variants of 4^n-QAM can be modeled with this formulation. It also incorporates mapping independence, improving over [82]. For 4^n-QAM, the special case where $\mathcal{M}_r$ is an extension of $\mathcal{M}_b$ occurs. For this case, the following theorem holds assuming an exact LLR computation.

Theorem 2.1. *$\mathcal{M}_b$ is extended by $\mathcal{M}_r$* $\rightarrow P(b_{ij} = 1 \mid y, \theta) = P(r_{ij} = 1 \mid y, \theta, r_{i1}, \ldots, r_{i(j-1)})$.

Proof. $P(b_{ij} = 1 \mid \mathbf{y}, \theta) = \displaystyle\sum_{\substack{k=1 \\ (\mathcal{M}_b(\mathbf{s}_k))_j = 1}}^{|S|} P\left(c_{y_i} = s_k \mid \mathbf{y}, \theta\right)$

$$= \sum_{\substack{k=1 \\ (\mathcal{M}_b(\mathbf{s}_k))_j = 1}}^{|S|} \prod_{l=1}^{R} P(r_{il} = (\mathcal{M}_r(s_k))_l \mid \mathbf{y}, \theta, r_{i1}, \ldots, r_{i(l-1)})$$

$$= \sum_{\substack{k=1 \\ (\mathcal{M}_b(\mathbf{s}_k))_j = 1}}^{|S|} \prod_{l=1}^{B} P(r_{il} = (\mathcal{M}_b(s_k))_l \mid \mathbf{y}, \theta, r_{i1}, \ldots, r_{i(l-1)})$$

$$= \sum_{\substack{k=1 \\ (\mathcal{M}_b(\mathbf{s}_k))_j = 1}}^{|S|} P(r_{ij} = (\mathcal{M}_b(s_k))_j \mid \mathbf{y}, \theta, r_{i1}, \ldots, r_{i(j-1)}) * \prod_{\substack{l=1 \\ l \neq j}}^{B} P(r_{ij} = (\mathcal{M}_b(s_k))_l \mid \mathbf{y}, \theta, r_{i1}, \ldots, r_{i(l-1)})$$

$$= P(r_{ij} = 1 \mid \mathbf{y}, \theta, r_{i1}, \ldots, r_{i(j-1)}) \textstyle\prod_{\substack{l=1 \\ l \neq i}}^{B} P(r_{il} = 0 \mid \mathbf{y}, \theta, r_{i1}, \ldots, r_{i(j-1)}) + P(r_{il} = 1 \mid \mathbf{y}, \theta, r_{1l}, \ldots, r_{i(j-1)})$$

$$= P(r_{ij} = 1 \mid \mathbf{y}, \theta, r_{i1}, \ldots, r_{i(j-1)}).$$ □

To describe the proof in text, the first two steps are applications of the definitions of Equation (2.4) and Equation (2.6). In step 3, the hypothesis that $\mathcal{M}_b$ is extended by $\mathcal{M}_r$ is applied. Then, in step 4, the case where bit l is the same as bit j is factored. The sum is then moved inward, resulting in the product becoming a product on terms that all evaluate to 1. The term that was originally moved out of the product sum becomes $P(r_{ij} = 1 \mid \mathbf{y}, \theta, r_{i1}, \ldots, r_{i(j-1)})$. This is a result of moving the sum inward. This step also concludes the proof. This result implies that the formulation for 4^n-QAM is a generalization of direct modeling of LLRs with a neural network. The framework should thus be as accurate as models directly optimized on the LLR. However, it has mapping independence. As a side note, non-square constellation can also be added to the representation. This can be done by appending all and not including them in the hierarchical structure. Doing so gives a shared representation for 4^n-QAM and separate representations for the other QAM constellations.

Scaling. An issue exists with the masking hierarchy found by [82]. The concept of this hierarchy is that 16-QAM can be considered as an extension of QPSK, 64-QAM as an extension of both 16-QAM and QPSK, etc. However, looking at Figure 2.1, there is a deviation in this hierarchy visible. For 16-QAM to be a proper extension of QPSK, it is expected that the constellation points of QPSK align with those of 16-QAM. An appropriate alignment based on the given decision boundaries would be as follows. First, take the average of the four

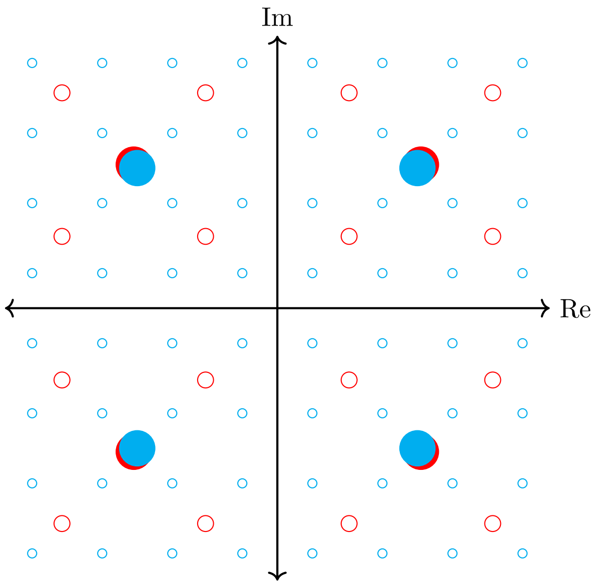


Figure 2.2: Comparison of 16-QAM and 64-QAM. The red circles represent 16-QAM with the large red circle being the average for each quadrant. We presented the same in green for 64-QAM. While the averages for both constellations are quite similar, they are not exactly equal. This causes a small reduction in BER when masking is performed.

constellation points with positive real and imaginary components of 16-QAM. Then, align this average with the QPSK point having a positive real and imaginary component.

However, this is not achieved when normalizing a signal based on the average power. The averaged points in the 16-QAM constellation have the same angle as the QPSK constellation points but a slightly smaller amplitude. This is not an issue when only 16-QAM and QPSK are combined, as the decision boundaries between points still align perfectly. However, this causes problems when including a third constellation from this family. Although the alignment between 64-QAM and 16-QAM is much better than between QPSK and 16-QAM, there is a discrepancy. This has been illustrated in Figure 2.2. This discrepancy becomes an issue in a multi-demapper that demaps, e.g., four different types of 4^n-QAM constellations. This demapper then achieves worse BER rates than a neural receiver that only has to demap one type.

Note that if the signal power of a received signal is normalized, the constellations can be mapped to any desired space through multiplication. Thus, an appropriate hierarchy for 4^n-QAM can be constructed with a scaling factor. This ensures that the underlying demapping problem is on the same scale for all of these constellations. Given the set S^{4^n-QAM} that contains all complex-valued constellation points for a certain constellation, define the set of $S_{+}^{4^n-\text{QAM}} = \left\{s \in S^{4^n-\text{QAM}} \mid Re(s) > 0 \;\&\; Im(s) > 0\right\}$. Then, one can compute the scaling

	QPSK	16-QAM	64-QAM	256-QAM
γ	1	1.118	1.146	1.152

Table 2.1: The multiplication factors used for different QAM constellations.

factor for a constellation as

$$\gamma = \frac{\sum_{c \in S_+^{4^n-\text{QAM}}} c}{\sum_{s \in S_+^{4^n-\text{QAM}}} s}. \tag{2.7}$$

Using this formula, the appropriate scaling factors for the 4^n-QAM constellations are presented in Table 2.1. By multiplying a received normalized signal by these factors, the received signal can be lifted back into the same space. The hierarchical approach with a scaling factor is only applicable to 4^n-QAM constellations.

2.3.3 APSK Representation

The advantage of separating $\mathcal{M}_r$ and $\mathcal{M}_b$ and defining $\mathcal{M}_r$ as non-bijective mapping is not immediately apparent for 4^n-QAM. For square QAM constellations, a masking-based approach can encode all of these with a bijective bit mapping. However, constellation groups generally do not allow solutions where $\mathcal{M}_r$ is bijective. Thus, elements of ∇ are not necessarily independent of each other. Furthermore, it is not always practical to assume that the bit mapping is the same. Different standards may have different bit mappings that can be decoded in a similar way. An important example of this is the family of circular APSK constellations.

These types of constellation are common in the DVB-S2 standard [126]. They come in various configurations and shapes, but can often be defined on the basis of a few aspects. These include the number of amplitude levels, the number of phase levels, the distance between each amplitude level, and the phase offset per amplitude level. For convenience, from now on the different levels of amplitude shift are referred to as *circles*. Note that no assumption has to be made that the outer circle has an amplitude of 1. Similarly to traditional receivers, the only assumption necessary is that the amplitude level of each circle is consistent.

For many of the constellations in the digital video broadcasting-satellite second generation extensions (DVB-S2X) standard, there are a few commonalities. The circles in many APSK constellations have a phase offset of π/M where M is the number of constellation points on that circle. The simplest instances of this family are BPSK and QPSK. To better illustrate this family, QPSK overlaid with offset 8-PSK was drawn in Figure 2.3. A potential Gray mapping for both was included. The figure shows that the QPSK points split the offset 8-PSK points exactly through the middle for each quadrant. Interestingly, the real and imaginary axis lines form decision boundaries for both constellations. For QPSK, they form the

Algorithm 1 Get APSK Representation of Constellation

```
Require: none
Ensure: Returns updated representations for QAM, circle, and family bits
 1: Initialize $q \leftarrow$ zeros(representation_size $\times$ #symbols)
 2: Determine the quadrant in which each point lies
 3: for each point do
 4:     if point in bottom-left quadrant then
 5:         Assign quadrant bits 00 in $q$
 6:     else if point in top-left quadrant then
 7:         Assign quadrant bits 10 in $q$
 8:     else if point in bottom-right quadrant then
 9:         Assign quadrant bits 01 in $q$
10:     else
11:         Assign quadrant bits 11 in $q$
12:     end if
13: end for
14: Number the circles from innermost to outermost
15: for each point do
16:     Compute the binary value with
17: $\lfloor current\_circle * 2^{\lceil \log_2(total\_circles) \rceil} / (num\_circles - 1) \rfloor$
18:     Assign the Gray-coded binary representation of the circle in $q$
19: end for
20: for each quadrant and circle do
21:     Extract all points within the circle and quadrant
22:     Determine their order based on their (counter-)clockwise angle. The top-
    right and bottom-left have a clockwise order
23:     Number them using:
24: $\lfloor current\_point * 2^{\lceil \log_2(total\_points) \rceil} / (total\_points - 1) \rfloor$
25:     Gray-code these values and assign them to the symbols in $q$
26: end for
27: Return $q$
```

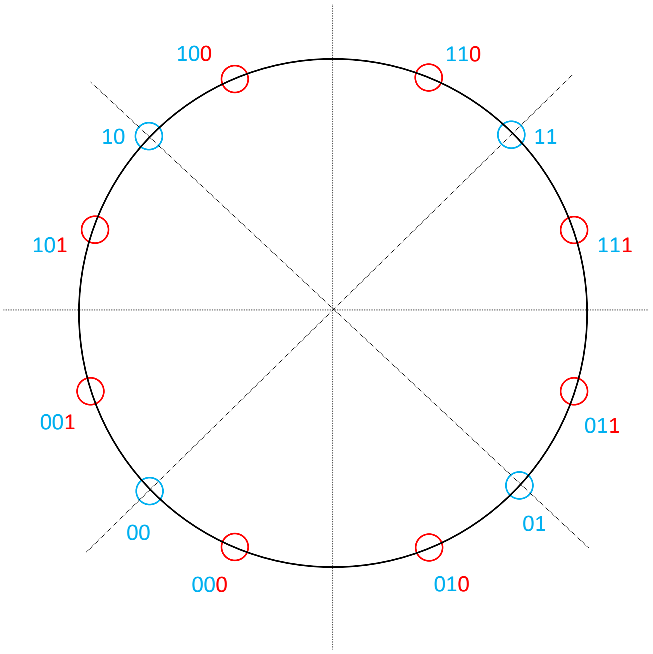


Figure 2.3: A QPSK (blue) and offset 8-PSK (red) constellation overlaid. The latter constellation hierarchically relates to the former, as by adding one more bit (red) to the two bits (blue), the quadrants can be split in two.

set of optimal hard decision boundaries. For offset 8-PSK, the lines through the QPSK points have to be added. The hard-decision boundaries between QPSK and offset 8-PSK thus overlap. This means that only a demapper for offset 8-PSK is needed to demap both QPSK and offset 8-PSK. It is thus clear that these two constellations form a hierarchy.

This hierarchy can be used to build an efficient and more general representation for APSK. The relationship in Figure 2.3 holds when the number of points in the circle is a power of two starting at four. However, many APSK constellations have a number of points from these different from those. Nevertheless, they still obey this hierarchical relationship as long as the phase offset is π/M. Note that this phase offset can thus differ per circle, as each circle can have a different number of points. The points that form a hierarchy follow $4 \times 2^n \times d$ where $n = 0, 1, 2, \ldots$ and $d = 1, 3, 5, \ldots$. This is equivalent to the set of all odd numbers. Intuitively, the hierarchies are formed with an odd number d at the base. Every factor of 2^n then forms a set in which the constellation points are split again in half. For example, when there are 12 points on a circle, the decision boundaries are perfectly contained by a circle with 24 points. As these circles do not allow for a perfect binary representation a non-bijective dependent mapping is necessary.

The process of generating the binary representation per circle is divided into two steps. First, the bits belonging to a quadrant are determined. The bits belonging to *family* are then determined. Here, family refers to the value of d that applies to the number of points in a circle. For example, a circle with 12 points is part of the 3-family. The general procedure is described in Algorithm 1. In each quadrant and for every circle, points are ordered based on their angle and then Gray coded. Doing so ensures that the representation has the smallest error possible. There are two important details to the labeling process. First, for only three points, they are not labeled as $0, 1, 2$ but as $0, 1, 3$. This procedure ensures that the labeling respects the hierarchical relationship that was established. Second, the top right and bottom left quadrants are labeled clockwise. This ensures that neighboring points in different quadrants have the same representation bits. The quadrant bits themselves correspond to the representation of QPSK. This representation is common in all 4^n-QAM constellations and most APSK constellations. These bits can then be combined with the 4^n-QAM representation. This is done by computing then separately for further efficiency. These quadrant bits are computed on the basis of the sign of the real and imaginary components.

One more aspect is the configuration of the amplitudes per circle. APSK constellations often consist of multiple circles, with each circle having a different amplitude. Although the outermost circle in a normalized APSK constellation usually has an amplitude of around 1, the inner circles can vary depending on the configuration. The optimal decision boundaries for the circles differ on the basis of the configuration. The assumption is made that there is a known set of configurations, such as in the DVB-S2x standard. For each configuration, a separate set of circle bits is determined. It is important not to share the circle configuration. The network becomes less capable of demapping when the circle bits are shared.

Finally, some APSK constellations do not follow any of the structures discussed above. A simple example of this is 8-PSK. As 8-PSK has a phase offset of 0, the decision boundaries do not form a hierarchy. A solution to include these constellations is to extend $\mathcal{M}_r$ with $\mathcal{M}_b$ directly. By doing so, no loss of performance for these constellations is guaranteed by Theorem 1. Thus, any arbitrary constellation can be included in the framework. It is also important to note that the framework does not consider phase ambiguity. As with other receivers, some mechanic is needed to address phase ambiguity. Resolving phase ambiguity does not affect the framework, however. If the framework is unable to resolve phase ambiguity, a standard neural receiver is unlikely to be able to resolve it, also.

2.3.4 Extension to AMR

As neural demapping has now been extended to simultaneously demap multiple constellations to increase parameter efficiency, it must be asked whether this task can be further extended. To do so, the link between the simultaneous demapping of multiple types of constellation and multitask learning is drawn. The shared challenges in channel estimation, equalization, and demapping are used to more efficiently demap multiple constellations. In that context, it can be seen that the pipeline could be extended to other tasks. It is observed that many tasks

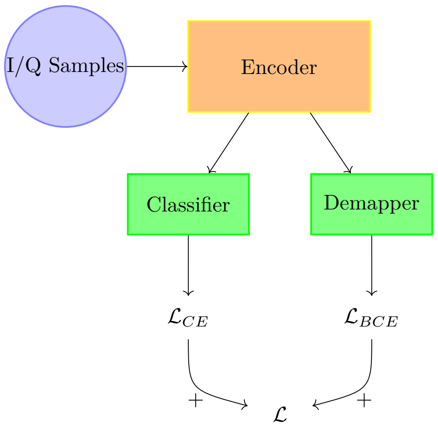


Figure 2.4: The structure of a multi-branch AMR and demapper model.

in communication are rendered trivial in the absence of complex impairments, implying that the most important task that a neural network must perform is the identification and correction of channel impairments.

For that purpose, AMR was added to the pipeline. AMR is an interesting fit for inclusion, as the framework's approach already allows demapping multiple types of constellations simultaneously. Thus, it is fairly straightforward to include AMR as well. This can be done using a shared representation learned for both tasks. Combining AMR with simultaneous demapping would be a complementary change. Making the change removes the need for the assumption that the modulation type has been communicated. One issue is that the scaling factor previously discussed cannot be included. This scaling factor violates a fundamental assumption of AMR, as the modulation type must be known. This limits the demapping performance of the model. Note that it is not required to perform the QAM masking approach used. One could treat 4^n-QAM in the same way as the other QAM constellations. This increases memory use and complexity, but removes the need for the scaling factor.

A schematic of the combined model has been included in Figure 2.4. The model consists of three components: the encoder, the classifier, and the demapper. The exact architectures of these models can be decided upon on the basis of the setting. In principle, the encoder contains most of the model parameters and has the primary task of correcting channel impairments. The demapper is a small model that first translates the corrected samples to our representation space and then translates them to bit LLRs. Finally, the classifier aggregates all symbols in the transmission and then provides a probability over the set of possible modulation types. For jointly training these models, the binary cross-entropy (BCE) loss over the bit LLRs is combined with the cross-entropy (CE) score of the modulation type labels. Note that all models discussed so far can be considered within this framework by leaving out either the classifier or the demapper.

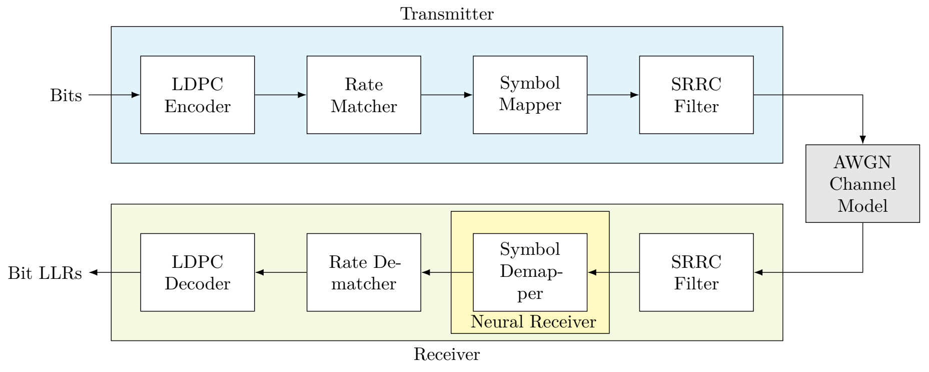


Figure 2.5: The AWGN link-level simulation that was used for part of the experiments.

2.4 Experimental Setup

To validate the framework, experiments were run using simulations. The simulations were performed using Sionna [83]. The experiments were first ran in an AWGN setting to validate the the approach. Second, simulations were run over an OFDM fading channel to validate whether the conclusions transfer to a more realistic setting.

2.4.1 Additive White Gaussian Noise

To evaluate the performance of the method, the data was simulated over a simple AWGN channel. This was done to better understand how the method behaves in various combinations of constellations. As an AWGN channel allows for a simple optimal hard-decision demodulation rule, it was used as a bound to benchmark the method. The simulation pipeline is depicted in Figure 2.5.

To perform our experiments, a simple DNN receiver was used as depicted in Table 2.2. The encoder takes the raw I/Q samples and consists of three hidden layers of size 256 each, followed by a ReLU activation function and a batch normalization layer. This model was chosen because it has sufficient capacity to approach the BER lower bound for the neural receiver. The DemapHead module then receives the output from the encoder and returns either the bit LLRs directly for the standard neural receiver or the representation bits for the multi-neural receiver. The ClassifierHead is only included when the model performs AMR. It first performs global average pooling over the time dimension and then passes the output through a one-layer deep neural network.

The rest of the hyperparameters are detailed in Table 2.3. These settings are included for reproducibility; the code will be released on GitHub. For the AWGN channel, the noise was sampled uniformly at random between -5 and 20 E_b/N_0. As we perform simulations, every sequence is new. The number of symbols per block refers to the number of I/Q samples encoded at each step. Coding is done with a

Table 2.2: AWGN Experimental Configuration

Layer Type	Configuration
Encoder	
Dense	Filters: 256, Activation: ReLU
BatchNormalization	
Dense	Filters: 256, Activation: ReLU
BatchNormalization	
Dense	Filters: 256, Activation: ReLU
BatchNormalization	
ClassifierHead	
GlobalAveragePooling1D	
Dense	Units: 256, Activation: ReLU
Dense	Units: Number of Classes
DemapHead	
Dense	Filters: 256, Activation: ReLU
BatchNormalization	
Dense	Units: Output Size

5G compliant LDPC de / encoder with a code rate of 1/2 [8]. For training, the coding module is not included as it does not affect the optimal decision boundary. The batch size depends on the number of constellation types included. For the baseline neural receiver, a batch size of 32 is used. For the multi-receiver, 32 is multiplied by the total number of constellations incorporated during training. In addition to the optimization settings mentioned in the table, we use a scheduler that reduces the learning rate by a factor of 10 in epochs 100 and 125. As new data is simulated every batch, there is no relevant dataset size. The only impairment used in the AWGN channel is the Gaussian noise element.

Each sequence has a different signal-to-noise ratio (SNR), and thus the DNN demapper should learn how to correct for the noise level. The APSK constellations are generated based on the DVB-S2x specification [126]. These constellations are specified by their configuration and code rate. For example, the constellation 64-APSK-7/9 is the APSK constellation with 64 constellation points associated with the code rate 7/9 under this specification. Different code rates generally refer to a different circle radius in the APSK constellations, but may also refer to entirely different layouts. A diverse subset of the 118 constellations is taken for the experiments. For example, there are many variants of 16-APSK with a configuration of $4 + 12$. This means that the inner circle contains 4 points and the outer circle 12 points. Including all of them is not necessarily as interesting as including an entirely different configuration.

To evaluate the method, the following three systems are compared:

1. A baseline approach that computes the bit LLRs based on the squared Euclidean distance of received symbols to every constellation point. As this

Table 2.3: AWGN System Parameters

Parameter	Value
Seeds	42 to 44
Batch size	32
Number of Epochs	150
Batches per Epoch	500
Optimizer	AdamW
Learning Rate	0.004
Weight Decay	0.01
#Symbols per block	512
Filter Type	Square Root Raised Cosine
Span in Symbols	10
Samples per Symbol	4
Roll Off Factor	0.25
SNR Range	-5 dB to 20 dB

approach is equal to the MAP-estimator assuming a Gaussian distribution per constellation point, this approach is optimal under AWGN. That is why we included this approach to demonstrate the achievable lower bound within our setup.

2. A neural demapper that directly predicts the bit LLRs. This single receiver allows us to evaluate how a receiver that only has to demap a single constellation performs comparatively.

3. The multi-demapper approach that we trained on multiple constellations. The constellations on which it is trained are QPSK, 16-QAM, 64-QAM, 256-QAM, 8-PSK, 16-APSK-2/3, 16-APSK-100/180, 32-APSK-2/3, 64-APSK-7/9, 64-APSK-128/180, 256-APSK-124/180.

2.4.2 Orthogonal Frequency-Division Multiplexing

Note that a neural receiver is redundant for an AWGN channel, as its primary objective is correcting channel impairments. A simple SIMO OFDM simulation pipeline is used to further evaluate the method. A schematic for the pipelines is given in Figure 2.6. Four different pipelines are compared for the OFDM setting:

1. For the first baseline, perfect channel state information is assumed. This means that ground-truth fading effects are used for equalization. This baseline served as a reference to what is achievable.

2. For the second baseline, LS estimation based on the pilot symbols is performed. These estimates are passed to the equalization module.

3. The single neural receiver is the DeepRX [82] model. It takes the received I/Q samples, the ground-truth pilot symbols, and the LS estimation based

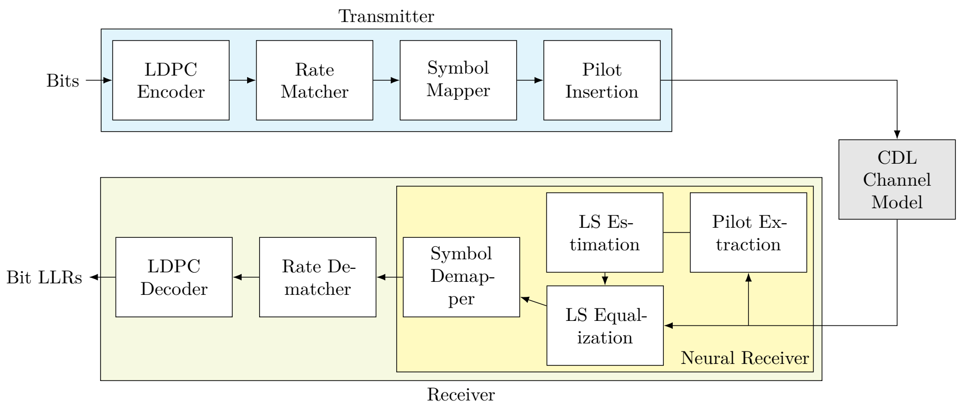


Figure 2.6: The SIMO OFDM pipeline that is used for part of the experiments.

Table 2.4: OFDM System Parameters

Parameter	Value
Number of Epochs	150
Batches per Epoch	500
Subcarrier Spacing (Hz)	30,000
Number of subcarriers	128
Number of OFDM Symbols	14
Number of RX Antenna	2
Number of TX Antenna	1
Pilot Pattern	Kronecker
Pilot Symbol Indices	2, 11
Training CDL Model	C
Evaluation CDL Model	D

on these pilot symbols as input.

4. The multi-neural receiver. It is built upon DeepRX and extends it with the representation module.

The OFDM system parameters for our method can be found in Table 2.4. Similar neural network settings were used as they worked well for OFDM too. Thus, each receiver is also trained for 150 epochs using the AdamW optimizer with a learning rate of 0.004. At epochs 100 and 125, the learning rate is reduced by a factor of 10. The receiver has a dual antenna setup with antenna patterns as in the 3GPP TR 38.901 specification. We have opted for a SIMO pipeline as it is sufficiently simple to allow isolating the important factors while still being interesting to evaluate a neural multi-receiver. The combination of a channel delay line (CDL)-C model for training and a CDL-D model for evaluation was chosen. These channel models are sufficiently general to draw conclusions for SIMO systems. A different channel model for training and evaluation is chosen to ensure that the approaches generalize.

In each training batch, a set of fading channels is sampled from the given CDL-C configuration. The channel coefficients are simulated using this configuration. These coefficients are then multiplied element-wise with the data to be transmitted. Finally, Gaussian noise is generated based on a batch of uniform random SNR values between -5 and 20 E_b/N_0. The noise samples are then added to the batch to simulate the OFDM samples received. As simulations are used, there is again no concept of a dataset size or training/test split. Every generated block is different from every other block. Frame synchronization is assumed to have been accomplished beforehand. This is fair, as this assumption is made for all methods compared. The experiments were run on a compute node that has a partition with 18 cores of an Intel Xeon Platinum 8360Y CPU, 128GB of RAM, and an NVIDIA A100 GPU with 40GB of VRAM.

Simulations are often sufficient for generalizable conclusions. Real-world demapping datasets are often limited in size and diversity. An alternative is to use the DeepMIMO dataset [16]. This is a highly realistic ray-traced set of scenarios that allows sampling channel coefficients.

2.5 Simulation Results

2.5.1 AWGN Pipeline

First, the methods were compared for AWGN. In Figure 2.7, the BER across the entire evaluated SNR range has been depicted for 6 different constellations. These constellations contain the four constellations that share the QAM bits, i.e., QPSK, 16-QAM, 64-QAM, and 256-QAM. It can be observed that for all of these constellations, both the neural and multi-receiver achieve a performance similar to the baseline. For QPSK, this implies that no performance is lost by sharing the representation with other constellation types. The QPSK constellation is one of the most interesting in this evaluation, as it shares its representation bits with every

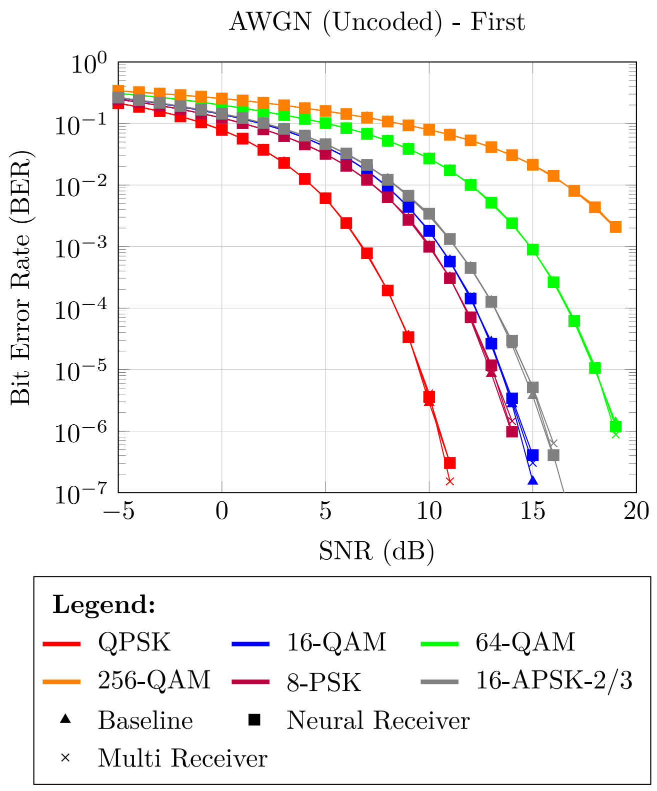


Figure 2.7: BER performance of different methods under AWGN as a function of SNR.

constellation other than 8-PSK. However, this has not hampered the optimization of this constellation type. The 256-QAM constellation again shares many of its representations with other methods, but it also has many unique representation bits. Furthermore, 256-QAM is one of the most complex constellation types in the setup. It is also relatively hard to fit, since the neural receiver needs to learn to distinguish many more symbols. Again, the approach achieves a performance similar to that of both the baseline and the neural receiver. Similarly, the performance of both the neural receiver and multi-receiver approaches the baseline for both 16-QAM and 64-QAM. One of the smaller APSK settings is also depicted in this figure. This constellation was depicted because it does not share many representation bits with other approaches. It is also one of the constellation types for which there is no bijective mapping for the constellation points on the outer circle. However, there does not seem to be a major performance difference between the standard neural receiver and multi-receiver. Finally, the results for 8-PSK are presented. Here, it is observed that the performance of the multi-receiver is approximately equal to the baseline and neural receiver. It is indicated by this that, without bit sharing, a similar performance can be achieved.

Second, the remaining five included in Figure 2.8 were depicted. These five

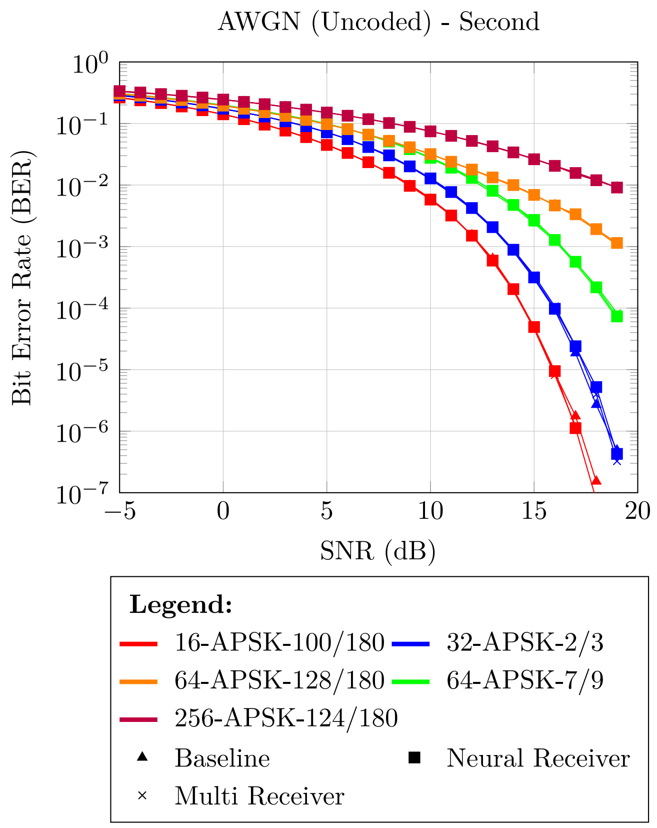


Figure 2.8: BER performance of different methods under AWGN as a function of SNR.

include three APSK constellations that share all family bits. These include 16-APSK-100/180, 64-APSK-128/180, and 256-APSK-124/180. These have $8 + 8$, $16 + 16 + 16 + 16$, and $32 + 32 + 32 + 32 + 32 + 32 + 32 + 32$ as configurations, respectively. It can be observed that these constellations all achieve similar performance to their respective baselines when modeled separately. Consequently, the proposed bit-sharing representation seems valid under AWGN. It can also be seen that the other two constellations achieve a performance similar to their respective baseline.

In general, it is thus observed that the multi-receiver is as accurate under AWGN as the neural receiver that only has to demap a single constellation. This conclusion seems to hold for all included constellations. This indicates that, under AWGN, there is no reason to constrain the set of possible constellations to just one. Using the presented approach, multiple different types of constellation can be included without loss of performance.

Learned Representation To better understand what the method learns, a range of I/Q samples was generated and passed through the multi-receiver. The

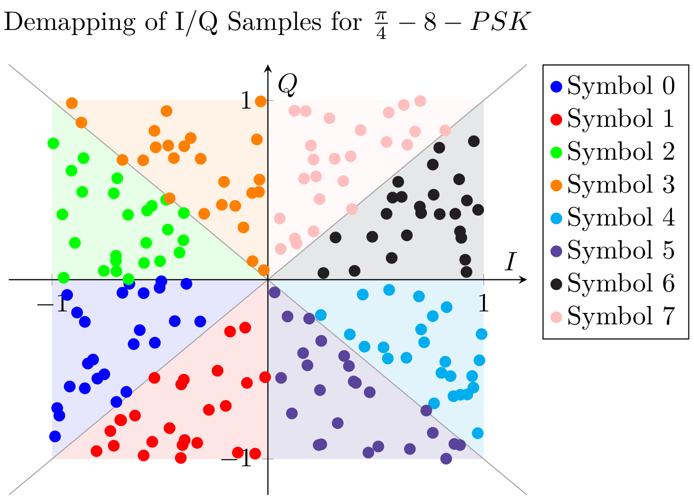


Figure 2.9: Distribution of I/Q samples based on the learned representation. The position is given, and the point color is the model prediction. The background is a visual aid to depict the ground truth. The bits associated with QPSK and the first 1− family APSK bit are depicted here.

range of I/Q samples consists of a set of evenly spaced points between the corner points $\{(-1,-1),(1,-1),(-1,1),(1,1)\}$. For each of these samples, the symbol label was determined based on the hard-decision bit labels. For example, if a bit has a hard-demapping of 11, the colour associated with the symbol that demaps to 11 was given. The first representation bit of the 1− family of the APSK representation was also added. According to the intuition of the approach, this representation bit should add the diagonal decision boundaries defined through the points $(-1,-1)$ to $(1,1)$ and $(-1,1)$ to $(1,-1)$. The result of this is depicted in Figure 2.9. Although some points around the edges may be classified incorrectly, the points are overall correctly classified based on their position. These results give an idea of what the approach learns. The expected, optimal decision boundaries seem to be the same as the ones learned by the neural multi-receiver. This result underlines the validity of the approach.

Demapping + Classification Finally, the classification head was included in the approach. The multi-receiver was trained with both the demapping head and the classification head simultaneously. To compare, a model with just a classification head was also trained. This experiment allows evaluating whether the tasks of classification and demapping can be performed jointly. The results of the AMR approach are shown in Figure 2.10. It can be seen that the accuracy of both is similar. This indicates that the classification task is not hindered by the demapping task performed. To further validate this finding, the BER of the combined model is compared with the original 256-QAM multi-demapper results in Figure 2.11. Here, the performance is again equivalent. This indicates that the tasks of AMR and bit demapping do not hinder each other. These results

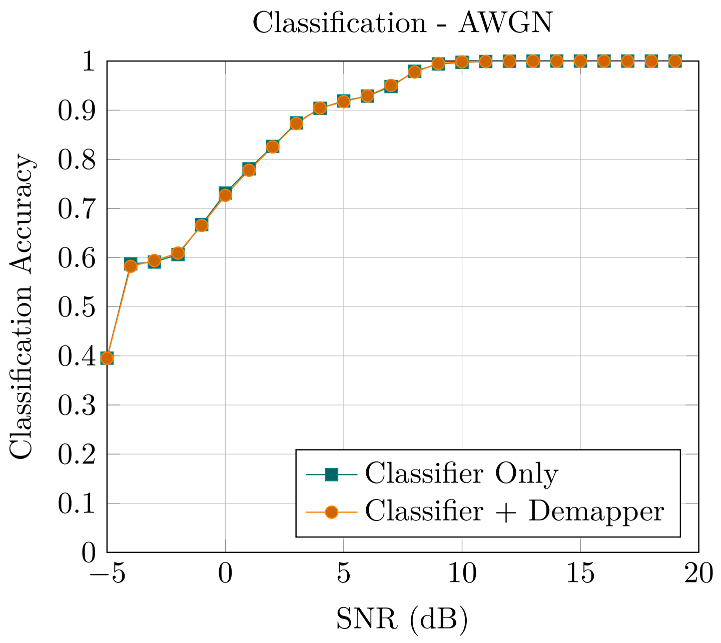


Figure 2.10: AMR performance of different methods under AWGN as a function of SNR.

demonstrate that it is possible to jointly learn a neural classifier and demapper with a shared representation under AWGN without loss of performance.

2.5.2 OFDM Pipeline

To get a better idea of the practical utility of this work, the framework was also evaluated in a set of four SIMO OFDM pipelines. These pipelines were previously described in the experimental setup.

BER Performance First, the BER results of the pipelines for various modulation types were evaluated. In Figure 2.12, the results for the first half of the constellations included in the experimental setup are depicted again. For all constellations, it can be observed that none is as good as the baseline under perfect channel state information (CSI). This is to be expected, as the perfect CSI baseline under simulation is optimal. The neural receiver and the multi-receiver also beat the LS-estimation significantly, which underlines that an NN-based approach is complementary for both pipelines.

The first four important constellations to consider are the group of QPSK, 16-QAM, 64-QAM, and 256-QAM. Together, these form an interesting family, as they all directly extend each other. Both the neural receiver and multi-receiver achieve relatively similar performance for all four of these constellations. Slight

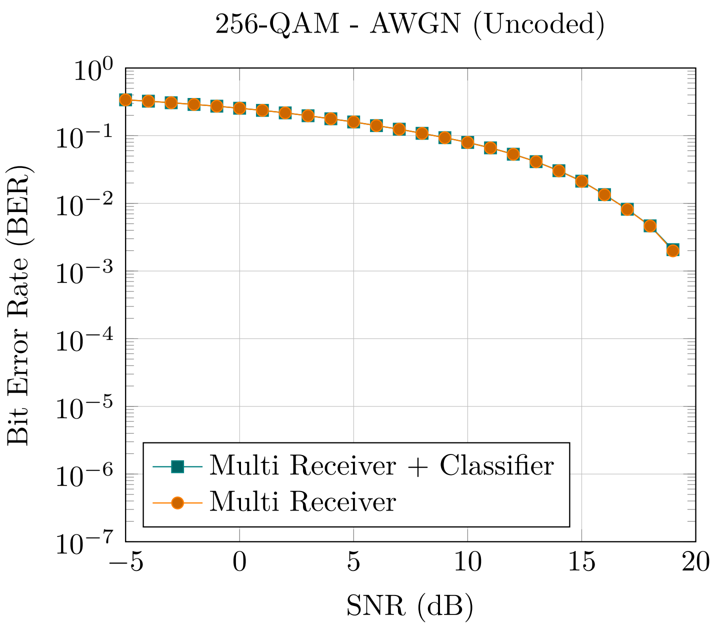


Figure 2.11: BER performance of different methods under AWGN for 256-QAM as a function of SNR.

discrepancies between the neural receiver and multi-receiver can be observed. Generally, the neural receiver achieves a lower BER when there is a difference. At an SNR of 4, the neural receiver for 256-QAM achieves a clearly lower BER than what we observe for the multi-receiver. Similarly, it can be observed that at an SNR of -3, the neural receiver is also slightly better than the multi-receiver for QPSK. Overall, the BER curve is fairly similar for these constellations. Note that the constellations QPSK, 16-QAM, 64-QAM, and 256-QAM share their representation bits but retain a similar performance as separately trained models. Even under a complicated channel model, masking is able to approach the performance of the single neural receiver for the QAM constellations. Another interesting observation is that the performance of 8-PSK is highly similar as well for both the neural receiver and multi-receiver. As the constellation does not share bits with any other constellations, it demonstrates that adding more general constellations does not harm the BER of these constellations.

In Figure 2.13, the BER curves for the remaining constellations are shown. The most important set of constellations to consider here is 16-APSK-100/180, 64-APSK-128/180 and 256-APSK-124/180 as they all share the 1 family for every circle. The performance for these constellations is again similar for both the neural receiver and multi-receiver. This implies that the APSK representation sharing feature also works under a realistic channel model. Furthermore, 32-APSK-2/3 and 64-APSK-7/9 share representation bits for some circles and do not share them for others. However, their performance is also similar for both the neural receiver

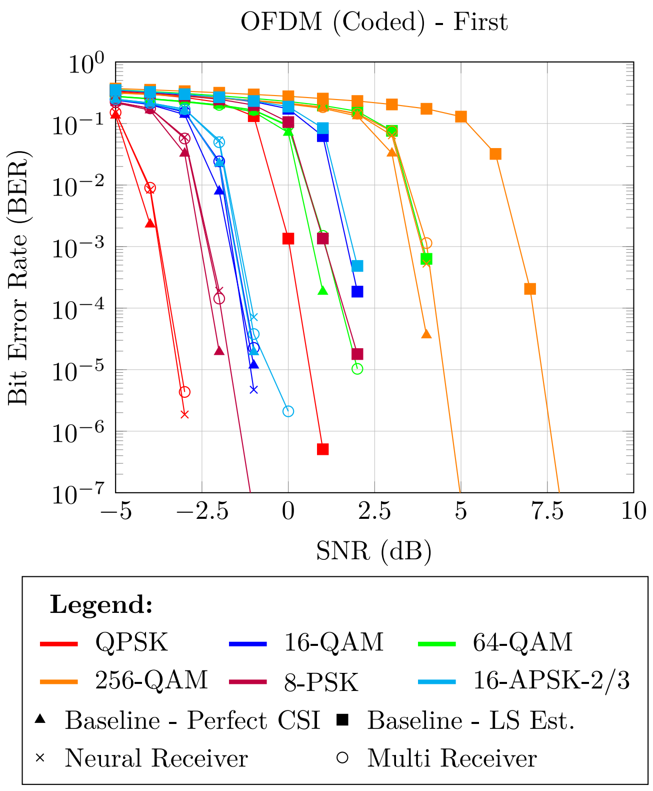


Figure 2.12: Coded BER performance of different methods under OFDM as a function of SNR.

and multi-receiver. Overall, while there are slight performance discrepancies at some SNR points, the model is overall similarly accurate between both the multi-receiver and neural receiver. This implies that the approach is also valid under OFDM with a realistic channel model.

Throughput To get an idea of what the overhead of the approach could be, the throughput values are presented in Table 2.5. These values entail the time it took between receiving the symbols and computing the bit LLRs. This process was repeated 5 times with 1000 MC iterations each. The minimum time of these repeats were taken. The minimum time is commonly used as it is the least noisy estimate of the average run-time of code. The bitrate was then calculated by dividing the number of demapped bits by the time it took to process them. The experiments were run on a compute node that has a partition with 18 cores of an Intel Xeon Platinum 8360Y CPU, 128GB of RAM, and an NVIDIA A100 GPU with 40GB of VRAM.

In the table, it can be seen that the throughput difference is minimal for methods with a smaller number of symbols. However, for 256 symbols, the performance

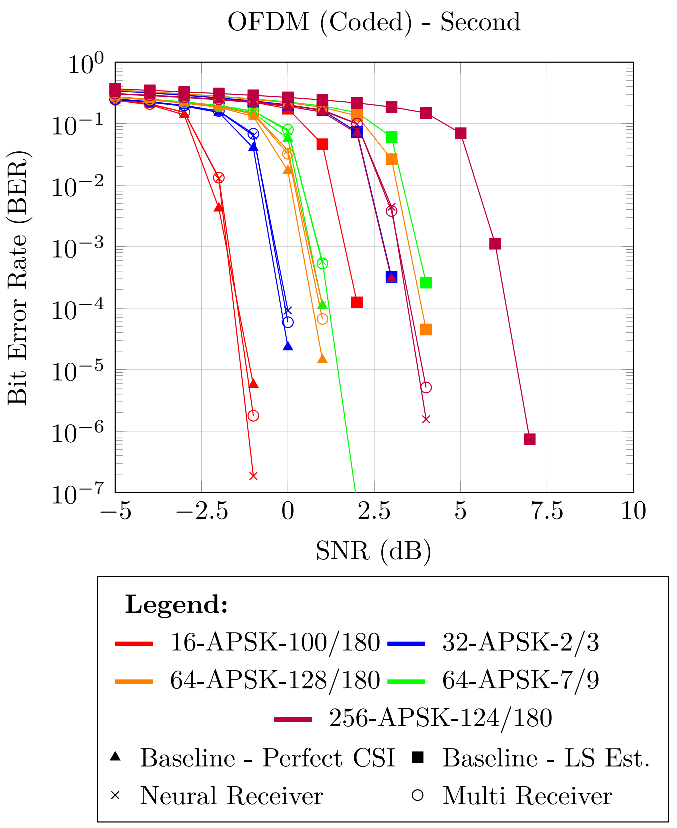


Figure 2.13: Coded BER performance of different methods under OFDM as a function of SNR.

difference becomes more noticeable. The evaluation of symbol probabilities for 256 symbols can be resource-intensive. Note that the assumption was made here that all neural models are already loaded into memory. While this may be realistic for the multi-model, it may prove less realistic for the standard neural receiver models. Thus, it is shown that the performance impact of using the approach under pessimistic assumptions is only slight.

Demapping + AMR Finally, the addition of a classification head to DeepRX is evaluated. To form a classification head, the last residual block of DeepRX was taken and duplicated for the classification head. A global average pooling layer and a dense layer were added at the end to arrive at the classification model. Again, training the classifier only was compared with the combination of the classifier and demapper.

The results were drawn in Figure 2.14. Surprisingly, the combination of a classifier and demapper is not just as accurate but significantly more accurate than training the classifier. This would imply that the task of demapping helps in learning the

Table 2.5: Throughput Measurements for Different Systems and Modulation Methods in Mb/s on an A100 GPU.

Method	Single	Multi
QPSK	4.37	4.25
8-PSK	6.47	6.27
16-QAM	8.46	8.13
16-APSK-2/3	8.41	8.17
16-APSK-100/180	8.44	8.13
32-APSK-2/3	10.05	9.66
64-QAM	11.65	10.98
64-APSK-7/9	11.66	10.99
64-APSK-128/180	11.63	10.97
256-QAM	14.71	12.38
256-APSK-124/180	14.71	12.35

task of classification. This result is consistent with the work by [74]. However, they have multiple loss functions that operate directly on the underlying data. The approach presented here can perform this process fully end-to-end. Intuitively, these results make sense, as the learning task becomes more structured when demapping is included. The AMR task needs to deal with at least two types of variance: the variance caused by the channel and the variance caused by the symbol transitions. When demapping is included in the objective, the latter becomes given.

To validate the demapper of the combined model, it is compared with the previously evaluated demapper in Figure 2.15. Here, it can be seen that the combined model is less effective than the model that just performs demapping. It seems that the cost of augmenting the classification performance under OFDM is that the demapping performance is worse. In conclusion, the classification head under the shared model performs better than the model purely trained with a classification objective. This indicates that the joint demapper can provide additional information for AMR approaches. This allows them to more quickly converge to a more effective AMR approach without increasing the complexity of the downstream model. This work thus also presents a contribution to AMR.

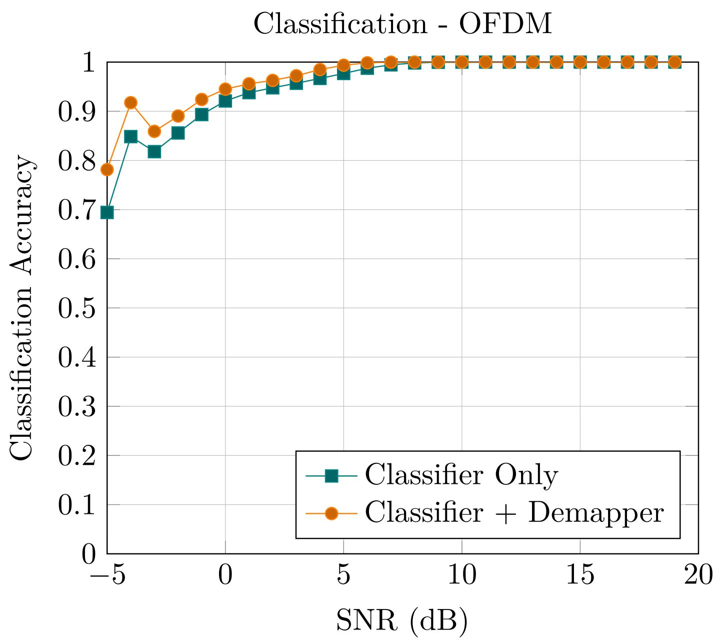


Figure 2.14: AMR performance of different methods under OFDM as a function of SNR.

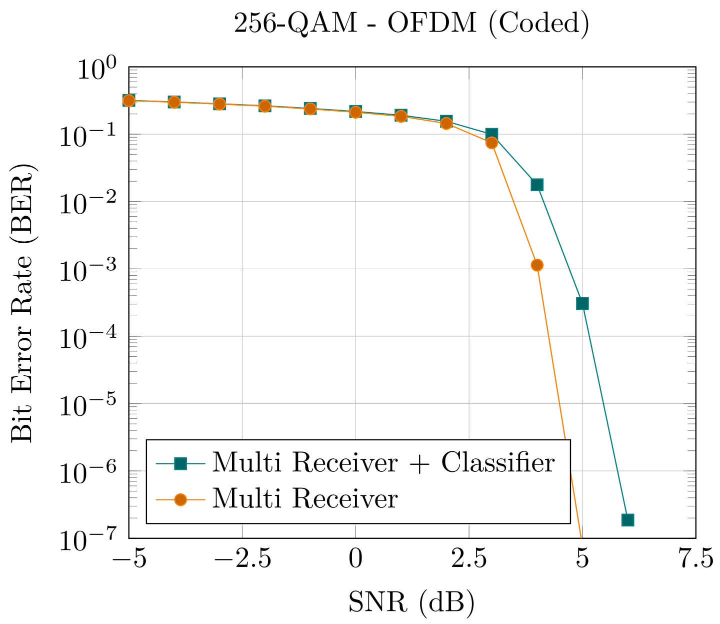


Figure 2.15: BER performance of different methods under OFDM for 256-QAM as a function of SNR.

2.6 Discussion and Conclusion

We proposed a framework that allows us to jointly train on and generalize to multiple types of constellations. In this framework, we include mapping independence and the ability to use hierarchical relationships in families of constellations. We found that it is not only possible to train a model jointly on multiple constellations, but that such a method approaches the hard-decision BER bound under AWGN. It also performs similarly to neural receivers under both AWGN and SIMO OFDM channels. Furthermore, we found that combining AMR and demapping results in better AMR accuracies under OFDM.

Thus, the framework introduced opens up the possibility of efficiently modeling families of constellations in the context of DNNs. This is an important step in making DNN receivers sufficiently adaptable in dynamic environments. It also shows the promise of combining various tasks in the communication pipeline. The framework is modular and applicable to any deep learning receiver pipeline. As the approach is implemented as a module applied to every symbol individually, it is also applicable to multi-user MIMO scenarios without further extension. Thus, this work addresses an important issue in the flexibility of neural receivers. The framework applies to many related works, as the approach can be modularized and attached to any neural receiver that outputs bit LLRs. Due to its modular nature, the work can be easily extended to larger datasets and more complex architectures. Future work includes extending the work to MIMO, beamforming, and real-world datasets. Future work could also look at approaches to reducing the exact computation. This is necessary for scaling to very large constellations, such as, e.g., 4096-QAM.

CHAPTER 3

Relative Phase Equivariant Deep Neural Systems for Physical Layer Communications

Based on [4]:
Arwin Gansekoele, Sandjai Bhulai, Mark Hoogendoorn, and Rob van der Mei
Relative Phase Equivariant Deep Neural Systems for Physical Layer Communications
Published in Transactions on Machine Learning Research (TMLR).

Abstract

In the era of telecommunications, the increasing demand for complex and specialized communication systems has led to a focus on improving physical layer communications. AI has emerged as a promising solution avenue for doing so. Deep neural receivers have already shown significant promise in improving the performance of communications systems. However, a major challenge lies in developing deep neural receivers that match the energy efficiency and speed of traditional receivers. This work investigates the incorporation of inductive biases in the physical layer using group-equivariant deep learning to improve the parameter efficiency of deep neural receivers. We do so by constructing a deep neural receiver that is equivariant with respect to the phase of arrival. We show that the inclusion of relative phase equivariance significantly reduces the error rate of deep neural receivers at similar model sizes. Thus, we show the potential of group-equivariant deep learning in the domain of physical layer communications.

3.1 Introduction

In the modern era, our world has become increasingly interconnected, with large amounts of information flowing seamlessly across continents. Telecommunications has been a key enabler in this transition to the era of information. However, as connectivity requirements become more extreme, the need for more specialized software-based networking systems increases. An important component in these developments is the physical layer of the networking stack. The physical layer manages the electrical and mechanical components of the transmission process, where the information is generally treated as some arbitrary sequence of bits to ensure compatibility with the rest of the stack. The modularity of the networking approach has allowed the rapid adaptation and development of telecommunications systems. The concept of an SDR could be a solution to many of the demands for increasing flexibility and speed. An SDR performs networking on a software level, as opposed to a hardware level, giving a high degree of flexibility at the cost of an increase in compute.

Given the recent advancements in AI as well, it becomes interesting to look into its potential for physical layer communications [84]. The potential of AI for 6G is often considered to lie in its data-driven nature. When a signal with data is transmitted, it often arrives at a receiver distorted, requiring multiple modules to correct the errors and retrieve the original data. One type of distortion is called *fading*, which requires known pilots on the receiver side to perform data-driven channel correction. Korpi et al. [100] proposed to replace a receiver entirely with a single DNN. They named this receiver DeepRx and demonstrated that replacing the entire receiver with a DNN resulted in significant performance gains in correcting fading channels.

Although this work showed the potential of DNNs in an unconstrained environment, a major problem remains that receivers have to be extremely fast and energy efficient. Although over-the-air DNN receivers are known to be feasible, they lack the throughput and energy efficiency to be viable on a large scale. As such, an important challenge is to make DNN models that are as efficient as possible without major reductions in performance. An approach to improve the efficiency of neural networks is to embed inductive biases into the network. Deep neural receivers commonly operate in complex baseband, which is a complex representation commonly used in telecommunications. When considering an OFDM system, they can be further expanded into time-frequency signals. DeepRx has already implemented a form of inductive bias through its convolutional architecture. It performs convolutions across the time and frequency components, resulting in translational equivariance for both components.

To identify further inductive biases, we focus on the phase of the signal. Most digital modulation schemes use some form of amplitude and phase shifting to encode information. Information is encoded by adjusting the phase and amplitude at set sampling points. When receiving a transmission, the initial phase can vary depending on, e.g., the distance between sender and transmitter and the angles of the transmit and receive antennas. Unlike the relative phase shifts between sampling points, the initial phase offset of the signal does not affect the underlying data. However, synchronizing the initial phase offset of a signal is important to get the correct data sequence back. In principle, the initial phase of a signal is relative as it depends entirely on the moments chosen to measure the signal.

Without incorporating this knowledge, a deep receiver is unaware of the phenomenon of a relative phase before seeing any data. Deep receivers generally learn to correct for this phenomenon by observing many instances of fading channels. If the training data is sufficiently diverse, the DNN can allocate parameter sets that identify the absolute phase offset in the underlying signal. An interesting question is whether a model that is aware of this relative initial phase is more capable with fewer parameters than a model that is not. Here, we define a model that is aware of the relative phase as a model that behaves predictably under changes in the initial phase.

We propose to tackle this problem using the concept of group equivariance. Group theory provides a convenient mathematical framework to model symmetries, for example. In work by Cohen and Welling [42], a general approach was proposed to encode discrete group equivariance into neural networks. In line with this field of research, we propose a framework for the construction of deep neural receivers for physical-layer communications that behave predictably under differences in the initial phase of the signal. In doing so, we found the need for element-wise equivariant solutions on regular grids of complex values, and propose a first approach for incorporating these in a DNN. The contribution of this paper is threefold:

1. We identify and construct the components necessary to achieve relative initial phase equivariance for deep neural receivers.
2. We propose a modernized version of DeepRx ([100]) that incorporates these group equivariant components, alongside other recent improvements to the

neural architecture.

3. We validate our approach empirically and demonstrate that including relative phase equivariance achieves a better trade-off between the number of parameters and the performance of the model.

3.2 Related Works

Deep neural receivers. The first important domain to which we contribute is the field of deep neural receiver design. Xuanxuan Gao et al. [185] and Yue Hao et al. [191] were some of the preliminary works on performing the tasks of channel estimation, equalization, and symbol demodulation with a single deep neural network. These works were among the first to highlight the potential of deep learning in the context of receiver design. Neev Samuel et al. [129] extended this work to the MIMO setting and similarly achieved strong results. Honkala et al. [82] proposed DeepRx, one of the deep receivers based on modern CNN design choices along with considerations specifically for physical layer communications. They showed that with a strong architecture, system performance can approach the theoretical limit in certain settings. Korpi et al. [100] extended DeepRx to a MIMO setting. Cammerer et al. [33] proposed using a GNN to better incorporate the interaction between users in a MIMO setting for a receiver such as DeepRx. The MIMO users can vary, highlighting the need for a flexible solution. Raviv and Shlezinger [137] proposed a set of data augmentation strategies to enhance the training of deep receivers. Huttunen et al. [87] proposed DeepTx, which learns the sender part as opposed to the receiver part. Raviv et al. [138] investigated the use of meta-learning for rapid adaptation of deep receivers to new channels. Pihlajasalo et al. [136] looked into making deep OFDM receivers more efficient in terms of power use. Gansekoele et al. [60] proposed a neural receiver that can demodulate multiple constellation types simultaneously without having to adjust the model parameters. To contribute to this literature, we are among the first to explicitly demonstrate that incorporating inductive biases into deep receivers can result in models that require significantly fewer parameters without loss of performance.

Group equivariance and point clouds. The field of geometric deep learning or equivariant deep learning is well-established [31]. One of the seminal works by Cohen and Welling [42] introduced the concept of a G-equivariant network. They propose a method to construct networks that are equivariant with respect to arbitrary discrete groups. This paper provides the basis for our equivariant construction. We found our work to be most related to the field of equivariant point-cloud learning. Relative phase equivariance results in an element-wise equivariance over a grid of complex values or 2D points. One of the first DNNs that operated directly on point clouds is PointNet [37]. Although they included permutation invariance to point order, they did not investigate point rotation equivariance. Thomas et al. [161] proposed TensorField networks for general SO(3) point-cloud equivariance. They use filters built from spherical harmonics to enforce rotation equivariance. Esteves et al. [53] proposed a network for discrete views of 3D point cloud data. This approach differs substantially from TensorField networks in that they lift to the group as opposed to restraining the filters. At a similar time, Li et al. [107] proposed a simple architecture that is equivariant with respect to discrete rotation groups in 2D. Building on these works, Chen et al. [39] proposed a form of depthwise separable convolutions over groups for a more efficient pointcloud network. On the more fundamental side, Dym and Maron [51] demonstrated the universality of G-equivariant point cloud networks. Finally, Bokman et al. [30] proposed a network for 2D point cloud recognition that can incorporate 2D-2D correspondences. However, we still found that the needs of our work differ substantially because the sequence of 'points' is meaningful in the case of I/Q samples. I/Q samples arrive at specific frequencies and times. We found that the combination of operating on a grid with a requirement for element-wise equivariance poses unique challenges and solutions. That is why we contribute to the field of G-equivariant deep learning by addressing (some of) the unique challenges of this new application domain.

3.3 Methods

To construct our model, we first discuss the preliminaries needed to capture relative phase equivariance appropriately.

3.3.1 Preliminaries

When thinking of absolute phase offsets, they most commonly result from *fading*, i.e., a signal changing in strength over time due to geographical positions, environmental effects, etc. Fading can also occur when multiple instances of the signal arrive at the receiver out of phase. Fading as an effect is often modeled as a linear time-invariant (LTI) system $y = h * x + \mathbb{C}\mathcal{N}(0, \sigma^2)$. Here, x are the transmitted data carriers, h the fading coefficients that are multiplied element-wise ($*$), $\mathbb{C}\mathcal{N}$ a complex Gaussian distribution, σ^2 the noise variance, and y a sequence of received I/Q samples. Modeling fading in such a manner gives us a clear way to evaluate the effect on the received signal. These fading effects are often modeled and generated using a channel model to perform system-level simulations. Some common channel models are Rician and Rayleigh fading. These fading models abstract away many of the underlying complexities of signal propagation by modeling fading as a random process. A Rician fading process can be modeled as

$$h = me^{i\theta} + s. \tag{3.1}$$

Here, m is the magnitude of the direct path, and s is the term representing the Rayleigh fading paths sampled as $s \sim \mathbb{C}\mathcal{N}(0, \sigma^2)$. The term θ represents the initial phase of the line-of-sight (LOS) path in the Rician fading channel. The LOS path is the path with the most power during transmission. Often, the value θ is left at 0 to exclude it from the simulations. However, this introduces a disconnect between practical systems and simulation. The work of [134] investigated the difference in performance between a system that knows the value of θ and one that is unaware of it. They found a difference between 2% and 50% of spectral efficiency depending on the benchmark system. Their results indicate that estimating the initial phase of the LOS component is important to correct for the underlying fading effects.

Frequently, channel models are used to simulate fading coefficients h to train deep neural receivers. Many channel models exist, some realistic enough to validate practical systems directly [8]. Assume that we sample channel coefficients from a Rician fading model and sample a different initial phase $\theta \sim U(-\pi, \pi)$ to ensure proper simulation. A deep neural receiver can learn the characteristics of this channel by observing many different combinations of realizations of channel coefficients and symbols. For complicated channel models, it may take many realizations and many model updates before a good deep receiver is obtained. Different initial phases can result in additional learning challenges. To better demonstrate this, we plotted some common constellation types in Figure 3.5 in the appendix. One common characteristic of these constellations is that their shape is identical under 90-degree rotations. However, the bit labels the receiver now expects differ after the 90-degree rotation, resulting in bit errors. As such, it is essential that a deep receiver learns to identify the initial phase rotation in this case. Probably, the deep receiver has to build a form of redundancy to identify different cases of rotation and the features needed to identify these cases.

An interesting question is whether there is a more efficient approach. After all, if the deep receiver receives a signal in which only the initial phase differs, it should still be able to recover the original symbols. This is in part due to the fact that a phase shift in y under the previously defined LTI is proportional to a phase change in h under Equation (3.1). It is commonly known that a rotated complex Gaussian random variable is again a complex Gaussian random variable with the same parameters. It also does not matter whether we apply the rotation to x or h, due to the linearity of the convolution operator. This is convenient, as a phase-shifted h in practice can represent, e.g., the same fading channel but with the receiver starting closer together or farther apart. In an ideal world, a phase shift in y should not affect the demodulation performance.

If some function f gives the same results regardless of some set of transformations, the function of f is often referred to as being *equivariant*. The concept of equivariance is commonly studied from the perspective of group theory. Groups are convenient when working with and reasoning about symmetries. A *group* is a set S with an associated operation $\circ : S \times S \to S$ satisfying the following properties:

1. *The identity element*: There exists an element $e \in S$ such that for any $f \in S$ it holds that $e \circ f = f \circ e = f$.
2. *Inverses*: For any element $f \in S$, there exists an inverse $g \in S$ such that $f \circ g = e$.
3. *Associativity*: For any $f, g, h \in S$, it holds that $(f \circ g) \circ h = f \circ (g \circ h)$.

A definition for f being invariant to g can be given as

$$f(x) = f(g \circ x). \tag{3.2}$$

Furthermore, the definition of equivariance can be given as

$$g \circ f(x) = f(g \circ x). \tag{3.3}$$

3.3.2 $\mathbb{T}$-equivariance

Given this framework, we identify the desired type of equivariance. Given a sequence of received samples y, the deep receiver should have equivariant operations with respect to the global phase of y. Adjusting the global phase of y is equivalent to multiplying all values in y by some complex value with a magnitude equal to 1. This group is commonly referred to as the *circle group* and is well studied. It can be defined as

$$\mathbb{T} = \{z \in \mathbb{C} : |z| = 1\}. \tag{3.4}$$

The group action on an OFDM grid is then the multiplication of a complex scalar from $\mathbb{T}$ with every element of the OFDM grid.

To see that this group is appropriate for relative phase equivariance, it is easiest to look at the effect on the frequency domain. In the frequency domain, each component is represented by an amplitude and a phase. As the Fourier transform is linear, multiplying every element in the time domain is equal to multiplying every element in the frequency domain. Multiplications by a complex value with a magnitude of 1 alter only the angle of the complex value i.e. the phase of the frequency component. As such, a transformation by $\mathbb{T}$ results in an identical phase rotation of all components in the frequency domain.

We note that this group is isomorphic to the SO(2) group, which is the group of all 2D matrices with a determinant of 1. This relationship is important because the next question that arises is how we can encode the properties set in (3.2) and (3.3). The work of [42] was among the first approaches to construct a neural network that is G-equivariant with respect to arbitrary discrete groups. Note that a neural network is nothing more than a composition of functions. If each function in this composition is G-equivariant, the whole network is G-equivariant. The combination of the circle group and their group equivariant work form the backbone of our framework.

It is important to note that a G-equivariant function no longer operates on the original space for some set of I/Q samples received y. Assuming an OFDM system, we operate on $\mathbb{C}^2$, which is a grid of frequency over time components. Thus, a set of $\mathbb{T}$-equivariant functions must operate on the semidirect product $\mathbb{C}^2 \rtimes \mathbb{T}$. A practical issue we face is that we cannot represent continuous sets within a computer. Fortunately, the group $\mathbb{T}$ has some convenient properties to overcome this issue. The group is Abelian, which means that the order in which a set of group actions is applied does not matter. Consequently, the circle group has countably infinite proper, closed, discrete subgroups. Each of these subgroups is a

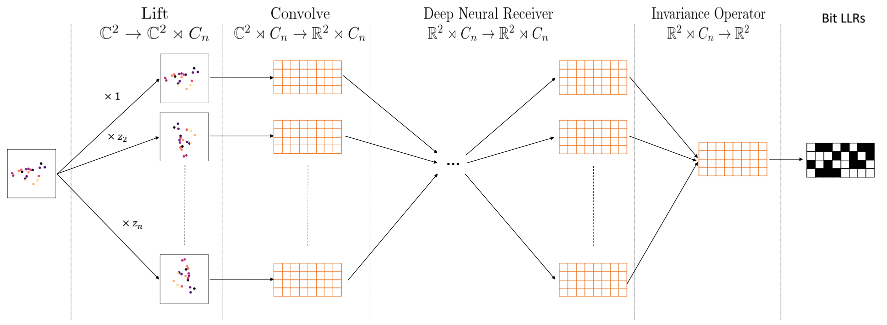

Lift
$\mathbb{C}^2 \to \mathbb{C}^2 \rtimes C_n$
Convolve
$\mathbb{C}^2 \rtimes C_n \to \mathbb{R}^2 \rtimes C_n$
Deep Neural Receiver
$\mathbb{R}^2 \rtimes C_n \to \mathbb{R}^2 \rtimes C_n$
Invariance Operator
$\mathbb{R}^2 \rtimes C_n \to \mathbb{R}^2$
Bit LLRs
$\times 1$
$\times z_2$
$\times z_n$

Figure 3.1: Demonstration of the flow of the equivariant model. A sequence of I/Q samples is lifted to C_n, pointwise convolved to a latent, transformed by the neural receiver, made invariant and finally transformed to bit LLRs.

cyclic group defined by the nth roots of unity, where n is the size of the subgroup. The nth roots of unity can be computed as

$$z_k = \exp\left(\frac{2\pi k i}{n}\right), \quad \text{for } k = 0, \dots \ n-1. \tag{3.5}$$

The nth roots of unity divide the circle into equally large areas, with the first multiplication always being equal to 1 i.e. the identity element. Clearly, any multiplication of these numbers with each other results in another number from that group. Consequently, all of these sets are closed. The notion that these are subgroups of $\mathbb{T}$ is convenient, as building an equivariant network over discrete groups is much simpler.

3.3.3 Constructing the Equivariant Operators

We have identified all the components for the construction of a $\mathbb{T}$-equivariant neural receiver and can thus discuss the process of building one. Generally, the construction of a G-equivariant network follows a three-phase framework. These phases consist of a lifting convolution, group convolutions, and an invariance operator. First, a lifting convolution lifts the original input to the group space. In our case, we raise an OFDM signal from $\mathbb{C}^2$ to $\mathbb{C}^2 \rtimes C_n$ where C_n refers to the cyclic group of order n. Lifting the input to the product space ensures that the model is unconstrained with respect to the functions it can learn. Given the roots of unity, the lifting operation can be defined as

$$y(k, f, t) = z_k \times x(f, t). \tag{3.6}$$

Here, $x(f, t)$ refers to the input at frequency f and timestep t. We note that most deep receivers do not actually operate on $\mathbb{C}^2$, however. Often, the choice is made to first transform the input from $\mathbb{C}^2$ to $\mathbb{R}^2$ to allow the use of recent developments in neural architecture design. We opted to mirror this approach and map our semidirect product $\mathbb{C}^2 \rtimes C_n$ to $\mathbb{R}^2 \rtimes C_n$. The first convolution thus translates the multiple complex-valued input stream to some higher dimensional representation without an explicit interpretation. In our neural network design, we opted to perform an element-wise linear map to perform this transformation, as it ensures equivariance while simplifying system design.

The consequence of this mapping is that the group is now represented as an ordered grid across the group dimension. Each element k corresponds to a specific phase offset of the original signal. If we multiply the original input by some rotation from the C_n group, the feature vectors of the original input would be the same, but their order would change. This result follows straightforwardly from the closed property of the group. Assume that we have a multiplication $\times z_j$ corresponding to a multiplication of some root of unity given the cyclic group C_n. As the group is Abelian, the order in which we perform this group operation does not matter, i.e., whether we apply it on the data first or directly on the vector of group transformations. The vector of group transformations can be written as a vector of multiplications by roots of unity as

$$[\times z_0, \ldots, \times z_{n-1}] \, . \tag{3.7}$$

As we know, this vector spans all of the elements of the group. Applying the same group operation again results in a vector containing all elements of the group as

$$[\times z_k, \ldots, \times z_{k+n-1}] \, . \tag{3.8}$$

A characteristic of the circle group is that given some z_k with $k \geq n$, the value z_k wraps around. Intuitively, a rotation by 3π gives the same result as a rotation by π. Interestingly, the permutation itself has a convenient ordering because of its cyclic nature. If we take the product of a group element with all group elements, the order is retained but shifted until the end of the circle is reached. At that point, it wraps around and the order resumes. Functionally, this means that the order of the feature maps is also cyclic.

To formalize this observation, consider a signal s and a kernel ψ, where we assume both to be sequences operating in C_n. Take $C_n = \{e, z_1, z_2, \ldots, z_{n-1}\}$ as the cyclic group of order n following previous definitions, with e the identity element corresponding to multiplication by 1. Furthermore, define L_{z_m} as the left action of z_m and f_ψ as a linear function operating on s and defined using some kernel ψ. We then arrive at the following theorem:

Theorem 3.1. *$L_{z_m}\left[f_\psi(s)\right](z_i) = f_\psi\left(L_{z_m}[s]\right)(z_i)$ iff f_ψ is a circular convolution.*

Here, z_i is an arbitrary group element corresponding to C_n. The above theorem states that equivariance with respect to C_n is not just guaranteed for circular convolutions but that any linear operation between two functions or sequences operating on C_n necessarily has to be a circular convolution to ensure equivariance. This result demonstrates that using cyclic convolutions for equivariance is not only valid but also the most general and expressive solution possible. We note that most parts of the above result have been studied before in both mathematics and the geometric deep learning literature [12, 15, 142]. As such, we have opted to include the proof for the reader's convenience in the appendix under Section 3.B.

We can thus learn parametrized functions that correlate locally relevant information across a variety of phase offsets. These cyclical convolutions can be implemented by using regular convolutions combined with circular padding. Circular padding implies padding the edges under the assumption that the data repeats cyclically. This padding can thus be implemented efficiently in all deep learning frameworks. Overall, this drastically simplifies the implementation of convolutions that are C_n-equivariant.

Finally, it is necessary to cast the results back to $\mathbb{R}^2$. Common aggregators over C_n, such as the mean and max operators, are sufficient to map from the product space $\mathbb{R}^2 \rtimes C_n$ to the desired latent space in $\mathbb{R}^2$. These are then translated to bit LLRs that can be used in a similar way as other neural receivers. Using such an aggregation method results in an output invariant with respect to C_n. To conclude, we now have all the components necessary to construct a $\mathbb{T}$-equivariant deep neural receiver. Note that this receiver does not depend on the choice of constellation. Any arbitrary constellation can be modeled with a $\mathbb{T}$-equivariant deep neural receiver. The components are visually depicted in Figure 3.1.

3.3.4 A Deep Phase Equivariant Receiver

Given the three operators that we discussed previously, we are now able to construct a neural network that is provably equivariant with respect to the phase of the original signal. To do so, we initially propose a more modern variant of DeepRx [82]. We base many of our design choices on the ConvNeXt architecture [116]. In this work, the authors proposed various changes to common design choices in convolutional neural networks based on the success of the transformer architecture. As this architecture was built for image classification, we made multiple tweaks to make it suitable for OFDM signal processing. This resulted in the following overall design choices.

Activation Functions and Normalization. A common method to construct CNNs was to follow every convolution by both a normalization and an activation function. DeepRx also followed this design principle. As a normalization function such as LayerNorm is also nonlinear, ConvNeXt experimented with reducing the number of activation and normalization operations. They reduced the number of activations to one LayerNorm and one GeLU activation per block and found noticeable improvements. We mimic this design and adjust the activation functions for our ConvNeXt-based architecture.

Inverted Bottleneck Block Structure. Another design choice made to reduce the number of parameters without significantly reducing the expressive power of the architecture is the inverted bottleneck. DeepRx already included depthwise separable convolutions, which are based on the observation that a fully connected layer per kernel element is unnecessary. The inverted bottleneck design principle involves reducing the overall width of the model but temporarily increasing it within a block. This reduces the overall number of parameters, while still allowing the model to induce sparsity in higher dimensions.

Increased Kernel Size. The depthwise separable convolutions in DeepRx each had a kernel size of 3×3 with increasing amounts of dilation. One of the design concepts of the ConvNeXt architecture was to perform fewer convolutions but increase the size of the kernel, that is, one 7×7 instead of three 3×3. In a similar vein, we opt for a 5×9 filter, where 5 corresponds to the number of subcarriers and 9 to the number of OFDM symbols. The number of subcarriers is often much smaller than the number of symbols, hence why we decided on an imbalanced kernel. We found that this change also allowed us to reduce the amount of dilation.

The previous changes have no bearing on the equivariance operators. The equivariant components can be added to the architecture with some small changes. First, the projection layer is added at the beginning of the network. The network is positioned after the discrete Fourier transform (DFT) module to extract the OFDM symbols, but the projection layer could also be applied before because of the linearity of the DFT. An important factor to note is that the input to DeepRx does not consist only of the signal. Both the pilot pattern and the least-squares estimates are included, as well, to improve convergence. These additional inputs must be managed appropriately to ensure that the network remains equivariant. Lifting the signal would consist of copying the signal n times, where n is the number of group elements, and rotating each by its corresponding root of unity. Naively copying and rotating the pilot patterns and least-squares estimates could cause problems. However, since the least-squares estimate is no more than a complex division of the received symbol by the pilot symbol, the neural network input remains valid if we multiply the least-squares estimate by the corresponding root of unity while leaving the pilot pattern unchanged. This is equivalent to rotating the input and then recomputing the least-squares estimates. We do not rotate the ground-truth pilot pattern, as these are predefined and should be independent of the initial phase of the signal.

For each of the n rotations of the group C_n that we include, we concatenate the rotated signal, the pilot pattern, and the rotated least-squares estimate. We then performed an element-wise linear transformation to lift each input to a higher-dimensional space. Afterward, as previously shown, the neural network operates on the semidirect product of $\mathbb{R}^2 \rtimes C_n$. Practically, this means that the input is now a batch of 3D elements. The three dimensions are the subcarriers, OFDM symbols, and group elements, respectively. By elementwise, we mean that we apply the same weights for every subcarrier, OFDM symbol, and cyclic group element.

Afterward, we insert a depthwise layer in every residual block that operates solely on the rotation dimensions. While a 3D-depthwise convolution that operates over all three dimensions is likely more expressive, common deep learning frameworks do

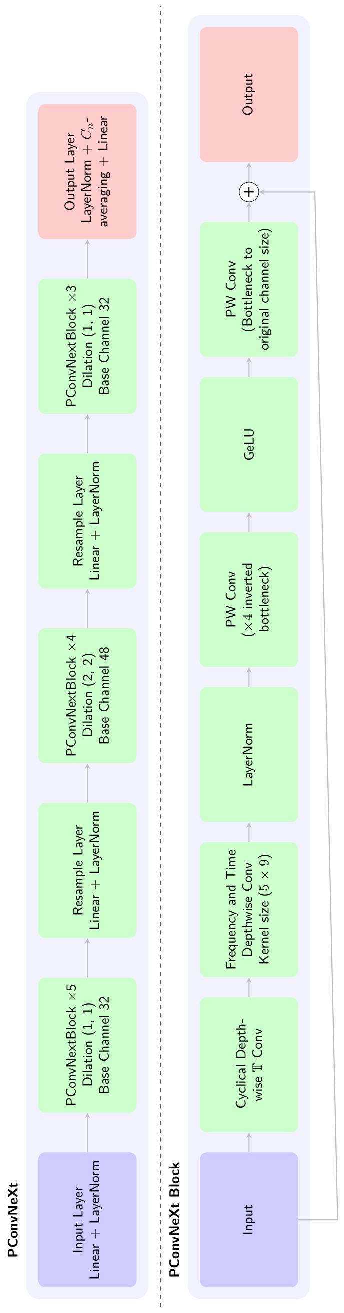

PConvNeXt
Input Layer Linear + LayerNorm
PConvNextBlock ×5 Dilation (1, 1) Base Channel 32
Resample Layer Linear + LayerNorm
PConvNextBlock ×4 Dilation (2, 2) Base Channel 48
Resample Layer Linear + LayerNorm
PConvNextBlock ×3 Dilation (1, 1) Base Channel 32
Output Layer LayerNorm + C_n-averaging + Linear
PConvNeXt Block
Input
Cyclical Depth-wise $\mathbb{T}$ Conv
Frequency and Time Depthwise Conv Kernel size (5×9)
LayerNorm
PW Conv (×4 inverted bottleneck)
GeLU
PW Conv (Bottleneck to original channel size)
+
Output

Figure 3.2: Schematic of the proposed ConvNeXt deep receiver with cyclical convolutions. Note that the cyclical convolutions become identity functions when considering C_1 (no equivariance).

not support GPU-accelerated implementations. By factorizing the 3D-depthwise convolution into 1D and 2D, we significantly improve the speed of the model at a small loss in expressivity. Finally, we implement a mean aggregation layer in the last feature map layer before mapping it to the bit detection layer. The joint architecture can be found in Figure 3.2.

3.4 Experimental Setup

For our experiments, we implemented the model in Figure 3.2. Unless otherwise mentioned, we maintain a specific structure for ease of consideration. The model consists of three stages, where stage 1, 2, and stage 3 consist of 5, 4, and 3 blocks, respectively. The word stage refers to a set of sequential blocks that share the same hyperparameter settings. Here, stages 1 and 3 have 32 channels, and stage 2 has 48 channels. We dilate the filters of stage 2 by a factor 2. This structure allows the network to widen the receptive field, after which it gets a simpler structure to correct.

To evaluate the effect of incorporating C_n-equivariance into a deep neural receiver, we implemented multiple sets of experiments. We used Sionna [83] to implement and evaluate our approach. Sionna provides TensorFlow implementations of physical layer components, as well as multiple 3GPP-compliant channel models that are sufficiently realistic to allow conclusions on practical systems. We performed our experiments with a SIMO setup. This means that there is one transmit antenna and multiple receiver antennas. We chose this setup as it enables us to better isolate the capacity of the model without inducing additional complexity from multiple users. We opt for an OFDM system as it is the standard in 5G communications. We use the Urban Macro (UMa) channel model to generate channel coefficients and Gaussian noise to simulate receiver noise. We chose an UMa channel model as it is one of the most comprehensive channel models available designed for dense urban environments. We randomly sample valid user configurations for each set of channel coefficients. We evaluated all

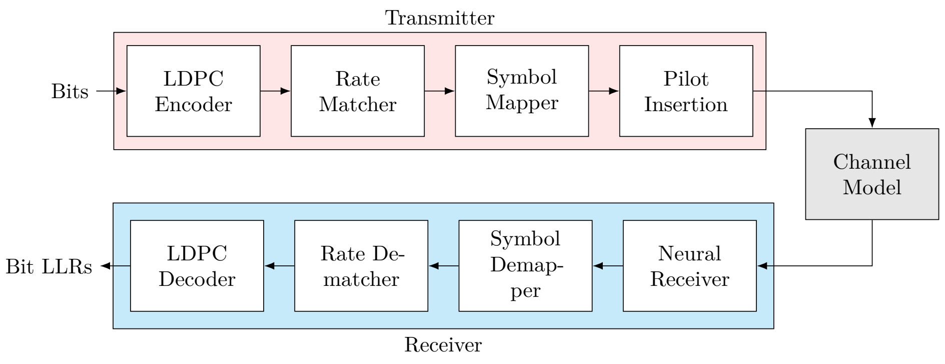


Figure 3.3: The OFDM sender/receiver pipeline that we used to evaluate our model.

methods using the 3GPP-compliant tapped delay line (TDL)-A to C and CDL-A to E channel models. We chose multiple different channel models to ensure that our receiver works generally and not just for the UMa channel model. All channel models have one transmit antenna and two receive antennas. We show the overall communication pipeline in Figure 3.3.

We train each model for 150 epochs with the AdamW [95, 118] optimizer using an initial learning rate of $1e-3$. We use a stepwise scheduler that reduces the learning rate by a factor of 10 in the epochs 100 and 125. We found that this reduction helps stabilize the training. We found that this approach overall gives models sufficient time to converge. All experiments were performed on an A100 40GB GPU and repeated 10 times. After training our models, we evaluate the model based on the BER. We simulate until 500 batches of blocks were processed or 5,000 blocks with errors were detected. For all settings, we include both the results under least-squares estimation and when the channel is known (perfect csi). These serve as upper and lower bounds on the BER. If a neural receiver is worse than the LS estimate, we consider it non-functional since it receives the LS estimate as input. The code used to experiment is available at the following link.[1]

3.5 Results

Main test bench. In Figure 3.4, we plot the BER curve of three methods for 16-QAM. We specifically focus on 16-QAM, as it is one of the most common constellation types. We included the constellations for visualization in Appendix 3.A in Figure 3.5. DeepRx [82] serves as the baseline for a strong deep neural receiver, ConvNeXt is our approach without any group convolutions over a subgroup of $\mathbb{T}$, and ConvNeXt-C_5 adds the cyclical convolutions over C_5. We chose to use the subgroup C_5 for the equivariant approach unless otherwise mentioned. We chose C_5 as our results indicated during validation that this group achieves most

[1]Anonymous for now, will be made available in the definitive version.

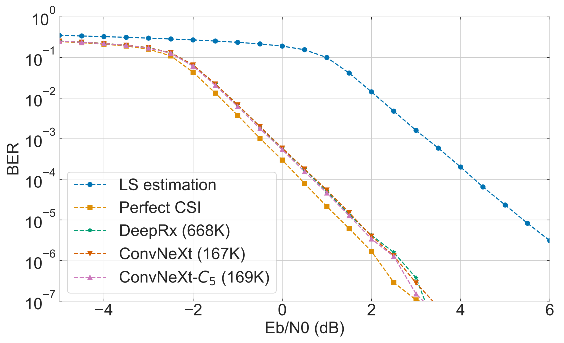
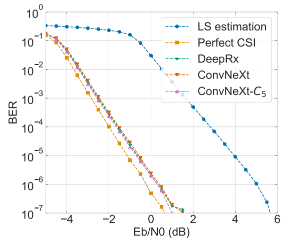
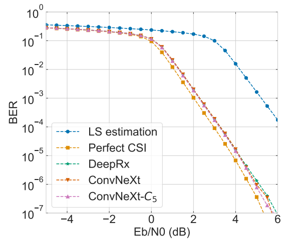


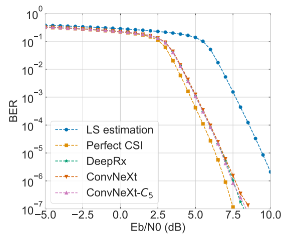
Figure 3.4: Comparison of the coded BER performance of DeepRx, ConvNeXt, and ConvNeXt-C_5 over 10 runs. The parameter counts are noted in the legend.



(a) Performance of QPSK modulation



(b) Performance of 64-QAM modulation



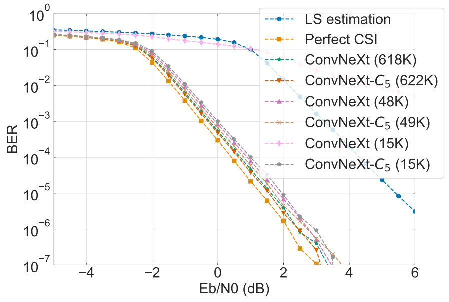
(c) Performance of 256-QAM modulation

Figure 3.5: BER curve comparison of the (a) QPSK, (b) 64-QAM and (c) 256-QAM constellation types.

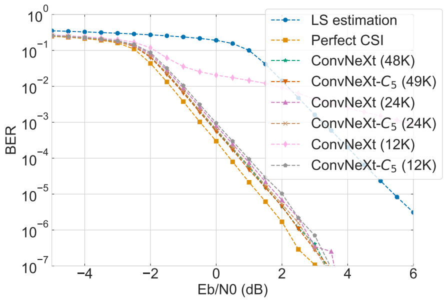
of the performance gain without inflating the compute requirements too much. The results show that the ConvNeXt and DeepRx approaches have similar BER curves, despite the fact that ConvNeXt has less than a quarter of the parameters. Furthermore, ConvNeXt-C_5 achieves a BER curve that is tighter than both. This, again despite having less than a quarter of the parameters than the DeepRx model and slightly more than the ConvNeXt approach. We evaluated the mean BER of these curves and found that all differences were significant other than the difference between DeepRx and ConvNeXt. ConvNeXt-C_5 therefore performs significantly better than ConvNeXt with a similar number of parameters and DeepRx with more than four times fewer parameters.

Translation to other constellation types. In Figure 3.5, we further compare ConvNext with ConvNeXt-C_5 for other common constellation types different from 16-QAM. Comparison to constellation types other than 16-QAM is interesting because of their different geometric properties. For instance, QPSK differs from the other constellation types in that it can be created by pure phase modulation. Interestingly, for QPSK, we see that the C_5 equivariant approach achieves the largest performance increase compared to the ConvNeXt model without C_5-equivariance. Although 64-QAM and 256-QAM both achieve a reduction in BER when including C_5-equivariance, we see that this reduction is more pronounced for QPSK. A possible explanation may lie in the lack of amplitude shifting that is present in the other constellation types we tested.

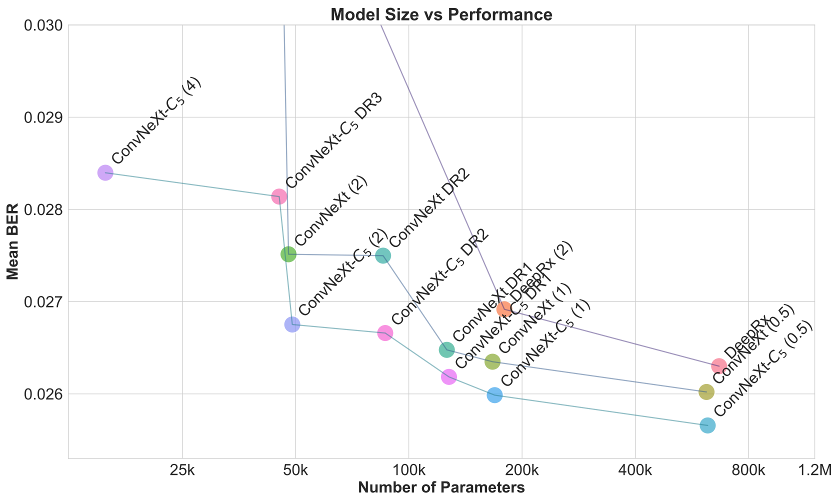
Impact of model size. We chose model sizes of 167K and 169K in previous experiments, as we found that these models present a good trade-off between parameter count and performance. However, the size of the model and the amount of computation needed for inference is often more important for deep receivers than small improvements in BER. If a receiver makes slightly more block errors but can process twice as many blocks in the same amount of time, the resulting bandwidth will be higher overall. That is why we opted to adjust the model size for both the ConvNeXt and ConvNeXt-C_5 receivers. We report the results of these experiments in Figure 3.6. In (a), we evaluated the adjustment of only the



(a) Results of channel width adjustment experiments.



(b) Results of depth reduction experiments.



(c) Mean BER model comparison.

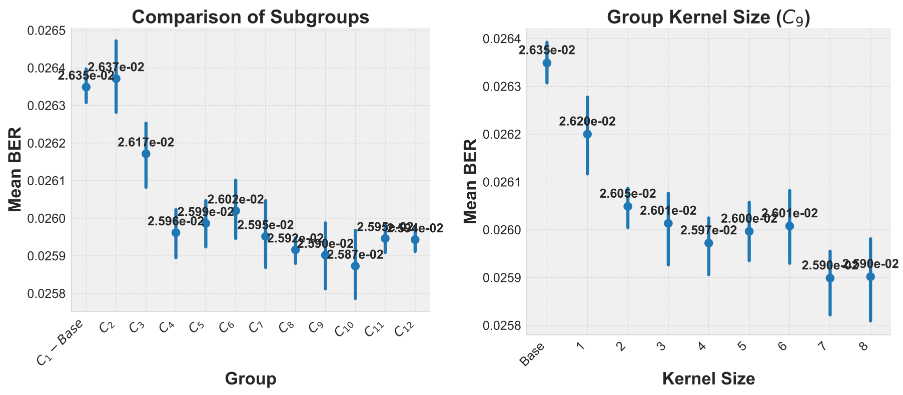
Figure 3.6: Experiments on model sizes: (a) impact of reducing model size, (b) impact of reducing model depth, (c) mean BER Pareto front illustrating trade-offs between model size and performance. DRn refers to all stages being reduced by n, where (n) refers to a division of all channels by n.

channel width. For example, the 618K ConvNeXt model is the result of doubling all channel widths in the original model in Figure 3.2. We see that an increase in channel width results in an improvement in BER performance. We tested statistical significance in the mean BER and found that the difference between all models was significant.

However, we see anomalous behavior for the ConvNeXt (15K) model compared to the other models. At approximately 1.5 Eb/N0 (dB), the performance becomes worse than the LS estimate. We evaluated the runs and found that the model did not converge for some of them. We experimented with different sets of hyperparameters, but were unable to find one with better convergence properties. What is interesting is that the ConvNeXt-C_5 (15K) model managed to converge for all 10 runs, thus displaying better convergence properties at a similar number of parameters. Once we reduced the channel size further than the (15K) models, neither the ConvNeXt nor the ConvNeXt-C_5 models were able to converge.

Reducing the size of the model by changing the width of the channel maintains the same receptive field. To further evaluate the impact of parameter reduction, we opted to leave the channel width the same and reduce the number of blocks instead. We show the results in Figure 3.6b. Here, every reduction in parameters shows a reduction of 1 block per stage. For example, the 126K model has 4, 3, and 2 blocks in the three stages, as opposed to 5, 4, and 3. Again, we see that the larger models are more effective than the smaller ones. Interestingly, we again see divergence at the smallest model size. However, it occurs here for the 45K model when reducing the depth, as opposed to at 15K for reducing the channel width. This means that the model becomes unusable with a greater number of parameters than when width reduction is performed. The ConvNeXt model C_5 (45K) still manages to converge, demonstrating a higher capacity than the model without inductive bias.

Finally, we plotted the mean BER across the curve for all models compared to the size of the model in Figure 3.6c. There are multiple interesting observations to take from this figure. First, we see that both ConvNeXt (0.5) and ConvNeXt-C_5 (0.5) are (significantly) better in terms of mean BER at similar parameter sizes compared to the DeepRx model. What is also interesting is that ConvNeXt-C_5 (1) is similar in mean BER compared to ConvNeXt (0.5), despite having almost 4 times fewer parameters, with ConvNeXt-C_5 (1) achieving a (nonsignificantly) lower mean BER. As the inclusion of $\mathbb{T}$ equivariance induces an extra computation that scales linearly with the size of the discrete group, this result is highly interesting. We also included smaller versions of DeepRx, named DeepRx (2) and DeepRx (4). We see that the difference between DeepRx (2) and ConvNeXt (1) is larger than between DeepRx and ConvNeXt (0.5). This difference is even greater between ConvNeXt (2) and DeepRx (4). This demonstrates that ConvNeXt scales more gracefully in low-parameter regimes. Further, we see that depth reduction is a less efficient way to reduce the size of the deep neural receiver than channel width reduction, especially for smaller models. Finally, we see that ConvNeXt (4) is fully unable to converge, and we therefore chose to display it as a vertical line. Overall, we can see from the figure that the incorporation of the C_5-equivariance provides



(a) Model performance across different group sizes.

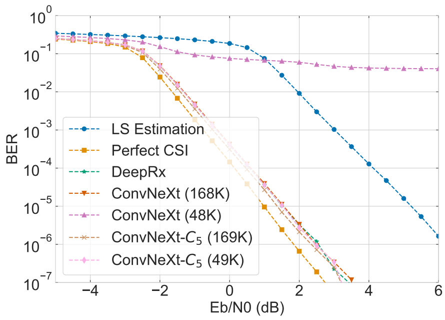
(b) Model performance based on filter size variations.

Figure 3.7: Influence of group size and filter size on model performance: (a) Performance across varying group sizes, (b) Performance under C_8 based on different filter sizes. The bars contain the 95% confidence interval.

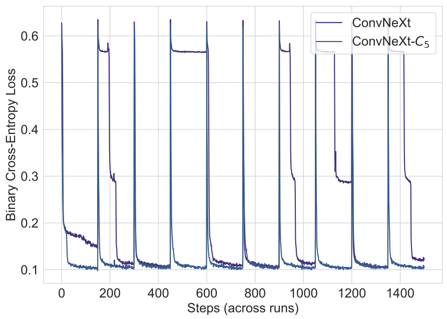
significant mean BER improvements at all the sizes of the models tested.

Impact of group size. We mentioned earlier that we use the group equivariant approach with C_5 as the discrete approximation of $\mathbb{T}$. We did so because we found that this approach provided a sweet spot between performance and the number of group elements that we needed to evaluate. To evaluate the impact of the choice of subgroup, we look at the following two variations. First, we tested the effect of different subgroups of $\mathbb{T}$ on the mean BER. Here, the mean BER refers to an average over the entire BER curve. We compared these with a ConvNeXt of similar size, denoted $C_1 - Base$, in Figure 3.7a. We can take multiple points from this comparison. First, we see that there is no performance improvement for C_2. While C_2 performs aggregation over two phase offsets, the kernel size is still 1. This means that no functions are learned across group elements. Second, we see that a small group such as C_3 is sufficient to achieve a significant performance increase. Interestingly, the next largest performance improvement comes from the transition of C_3 to C_4. Afterwards, no significant improvements are made in increasing the size of the subgroup. The 16-QAM constellation being identical under 90-degree rotations may explain why the BER reductions are maximal under C_4. Overall, our results indicate that the majority of the mean BER improvement is likely to come from the inclusion of the equivariance. Further, subgroup sizes larger than the possible phase ambiguities do not seem to make a significant improvement.

Secondly, we come back to the advantage of using cyclical convolutions. When performing convolutions over $\mathbb{T}$, the kernel size can be smaller than the size of the subgroup. This provides additional flexibility in the modeling since separate modeling choices can be made for the subgroup and the kernel size. To test this



(a) Average performance across different pilot parameter sizes.



(b) Loss curve over the 10 runs for the ConvNeXt (48K) and ConvNeXt-C_5 (49K) models.

Figure 3.8: Analysis of using a single pilot: (a) average performance when only including a single pilot, (b) loss plot illustrating the convergence of the smaller single-pilot models.

effect, we trained multiple ConvNeXt-C_9 receivers with different kernel sizes. We show their mean BER in Figure 3.7b. Note that Base again corresponds to a ConvNeXt without any group convolutions. What is interesting to note first is the result for a kernel size of 1. One of the largest reductions in BER when increasing the kernel size occurs when the group equivariance is added. The only difference between this model and a base ConvNeXt is the aggregation step over C_9. The group elements are, otherwise, never mixed throughout the model, and the parameter counts are identical for the base model and the kernel size 1 model. Also interesting is that we did not see this effect for C_2. This indicates that aggregation over sufficient group elements is necessary to achieve this effect. We again see the largest reduction in BER when moving from a kernel size of 1 to 2. Note that the effective kernel size of the overall model in this configuration is 13 due to the model containing 12 group convolutions. Interestingly, only for kernel sizes 7 and 8 do we find a significant difference in mean BER compared to a kernel size of 2. Thus, we again find that the C_9-equivariance already significantly improves the model without accessing patterns across elements of C_9. A kernel size of 2 that is sufficient for most of the mean BER reduction indicates that pattern recognition across elements of C_9 aids model performance.

Impact of number of pilots. To expand on whether $\mathbb{T}$ equivariance adds to the expressivity of the network, we evaluated it in a more challenging setting. In previous experiments, we placed the pilots in a Kronecker pattern at indices 2 and 11. This provides the network with information about the time-varying component of fading. As such, we opt to evaluate our approaches when we leave out the pilots at position 11. This scenario is no longer realistic, as performing an interpolation of the LS estimates between indices 2 and 11 is no longer possible. Thus, it makes time-varying fading impossible to correct over longer periods of time. However, its primary purpose is to provide an interesting stress test.

We show the BER curves in Figure 3.8a. We see that both ConvNeXt (168K) and ConvNeXt-C_5 (169K) are able to achieve strong performance over the given channel models. Interesting is the divergent behavior of the ConvNeXt (48K) model; a behavior that we do not see for ConvNeXt-C_5 (49K). To get a better idea of what happens there, we plot the loss curves for both models across the runs in Figure 3.8. We observe that the ConvNeXt model struggles to converge in some runs. By averaging over these failed runs, the resulting BER curve becomes unusable. We attempted more hyperparameter tuning, but were unable to find a setting where the ConvNeXt model converges consistently. Thus, we find that the C_5-ConvNeXt model can operate at smaller sizes compared to ConvNeXt before having convergence problems.

3.6 Discussion and Conclusion

To recap, we proposed an element-wise C_n-equivariant DNN suitable for signals that benefit from phase equivariance. We applied our method to build a deep neural receiver that is equivariant with respect to variations in initial observation times and antenna angles. We implemented both a ConvNeXt and a C_5-equivariant ConvNeXt model, and tested them in a variety of settings. The core results can be summarized as follows.

1. For 16-QAM, the ConvNeXt approach performs similarly to DeepRx with four times fewer parameters. The group equivariant ConvNeXt-C_5 model outperforms both significantly in terms of mean BER.
2. The C_5-equivariant ConvNeXt networks significantly outperform base ConvNeXt models at all model sizes in terms of mean BER. It is also, on average, more capable in tiny model settings, due to better convergence properties.
3. The $\mathbb{T}$ equivariant approach performs relatively better for QPSK than for other constellation types.
4. We found that most performance benefits are achieved at C_4 and that larger subgroups of $\mathbb{T}$ do not provide significant benefits. We also found a small kernel size of 2 to be sufficient in a deep model under a group of C_9.

We found that a group equivariant approach performed well with a minimal increase in the number of parameters. An important aspect to discuss is whether the performance increase is the result of the group equivariance or other changes in the network. Some interesting implications arise from the ablation of the influence of the kernel size in this regard. It shows that computing the group elements and averaging at the end is sufficient for a major performance improvement. Furthermore, a kernel size of 2 is sufficient to achieve most of the BER reduction, which may imply that the core gain lies in the $\mathbb{T}$-equivariance.

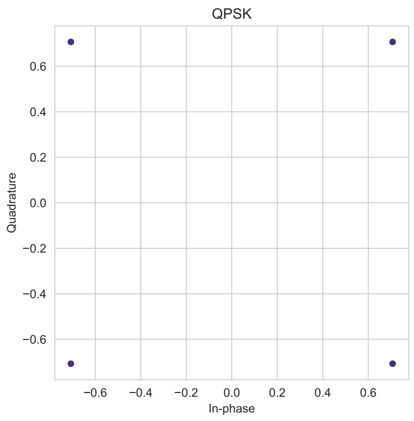
Figure 3.5 also provides an interesting perspective on this question. These figures demonstrate that the performance increase depends on the constellation type as well. The larger the constellation, the less group equivariance improves the BER relatively. In the context of phase equivariance, this makes sense, as larger QAM constellations induce additional complexity that is not dependent on the initial phase.

To conclude, we proposed an $\mathbb{T}$-equivariant deep receiver and demonstrated that the $\mathbb{T}$-equivariance provides performance benefits at similar numbers of parameters. The main limitation of our approach is an increase in computation that scales linearly with the size of the subgroup, which can make it impractical to implement in its current form. We note that for larger model sizes, we found that the $\mathbb{T}$-equivariance can provide similar performance at the same level of computation. We found that a C_4-equivariant model above a certain size achieves a similar BER to a C_1 model with 4x the number of parameters. We emphasize that we consider our core contributions to lie in the method of constructing a relative phase equivariant deep receiver and demonstrating that this property can improve the performance of deep neural receivers.

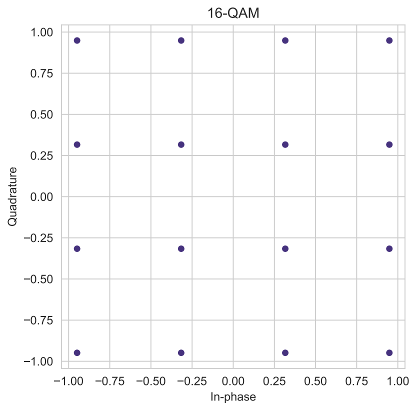
As our work is intended as preliminary work on the use of inductive biases to improve deep neural receiver design, future work could focus on the following aspects. The main aspect to look into is the improvement of compute times. The method scales linearly with the number of group elements, greatly limiting its practicality. Removing the need to evaluate all subgroup elements while retaining equivariance, for example, would be interesting. Research into other inductive biases of physical layer communications also holds potential. More practical limitations of our work include the lack of over-the-air validation and integration with traditional systems. Future work could look at both of these aspects. The approach would also enable direct generation of CSI, which is another interesting aspect to consider for backward compatibility with legacy systems.

Appendix

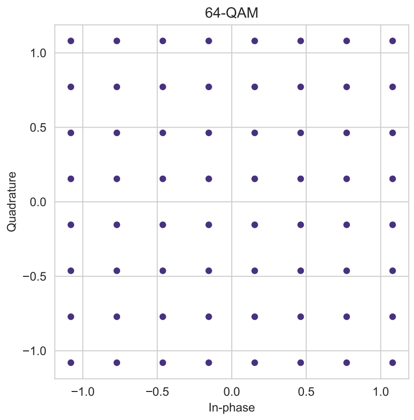
3.A Additional Figures



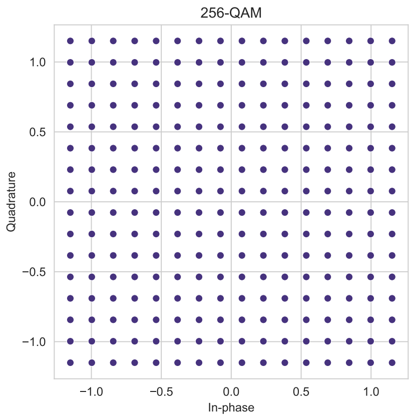
(a) QPSK



(b) 16-QAM



(c) 64-QAM



(d) 256-QAM

Figure 3.A.1: The QAM constellations used in our work.

3.B Circular Convolution Proof

As stated previously, we prove Theorem 3.1 here.

Proof. As a reminder, we consider a function f_ψ parametrized by kernel ψ that operates on a signal s , where we assume s and ψ to be sequences over C_n. In addition, we take $C_n = \{e, z_1, z_2, \ldots, z_{n-1}\}$ as the cyclic permutation group of order n with e the identity element corresponding to multiplication by 1. In addition, L_{z_m} is the left action of z_m. We will prove this theorem by showing both directions of the if and only if statement.

First, we prove that if f_ψ is a circular convolution, then $L_{z_m}\left[f_\psi(s)\right](z_i) = f_\psi\left(L_{z_m}[s]\right)(z_i)$

As f_ψ is a circular convolution, it follows that

$$f_\psi(s)(z_i) = \sum_{j=0}^{n-1} f(z_{j \mod n})\psi(z_{i-j \mod n}). \tag{3.9}$$

Let us take both sides of the equality.

$$\begin{aligned} L_{z_m}\left[f_\psi(s)\right](z_i) &= f_\psi(s)(z_{i-m}) &(3.10)\\ &= \sum_{j=0}^{n-1} s(z_{j \mod n})\psi(z_{i-m-j \mod n}). &(3.11)\end{aligned}$$

We can now substitute $j' = j + m$. Since the convolution is circular and thus periodic, we can rearrange the elements of sum such that we again sum j' from 0 to $n-1$, giving us the following

$$\begin{aligned} &= \sum_{j'=0}^{n-1} s(z_{j'-m \mod n})\psi(z_{i-j' \mod n}) &(3.12)\\ &= \sum_{j'=0}^{n-1} L_{z_m}\left[s(z_{j' \mod n})\right]\psi(z_{i-j' \mod n}) &(3.13)\\ &= f_\psi(L_{z_m}[s])(z_i). &(3.14)\end{aligned}$$

We thus find that if f_ψ is a circular convolution, it follows that $L_{z_m}\left[f_\psi(s)\right](z_i) = f_\psi\left(L_{z_m}[s]\right)(z_i)$.

Second, we prove that if $L_{z_m}\left[f_\psi(s)\right](z_i) = f_\psi\left(L_{z_m}[s]\right)(z_i)$ then f_ψ must be a circular convolution. We note that the function f_ψ is a linear map and we can thus represent it as

$$f_\psi(s) = As. \tag{3.15}$$

Here, A is some arbitrary linear matrix corresponding to the function f_ψ. We further note that we can represent the left action L_{z_m} as a a permutation matrix P

that respects the cyclic permutation equivariance. This follows straightforwardly from the fact that L_{z_m} operates on the cyclic permutation group C_n.

Thus, we can write the equivariance condition as $P(As) = A(Ps)$ for all s. Due to commutativity, it follows that $PA = AP$. As P is a permutation matrix corresponding to the cyclic permutation, it follows that P is a circulant matrix. As circulant matrices commute with each other, this means that A must be a circulant matrix for $PA = AP$ to hold.

As A is a circulant matrix, we can represent any arbitrary matrix A as a circular convolution using elements of ψ.

$$A = \begin{bmatrix} \psi(e) & \psi(z_{n-1}) & \psi(z_{n-2}) & \dots & \psi(z_1) \\ \psi(z_1) & \psi(e) & \psi(z_{n-1}) & \dots & \psi(z_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \psi(z_{n-1}) & \psi(z_{n-2}) & \psi(z_{n-3}) & \dots & \psi(z_e) \end{bmatrix} \tag{3.16}$$

It thus follows that f_ψ must be a circular convolution parametrized by ψ for the equivariance condition to hold. In other words, if $L_{z_m}\left[f_\psi(s)\right](z_i) = f_\psi\left(L_{z_m}[s]\right)(z_i)$, then f_ψ is necessarily a cyclic convolution.

Overall, we thus find that $L_{z_m}\left[f_\psi(s)\right](z_i) = f_\psi\left(L_{z_m}[s]\right)(z_i)$ iff f_ψ is a circular convolution. $\square$

CHAPTER 4

The Computation of Generalized Embeddings for Underwater Acoustic Target Recognition using Contrastive Learning

Based on [6]:
Hilde Hummel, Arwin Ganskoele, Sandjai Bhulai, and Rob van der Mei
The Computation of Generalized Embeddings for Underwater Acoustic Target Recognition using Contrastive Learning
In review.

Abstract

The increasing level of sound pollution in marine environments poses an increased threat to ocean health, making it crucial to monitor underwater noise. By monitoring this noise, the sources responsible for this pollution can be mapped. Monitoring is performed by passively listening to these sounds. This generates a large amount of data records, capturing a mix of sound sources such as ship activities and marine mammal vocalizations. Although machine learning offers a promising solution for automatic sound classification, current state-of-the-art methods implement supervised learning. This requires a large amount of high-quality labeled data that is not publicly available. In contrast, a massive amount of lower-quality unlabeled data is publicly available, offering the opportunity to explore unsupervised learning techniques. This research explores this possibility by implementing an unsupervised Constrastive Learning approach. Here, a Conformer-based encoder is optimized by the so-called Variance-Invariance-Covariance Regularization loss function on these lower-quality unlabeled data and the translation to the labeled data is made. Through classification tasks involving recognizing ship types and marine mammal vocalizations, our method demonstrates to produce robust and generalized embeddings. This shows to potential of unsupervised methods for various automatic underwater acoustic analysis tasks.

4.1 Introduction

The ocean environment is polluted by the increased level of artificially induced sounds. This negatively impacts the ocean health directly. For this reason, monitoring underwater acoustics is crucial to protect our ocean environments. By monitoring the underwater sound, the noise sources responsible for pollution can be mapped. One such system to monitor ocean sound is the implementation of a passive acoustic monitoring system. Within such a system, hydrophones are placed to passively listen to the underwater acoustics. These acoustics originate from multiple sources, such as ships, marine mammals, surface waves, and rain. Due to the mix of acoustic sources and complexity of the acoustic ocean environment, the analysis of underwater becomes challenging. In addition, these systems generate a large amount of data, creating the need for automatic classification of underwater acoustic sounds, called UATR. To automate UATR, machine learning has shown its potential [72, 86, 112, 160].

Until now, the development of UATR algorithms has focused on supervised learning, which requires a vast amount of high-quality labeled acoustic data for training. Unfortunately, the amount of publicly available labeled acoustic data is limited. In contrast, a large amount of unlabeled underwater acoustic data is available to the public. Until now, no research has been conducted on the application of these lower-quality data utilizing unsupervised learning models for UATR. One promising unsupervised approach to leverage this unlabeled acoustic data is CL. This method tries to maximize the similarity between positive data samples, while minimizing the similarity for unrelated data samples. Within the computer vision domain, several CL methods are proposed to achieve remarkable performance, such as SimCLR [40] and Momentum Contrast (MoCo) [78].

In addition to computer vision tasks, this specific method has excellent performance in general audio tasks, as seen in [145] and [59, 131]. Furthermore, multiple studies have already implemented a form of CL to recognize the unique acoustic signatures of ships. For example, [130] proposed a ResNet-based encoder and optimized this in a supervised setting. In this study, they forced recordings of the same ship to be close together in the embedding space while pushing recordings of other vessels further away. They built a distance-based classifier on top of their CL framework to avoid overfitting under limited data. Similarly, [157] proposed a three-layer Multi-Layer Perceptron (MLP) optimized using the supervised contrastive loss function [94]. They did not include any augmentations and only used the label information, achieving over 98% accuracy on ship type classification. Another supervised CL framework for ship type classification is proposed in [181]. In this study, they proposed two separate ResNet encoders, where both encoders process two different transformations of the same audio. They implemented a combined loss of cosine similarity and the cross-entropy loss, achieving 80% accuracy. In addition to these supervised methods, [162] proposed a few-shot learning method to cope with limited labeled data. In this study, they proposed a four-stage training scheme inspired by SimCLR [40] and Bootstrap Your Own Latent (BYOL) [65]. They treated a dataset of labeled ship recordings as partially unlabeled and explored the impact of reducing the number of labeled data samples on their proposed method. In this study, they showed that they only needed 10% of the labels to achieve the desired performance. A similar few-shot learning method is proposed in [113] using a MoCo-based framework to achieve over 96% accuracy.

Although prior UATR studies report high recognition accuracy scores, their ability to generalize to similar tasks remains unexplored [127]. Generalization of models is needed to transfer the knowledge captured within the model to related tasks, making the generalized model applicable to various UATR tasks. It is an important property to enable more broadly applicable UATR. This study aims to address this limitation by generating generalized embeddings from publicly available unlabeled data. These embeddings are optimized by the implementation of CL. The ability of the model to generalize is tested by implementing a simple classifier on Deepship [89], ShipsEar [147] for ship type classification, and The Best of Watkin's [148] for marine mammal sound classification. The contribution of this paper is three-fold:

- The first implementation of unsupervised CL on a separate unlabeled underwater acoustic dataset is presented.
- A translation from this raw unlabeled data from a single hydrophone to a labeled classification task is made.
- The unsupervised CL approach generates more robust and generalized embeddings compared to the supervised CL approach.

Altogether, this work shows the potential of an unsupervised CL framework for automatic UATR trained on highly available unlabeled data. This removes the dependency on a vast amount of high-quality labeled data and can be applied to various UATR tasks. Furthermore, this proof-of-concept can be extended by including more freely available unlabeled underwater acoustic data and by introducing more underwater acoustic analysis tasks.

4.2 Material and Methods

In this paper, a framework for automatic UATR is proposed using lower-quality unlabeled data which is freely accessible. A CL pipeline is proposed to generate generalized and informative embeddings, where the proposed architecture is visualized in Figure 4.2.1. In this architecture, an encoder is defined to compute the embeddings, and an expander is added to project these embeddings to a higher-dimensional space for optimization. After training this pipeline, the generated embeddings are used to fit simple task-specific classification models.

4.2.1 Data

In this work, both labeled and unlabeled data are utilized to train and evaluate the model.

Unlabeled datasets

For the model's training, we selected a single hydrophone from the Ocean Network Canada (ONC) dataset [36]. The hydrophone is located in the bay near Vancouver and is characterized by abundant shipping activity (longitude -123.43 West and latitude 49.04 North at 298 m depth). This hydrophone was selected to align with the same site as the Deepship benchmark dataset. From the recordings of this selected hydrophone, multiple 5-minute recordings were selected within the time range of September 2019 to December 2020. To ensure comprehensive

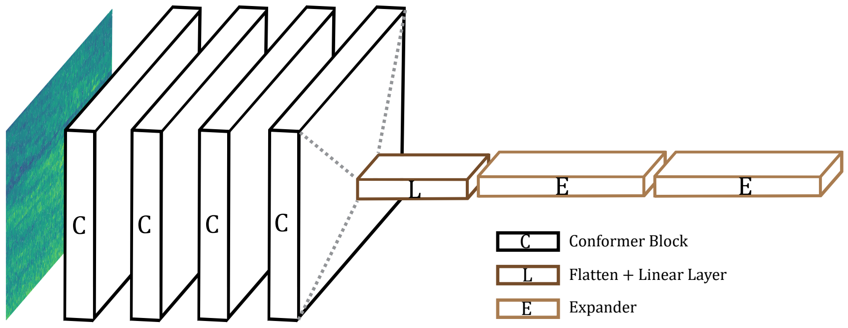


Figure 4.2.1: The proposed architecture for the unsupervised framework.

representation, the selection procedure ensured equal representation of every month and time of day (morning, afternoon, evening, and night), resulting in 30 GB of raw data.

Labeled datasets

For the evaluation of the proposed method, three labeled datasets were selected as benchmark. All datasets in the evaluation are commonly used, making it easier to compare the results with previous research [86].:

1. **Deepship** [89]: This benchmark dataset contains 47 hours and 3 minutes of single-ship recordings categorized into four classes.
2. **ShipsEar** [147]: A smaller dataset recorded near Port Ria de Vigo, totaling 3 hours and 8 minutes of single-ship recordings categorized into five classes.
3. **The Best of Watkin's** [148]: A marine mammal vocalization dataset categorized into 45 classes.

The Deepship dataset was split into a training and test set using a time-wise split, ensuring that the training set represents the past and the test set represents the future. The recordings until November 2017 were used as training data and the recordings from December 2017 onward as test set. A time-wise split ensures that the proposed framework is most representative of practical systems, making a better evaluation for real-world deployment. However, due to this split, almost 60% of the data samples in the test set are newly seen ships. This requires the model to learn generalizable embeddings to correctly classify these previously unseen ships. This property is crucial for generalized models, as they need to be applicable to various downstream tasks and therefore robust to distribution shifts [169]. The remaining labeled datasets had no available time information and, therefore, were divided into 80% training data and 20% test data.

Preprocessing

After the datasets were split, the audio samples were downsampled to 16 kHz. Doing so reduces the dimensionality of the data while minimizing the loss of important information captured within the audio. From these downsampled audio, nonoverlapping two-second windows were created.

4.2.2 Augmentations

In a CL framework, it is necessary to define positive samples for training. These samples are defined by applying two individual augmentation functions to a single audio fragment in the time domain. These two augmentation functions are randomly chosen from a family of augmentation functions. This family consists of four different augmentation functions, namely:

1. **No augmentation:** Retains the original audio.

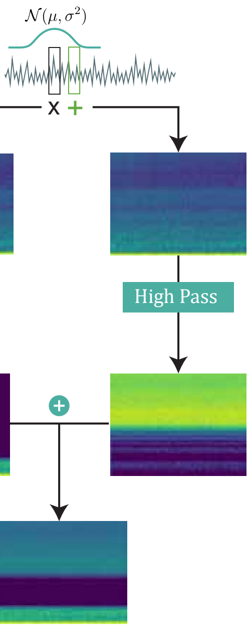


Figure 4.2.2: The MixUp augmentation pipeline.

2. **Gaussian noise:** Adds random Gaussian noise and constrains the output SNR between 0.3 dB and 0.5 dB. The addition of Gaussian noise simulates more noisy different underwater acoustic environments.
3. **Low-pass filter:** Filters the input audio by a maximum of 1 kHz as the cutoff frequency. As stated in [86], the most discriminative information for the classification of the type of ships lies between 50 Hz and 1 kHz. Therefore, a low-pass filter is a representative augmentation function.
4. **Mixup strategy:** Combines two filtered audio samples and is inspired by the work of [165] who presented Temporal Neighborhood Coding for positive sample definition in time series data. The original audio sample is filtered by the 1 kHz low-pass filter as previously described. A second audio sample is selected by normal distribution over time, using the center time of the recording as μ and 50 seconds as σ. From this distribution, another two-second audio sample is drawn. This drawn sample is then filtered by a high-pass filter from 1 kHz-8 kHz. Finally, these filtered samples are added, forming a mixed audio sample. A MixUp strategy is commonly applied in audio pipelines [93, 183], this adds more contrast to lower-contrast spectrograms.

The complete mixup pipeline is illustrated in Figure 4.2.2.

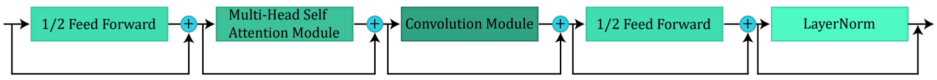


Figure 4.2.3: A single Conformer Block.

4.2.3 Conversion to Mel spectrogram

The augmented audio is converted to a Mel spectrogram. This is a time-frequency representation of the time-domain audio convolved by the Mel scale. This Mel scale comprises multiple half-overlapping triangular filters designed to mimic the human perception of sound. Each triangular filter is centered at frequency f_{mel} which is defined as:

$$f_{mel}(Hz) = 2595 \times \log_{10}\left(1 + \frac{f(Hz)}{700}\right). \tag{4.1}$$

In this research, the number of Mel filters was set to 128.

4.2.4 Architecture

The architecture of the proposed method combines an encoder with an expander (see Figure 4.2.1). In this research, a small architecture is chosen to create a proof of concept. In this regard, the encoder is defined by four Conformer blocks [67], each including feedforward layers, four multihead self-attention heads, and convolutional modules with a kernel size of 31 (see Figure 4.2.3). Previous research has shown the potential of the Conformer model in audio classification on AudioSet [156]. The resulting embeddings are flattened to a 2,048-size vector and passed through a linear layer. The architecture is finalized by the expander, composed of a two-layer MLP of dimension 8,196.

4.2.5 Loss function

The unlabeled training dataset can be viewed as a single long recording. Since there is no information on the content of this recording available, it is difficult to assign negative samples as proposed in the original SimCLR paper [40]. For this reason, the Variance-Invariance Covariance Regularization (VICReg)[25] loss function is implemented. Here, no prior definition of negative samples is needed. The loss function consists of three components: the invariance, variance, and covariance component. A weighted average is taken from these components to define the loss function. The first component is the variance component ($v(Z)$), where Z is the batch:

$$v(Z) = \frac{1}{d}\sum_{j=1}^{d} abs\big(\gamma - S(z^j, \epsilon)\big), \tag{4.2}$$

where d is the number of features and z^j is the specified feature value of all batch samples. γ is a constant target value for the standard deviation, which is fixed at

1 as in the original paper. Finally $S(z^j, \epsilon)$ is given by:

$$S(x, \epsilon) = \sqrt{Var(x) + \epsilon}, \tag{4.3}$$

where ϵ is a small constant fixed to 0.0001. This function encourages the variance to be the same for all j in the current batch. The covariance matrix of Z is defined as:

$$C(Z) = \frac{1}{n-1} \sum_{i=1}^{n} (z_i - \bar{z})(z_i - \bar{z})^T, \tag{4.4}$$

where

$$\bar{z} = \frac{1}{n} \sum_{i=1}^{n} z_i. \tag{4.5}$$

The regularization of the covariance can then be defined as:

$$c(Z) = \frac{1}{d} \sum_{i \neq j} [C(Z)]^2_{i,j}. \tag{4.6}$$

This term encourages the off-diagonal components to be close to 0. This decorrelates the variable of each embedding, which prevents informational collapse. The invariance component is given by the negative cosine similarity:

$$s(Z_i, Z'_i) = -\frac{Z_i \cdot Z'_i}{\|Z_i\| \|Z'_i\|}. \tag{4.7}$$

Here, Z' is defined as the augmented version of Z. These components are then combined in a weighted average to define the overall loss function:

$$l(Z, Z') = \lambda s(Z, Z') + \mu[v(Z) + v(Z')] + v[c(Z) + c(Z')]. \tag{4.8}$$

In this loss function, the scaling components are hyperparameters that need to be tuned. The overall objective function is given by a summation over batches I in dataset D and over augmentations functions t and t' derived from the augmentation family T:

$$L = \sum_{I \in D} \sum_{t, t' \ T} l(Z^I, Z'^I). \tag{4.9}$$

4.2.6 Parameter selection

The proposed model is optimized for 30 epochs using a batch size of 2,048. Due to this large batch size, the number of update steps for individual weights within a single epoch is reduced. This may be problematic if the number of epochs is not increased when the model is optimized by the commonly applied Adam optimizer. For this reason, the model was optimized using the Layerwise Adaptive Rate Scaling (LARS) optimizer[190] with a learning rate of 0.01. This optimizer updates layer-wise instead of the individual weights, reducing the need for more epochs during training. A decay was implemented, where the learning rate is reduced by a factor of 10 if the learning stagnates. The weights of the loss function were set to $\lambda = 5$, $\mu = 5$, and $v = 1$. The complete pipeline was built using the PyTorch module and available on GitHub[1]. For classification, a logistic regression was used, using the learned embeddings as features. Finally, the performance of this classifier is evaluated using the accuracy score.

4.2.7 Baselines

The proposed unsupervised method was compared to a supervised CL method inspired by the SimCLR framework [40] in terms of accuracy. This baseline consists of a ResNet18 encoder followed by a two-layer MLP of dimension 128 as the projector. A smaller baseline architecture is chosen to cope with the limited quantity of labeled data.

The baseline model is trained using the supervised contrastive loss function [94] This loss function is derived from the original NTXent loss [40] which takes the following form:

$$\mathcal{L}^{self} = \sum_{i \in I} \mathcal{L}_i^{self} = -\sum_{i \in I} \log \frac{\exp(\mathbf{Z_i} \cdot \mathbf{Z_j}/\tau)}{\sum_{a \in \mathcal{I} \setminus \{i\}} \exp(\mathbf{Z_i} \cdot \mathbf{Z_a}/\tau)}. \tag{4.10}$$

In this function, let $i \in \mathcal{I} \equiv \{1..2N\}$ be the index of the sample of interest called the anchor sample and j be it's augmented version and defined as the positive sample. All the other $2(N-1)$ samples are defined as the negatives and the temperature scalar parameter is defined as $\tau \in \mathcal{R}^+$.

This original function is rewritten to incorporate label information into the following format:

$$\mathcal{L}_{sup} = \sum_{i \in \mathcal{I}} \frac{-1}{|\mathcal{P}(i)|} \sum_{p \in \mathcal{P}(i)} \log \frac{\exp(\mathbf{Z_i} \cdot \mathbf{Z_p}/\tau)}{\sum_{a \in \mathcal{I} \setminus \{i\}} \exp(\mathbf{Z_i} \cdot \mathbf{Z_a}/\tau)}. \tag{4.11}$$

Here, $\mathcal{P}(i)$ is the set of indices for data samples with the same label as anchor i. The loss is summed over all anchor samples within the batch and normalized by $\mathcal{P}(i)$ which accounts for the number of positive pairs for each anchor. This function aims to place the samples with the same label close together in the embedding space and pushes samples with other labels further away. The supervised baseline model is trained on the labeled Deepship data and is optimized using either the LARS optimizer with a learning rate of 0.01 or the Adam optimizer with a learning rate of 0.0005. The Adam optimizer is also implemented since this optimizer is suggested in previous work that proposed a supervised CL framework for UATR [130, 157, 181]

[1]https://github.com/hildeingvildhummel/UATR

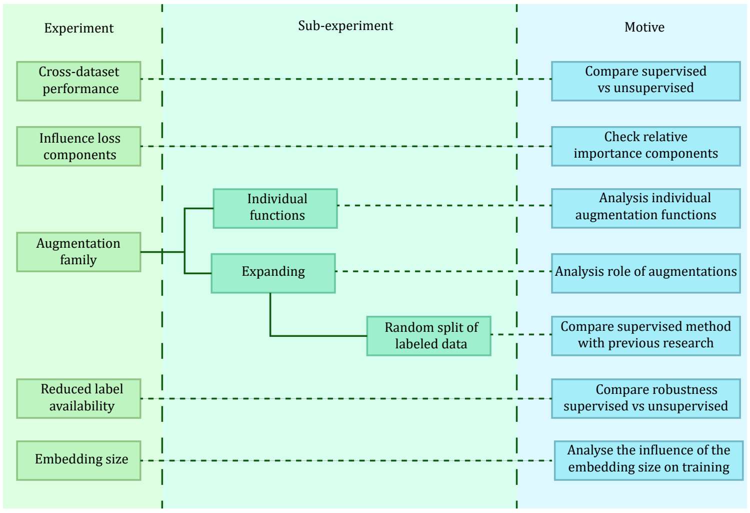

Experiment
Sub-experiment
Motive
Cross-dataset performance
Compare supervised vs unsupervised
Influence loss components
Check relative importance components
Individual functions
Analysis individual augmentation functions
Augmentation family
Expanding
Analysis role of augmentations
Random split of labeled data
Compare supervised method with previous research
Reduced label availability
Compare robustness supervised vs unsupervised
Embedding size
Analyse the influence of the embedding size on training

Figure 4.3.1: Overview of the experiments.

4.3 Results

In this section, the generalization capability of the proposed unsupervised method is evaluated and compared to the supervised CL framework. One way to estimate this is to measure the cross-dataset performance. In addition, an in-depth analysis of the weights of the VICReg loss components, augmentation functions, availability of labeled data, and embedding dimensions is presented. An overview of the experiments is visualized in Figure 4.3.1.

4.3.1 Multi-Purpose classification

The performance of the unsupervised CL method was compared with the supervised baseline models. As summarized in Table 4.3.1, the supervised method optimized with the LARS optimizer has the best performance on Deepship, achieving a 56.51% accuracy score. The proposed unsupervised method competes with the supervised setting, with a 55.61% accuracy score. However, when the models were applied to ShipsEar, the performance of both supervised methods declined. In contrast, the unsupervised method maintained a similar performance level, surpassing the supervised method by more than 30% for the Adam optimizer and nearly 10% for the LARS optimizer. In addition to ship-type classification, the unsupervised model outperforms supervised baselines in marine mammal vocalization classification, indicating enhanced generalization capabilities. In particular, the supervised model optimized by Adam suffers from a larger decline in performance when translated to other datasets compared to the supervised model optimized with LARS. These optimizers differ in strategy, where Adam optimizes each weight individually and LARS optimizes each layer separately. Due to the large batch size during training, the number of update steps per epoch is reduced. Therefore, the supervised method optimized by Adam loses its ability to translate to other related UATR tasks.

Table 4.3.1: Accuracy over multiple datasets.

Dataset	Deepship	ShipsEar	Watkin's
Supervised CL ResNet18 (Adam)	53.94%	21.40%	54.54%
Supervised CL ResNet18 (LARS)	**56.51%**	43.94%	80.46%
Unsupervised CL Conformer	55.61%	**52.14%**	**87.00%**

4.3.2 Influence Individual Loss Components of VICReg

In the original VICReg paper [25], the authors demonstrated that at least the invariance and the variance components are essential to prevent collapse during training. They originally recommended a weight combination of $\lambda = 25$, $\mu = 25$, and $v = 1$, focusing on computer vision tasks. However, images encounter more contrast than spectrograms, making UATR more prone to informational collapse in which the variables of the embeddings carry redundant information [25]. Consequently, the covariance component of the VICReg loss function can be more important for underwater acoustic data than for computer vision-related tasks. For this reason, various weight combinations were tested (see Table 4.3.2), confirming that the relative importance of the covariance component needs to be increased to optimize the embeddings in UATR.

Table 4.3.2: The influence of the augmentation functions on the classification of Ship types.

λ	μ	v	Accuracy
1	1	1	55.35%
5	5	1	**55.61%**
25	25	1	23.81%

4.3.3 Influence of the Augmentation Family

An augmentation family, consisting of various augmentation functions, is needed to define positive samples. During training, a single audio recording is augmented by randomly selecting two individual functions from this family. Both augmented samples will then be optimized to be close together in the expanded space using CL. The performance of a model optimized using CL, may be sensitive to the choice of the augmentation functions. For this reason, the influence of the different augmentation functions within the specified family is analyzed (see Table 4.3.3). The low-pass filter alone achieves the best performance, resulting in a 0.48% accuracy gap on Deepship compared to the model that combines all three augmentation functions. This is likely due to the fact that the most discriminative information in the spectrogram is primarily located in the lower frequency bands. This result indicates that the unsupervised model focuses on the most informative spectral regions, meaning that more augmentation functions related to the low-pass filter in the augmentation family can increase the model's performance.

Expanding the Augmentation Family

The augmentation family was expanded by multiple speech-based augmentation functions, such as varying gain, reverb, pitch, and performing polarity inversion. However, introducing more types of speech-wise augmentations does not improve the model performance on all three datasets (see Table 4.3.4). The performance on Deepship is improved; however, the performance on the other datasets is decreased. This shows the importance of representative augmentation functions in guiding the model in an unsupervised learning framework. In contrast, in the supervised setting, these models benefited from the additional augmentation functions. Here, both models show an increase in performance on all three datasets. The performance on all three datasets is greatly improved, showing that the generalization capacity of the supervised models is increased by expanding the augmentation family. The supervised model trained with the LARS optimizer has a competitive performance similar to the unsupervised learning model with the original augmentation family. Expanding the augmentation family increases the variability of the data during training, since the supervised method can rely on label information, this method can benefit from this additional variability. This is not the case for the unsupervised method, which does not depend on high-quality labeled data.

Table 4.3.3: The influence of the augmentation functions on the classification of Ship types.

Augmentation Function	Deepship	ShipsEar	Watkin's
LowPass	**55.93%**	49.42%	87.00%
Noise	23.45%	20.18%	30.88%
MixUp	22.99%	20.73%	33.92%
Combined	55.61%	**52.14%**	**87.00%**

Table 4.3.4: Full augmentations Supervised vs Unsupervised.

Model	Deepship	ShipsEar	Watkin's
Supervised CL (Adam)	**63.07%**	45.91%	75.73%
Supervised CL (LARS)	61.17%	**52.64%**	**86.33%**
Unsupervised CL	57.78%	50.47%	86.14%

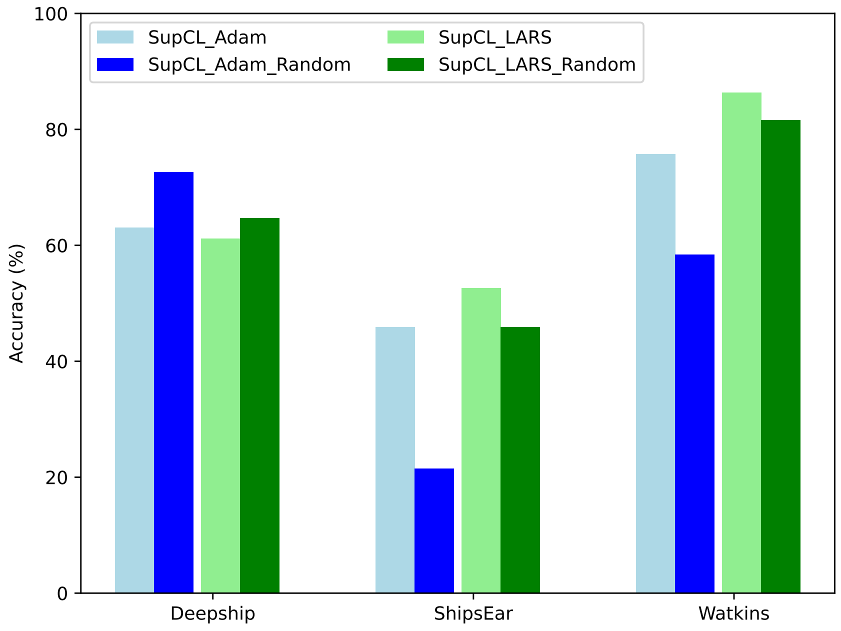


Figure 4.3.2: Accuracy of supervised baseline models trained on either time-wise split or random split of Deepship.

Random Data Split for the Supervised Model

So far, the models are evaluated using a time-wise split for training and test set. However, previous research on automatic ship recognition adopted a random split for evaluation [86]. Unfortunately, such a split provides a poor reflection of the performance of the model in real-world conditions. To examine the impact of the train-test split, a random split was generated by randomly splitting the individual Deepship recordings into a training set (80%) and a test set (20%), ensuring that no individual recordings were present in both sets. The baseline models were trained in a supervised manner using the expanded augmentation family. Next, the performance of the models is examined by translating them again to ShipsEar and Best of Watkin's (see Figure 4.3.2). Here, both baseline models have an increased accuracy score when trained on the Random split, comparable with previous research [86]. This is because the number of newly seen ships is smaller compared to the time-wise split. In addition, the impact of the different seasons is reduced with the random split. However, the generalization performance for both ShipsEar and Best of Watkin's is worse when the model is optimized on the random split. An explanation for this could be that the Deepship data is more stationary compared to ShipsEar and Watkins, which capture more temporal fluctuations (see Figure 4.3.3). This shows that the defined split in the literature results in a higher accuracy on Deepship, but reduces its generalization capacity in other related tasks or regions.

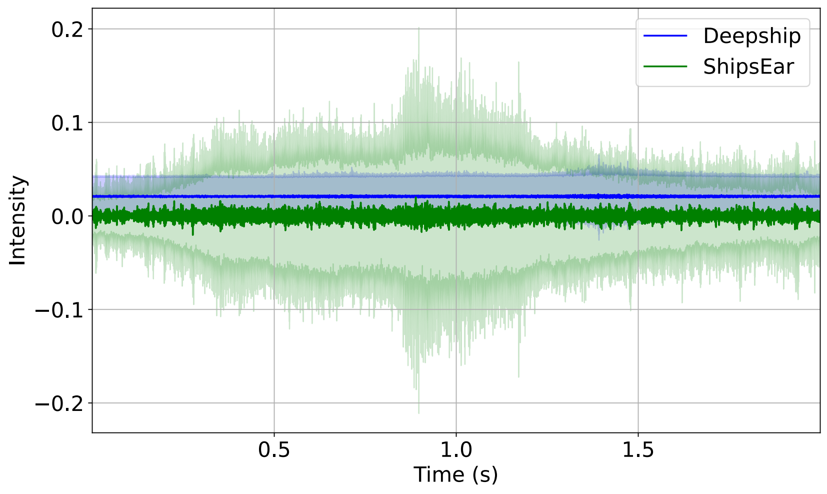


Figure 4.3.3: Temporal Variation of Deepship and ShipsEar, showing the mean and the variability.

4.3.4 Impact of Reduced Labeled Data

The performance of the supervised models is known to rely on a large amount of labeled data. To evaluate the impact of reducing labeled data on model performance, the size of the Deepship training dataset was reduced by randomly selecting a subset of individual recordings for training. The unlabeled data used for unsupervised CL remained the same. This experiment explores the relative impact of a reduced set of available labeled data for both methods. The impact on performance for both the baseline models and the Conformer model is visualized in Figure 4.3.4. Here, the unsupervised method outperforms the supervised methods when the number of labeled data samples is reduced. This observation highlights the robustness of the unsupervised method compared to supervised models. A notable exception occurs at the 10% dataset size, where the baseline model trained with the Adam optimizer exhibits marginally better performance than the unsupervised model. This result is likely due to the redundancy of individual ships in the 10% dataset compared to the 25% dataset, creating a performance peak for this configuration.

4.3.5 Influence of Embedding Size

The size of the embedding vector directly influences the ability of the model to learn the patterns within the dataset. Larger vectors have the ability to store more features; however, they increase the risk of underfitting the data and need more computational resources. However, smaller vectors may not capture essential patterns within the training dataset. This trade-off is examined in the proposed unsupervised framework, with the results illustrated in Figure 4.3.5. Here, the accuracy score increases as the embedding size increases to 512, with only slight

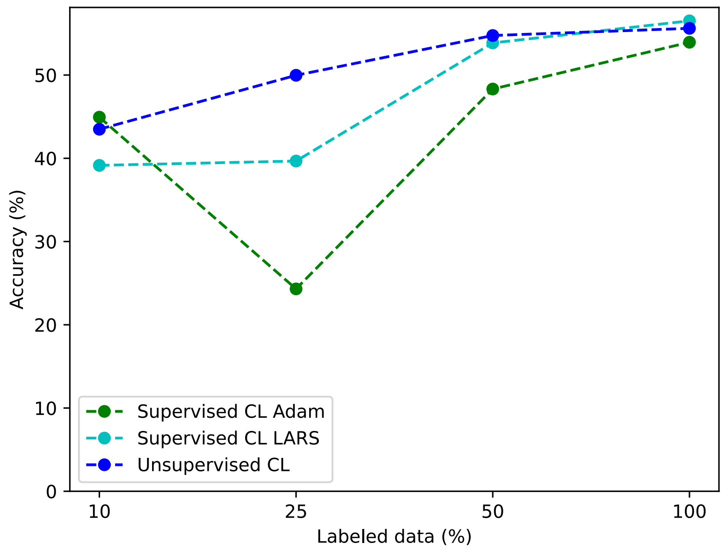


Figure 4.3.4: The accuracy on Deepship when trained on reduced amount of labeled data.

gains beyond this point. Further examination of the loss components during training showed that larger embedding sizes are associated with higher covariance loss and lower variance loss, while smaller sizes exhibit the opposite trend. These results align with the idea that larger vectors ease higher variations among them but also increase the likelihood of correlated elements, making the model prone to informational collapse.

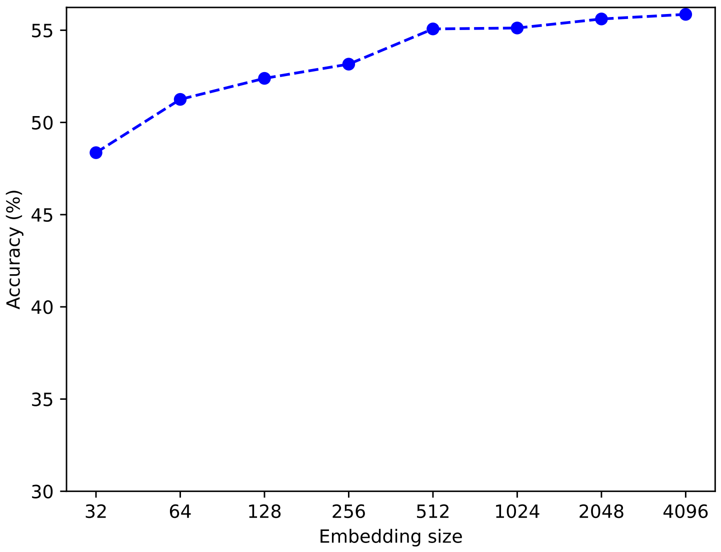


Figure 4.3.5: The accuracy on Deepship over the embedding size.

4.4 Discussion and Conclusion

In this paper, a new proof-of-concept small architecture is proposed on unlabeled data. The model successfully extracted informative features from this unlabeled dataset and was able to transfer this knowledge to other labeled datasets by classifying both the sounds of marine mammals and the types of ships. In particular, the unsupervised model correctly classified newly seen ships into their corresponding class. The proposed unsupervised framework shows performances comparable to the supervised baselines. However, the performance of these supervised models is susceptible to a reduction of the amount of labeled data, while the unsupervised framework that was presented was shown to be more robust to this. Although supervised models achieved superior accuracy on the complete labeled datasets they were trained on, their performance was greatly reduced when implemented to similar tasks. Expanding the augmentation family increased the general capacity of the supervised embeddings; however, this capacity was reduced under a random split. The random split increased the accuracy on the Deepship dataset, but decreased the performance on the other related tasks. This shows the trade-off between the accuracy score in the initial training dataset and the generalization of the learned embedding space to multipurpose underwater acoustic analysis tasks. This trade-off has not been mentioned in previous research, where high accuracy values were reported on both ShipsEar and Deepship.

Note that the generalization performance of the unsupervised framework is dependent on the choice of the augmentation family. This family should consist of task-related augmentation functions. This reliance arises because of the minimal contrast within the spectrograms. This lack of contrast is due to the limited temporal variance within the labeled datasets. Consequently, the unsupervised framework relies on augmentation functions to introduce more contrast.

The defined ship types in the labeled datasets have similar acoustic profiles with a high spread of acoustic signatures within a single class and a great overlap between classes. This makes the categorization of the acoustic profiles into these classes and more specific categorization of these profiles is needed. A solution to this problem could be to create classes based on the characteristics of the ship, such as the number of blades, the size of the ship, and the type of engine(s), instead of the first-level Automatic Identification System (AIS) categorization.

Overall, the proposed method achieves a performance similar to that presented in [162]. However, they reported a larger framework for the model, and they implemented few-shot learning using only Deepship data treating part of the data as unlabeled and not by making the translation between two different datasets. An advantage of this over few-shot learning is that there is a great amount of unlabeled data available. Therefore, this framework can be extended by incorporating more diverse underwater acoustical data from multiple publicly available hydrophones. When more data is included in the training dataset, the small framework that was presented is probably no longer sufficient, and a larger network is needed to capture all the variability and complexity of this diverse dataset. By incorporating more data, this framework could be translated to other underwater acoustic analysis tasks, such as climate change monitoring and nuclear bomb test detection.

This work highlights the potential of unsupervised CL methods in UATR. It explored the implementation of highly available unlabeled lower-quality underwater acoustic data and presented a CL pipeline for robust generalized embedding generation with competitive performance to supervised models. Overall, this shows the potential for the possibilities of unsupervised learning methods in passive acoustic monitoring to automatically monitor sound pollution. This work can be extended for multiple automatic underwater acoustic analysis tasks.

Part II

Deep Learning for the Application Layer

CHAPTER 5

Unveiling the Potential: Harnessing Deep Metric Learning to Circumvent Video Streaming Encryption

Based on [5]:
Arwin Gansekoele, Tycho Bot, Rob van der Mei, Sandjai Bhulai, and Mark Hoogendoorn
Unveiling the Potential: Harnessing Deep Metric Learning to Circumvent Video Streaming Encryption
Published in the 2023 IEEE / WIC International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT) proceedings

Abstract

Encryption on the internet with the shift to HTTPS has been an important step to improve the privacy of internet users. However, there is an increasing body of work about extracting information from encrypted internet traffic without having to decrypt it. Such attacks bypass security guarantees assumed to be given by HTTPS and thus need to be understood. Prior works showed that the variable bitrates of video streams are sufficient to identify which video someone is watching. These works generally have to make trade-offs in aspects such as accuracy, scalability, robustness, etc. These trade-offs complicate the practical use of these attacks. To that end, we propose a deep metric learning framework based on the *triplet loss* method.

Through this framework, we achieve robust, generalizable, scalable and transferable encrypted video stream detection. First, the triplet loss is better able to deal with video streams not seen during training. Second, our approach can accurately classify videos not seen during training. Third, we show that our method scales well to a dataset of over 1000 videos. Finally, we show that a model trained on video streams over Chrome can also classify streams over Firefox. Our results suggest that this side-channel attack is more broadly applicable than originally thought. We provide our code alongside a diverse and up-to-date dataset for future research.

5.1 Introduction

The Internet has become an indispensable part of everyday life in modern society. Being able to interact with family and friends in the blink of an eye, irrespective of their location is among the greatest accomplishments of the past century. A vital component in the pipeline to sharing information over the internet is HTTP.

HTTP was not designed initially with any security features, however. If a third party wanted to see the content of someone's actions on the internet, it would be reasonably simple to do so. Such a third party could even pretend to be one of the communicating parties, a man-in-the-middle (MITM) attack. With that risk in mind, a secure version of HTTP called HTTPS was proposed. This version added encryption and verification to the existing HTTP protocol.

This transition had major implications for automated packet inspection methods. Without encryption, connections could be inspected automatically using connection parameters to identify the traffic, e.g., video, audio, chat etc.[44, 146, 166]. This transition is thus great news for the end user as encryption gives a strong guarantee of privacy. However, it also complicates the work of internet service providers (ISPs), as well as giving cyber criminals more leeway to go undetected. Clearly, there are parties that would benefit from bypassing HTTPS. It is thus important to understand the weaknesses in the HTTPS protocol to be able to patch them. To this end, a new form of side-channel attacks has been researched recently [170]. While it is nearly impossible to decrypt the content of an interaction, the interaction itself still contains information.

One instance of such connections is video streaming. The majority of video streams are transmitted using either the *HTTP live streaming* (HLS) or MPEG-DASH protocol. These enable HTTP(S) servers to deliver video content adaptively based on users' preferences and internet availability. As such, they can identify the network conditions of users on the fly and send segments of appropriate quality. However, multiple works [20, 46, 49, 66, 140, 141, 150, 176, 177] have shown that these protocols leak information such that a third party could identify which video is being watched when matched against a set of known videos. These range from approaches based on network statistics [140, 141, 176], manual feature extraction with traditional ML methods[46, 49] and DNNs[150].

We find the latter two of these approaches to have their own strengths and weaknesses. A traditional ML approach such as the k-nearest neighbors (kNN) algorithm is cheap to extend with new classes but is not as accurate as DNNs. In turn, DNNs are expensive to retrain. Furthermore, no work has yet proposed a method to deal with out-of-distribution (OOD) data. This is data not seen during training but encountered at test time. Finally, no learning-based method is currently capable of transferring across different settings. Browsers can have different implementations of the MPEG-DASH controller, for example, which means that the same video watched on different browsers can result in entirely different streams. To address these issues, we propose a triplet loss approach with an extension we call outlier leveraging (OL). Our approach combines the strengths of both classes of methods while also being better able to deal with OOD data. More specifically, we make the following contributions.

1. **Robust**. We show that triplet loss models are significantly more robust in the presence of OOD video streams than current state-of-the-art models. The addition of OL further improves robustness when OOD streams are dissimilar from the streams trained on.
2. **Generalizable**. We find that the triplet loss model allows the inclusion of new classes without retraining the model. We empirically show that our method achieves this with high accuracy.
3. **Scalable**. We demonstrate that our method scales well when the number of videos to detect is large and the number of streams available is small.
4. **Transferable**. We show that our method transfers well across settings. A model trained on video streams from Chrome is also useful for classifying video streams from Firefox.

In practice, our work suggests the possibility of bypassing encryption on a large scale for video streaming.

5.2 Related Work

Through the challenges defined above, we identify three related areas of research. The main area of research is encrypted streaming content detection, the identification of which videos correspond to which encrypted video streams.

5.2.1 Encrypted Video Stream Detection

Many works have already shown that it is possible to identify the nature of encrypted network traffic [44, 146, 166]. While it was known that it is possible to determine the nature of traffic, Dubin et al. [49] were among the first to show that the connection is sufficiently distinct to determine which video is being watched. Gu et al. [66] created a novel fingerprinting algorithm to identify variable bit-rate video streams. Wu et al. [176] proposed a similar method but aligned the sequences instead of treating them as sets and applying ML. Reed and Klimkowski [140] demonstrated that it is possible to construct fingerprints of Netflix shows statistically purely by looking at the segment sizes and bitrates. In follow-up work, Reed and Kranch [141] showed that an update by Netflix did little to patch this vulnerability. Dias et al. [46] proposed a video streaming classifier based on the Naive Bayes method. Schuster et al. [150] were among the first to use DNNs for this problem, specifically CNNs. Bae et al. [20] demonstrated the effectiveness of this attack in long-term evolution (LTE) networks. Wu et al. [177] showed that it is also possible to identify the resolution of a video stream. We build on this body of work through our deep metric learning approach.

5.2.2 Out-of-Distribution Detection and Open Set Recognition

Our work also touches on the field of out-of-distribution detection (or open-set recognition). This ML field addresses the issues of models face in the presence of data they were not initially trained on. Hendrycks and Gimpel [79] introduced a baseline model with an evaluation suite for out-of-distribution detection. Lee et al. [105] opted to use additional synthetic OOD samples as training input. This method worked well for synthetic out-of-distribution samples, but natural images proved problematic. Hendrycks et al. [80] proposed the outlier exposure method, where they fine-tuned a classifier using a large dataset and showed it improves OOD detection. These works have not been applied to encrypted video streams and form the basis for the new outlier leveraging method we propose.

5.2.3 Deep Metric Learning

A weakness of the kNN approach is that applying a distance metric to compare features is not necessarily informative. Deep metric learning is an approach where neural networks are used to learn a feature transformation such that the transformed features have an informative distance metric. The most common deep metric learning approach is the triplet loss[149]. While originally thought not to be as effective as classification losses, Hermans et al. [81] set guidelines on how to train triplet loss models and showed that they can be more effective than classification losses. Sirinam et al. [153] approached the problem of web fingerprinting using a triplet loss-based network. They achieved promising results in adapting to new websites. Wang et al. [172] expanded on this by including adversarial domain adaptation to better transfer knowledge from a source dataset to a target dataset. We use these works to devise our own triplet loss-based method.

5.3 Methodology

We now explain our method to address the challenges described earlier. Recall that, in our case, we wish to determine which video was watched based on the stream we observed. A stream in this case is a time series of how many bits were received at every point in time. Every time a video is streamed, the bits received and the times at which they are received differ based on factors such as the browser or network conditions. As the stream is encrypted, we assume it is not possible to determine the video based on the content. In fact, it is important to note that the patterns captured in this side-channel attack are not based on the video content but on its MPEG-DASH segmentation. The same video hosted on different platforms would result in dissimilar streams if different parameters are used. The variance between different streams of the same video also makes it difficult to establish exact matches. That is why we use multiple different streams as samples and their corresponding videos as class labels to create a classification model. Our approach uses the triplet loss, which fulfils the same purpose but operates differently.

5.3.1 Triplet Loss

The triplet loss is a metric learning loss approach used to teach a model to predict the similarity between samples. A neural network learns to map samples onto a metric space where samples from the same class are closer together than samples from different classes. In our case, it means that it positions streams from the same video closer together. The triplet loss is computed over a set of triplets $\mathbb{T}$ where each triplet is a combination of three unique streams: an anchor $\boldsymbol{a}$, a positive $\boldsymbol{p}$ and a negative $\boldsymbol{n}$. The anchor is a stream that belongs to the same video as the positive stream, whereas the negative stream is a stream of a different video. Intuitively, the negative stream should be further away from the anchor stream than the positive stream. A margin parameter m is normally included to define how much further away a negative stream should be compared to the positive stream. The goal of the triplet loss is thus to teach a neural network f_θ parameterized by parameters θ to transform some stream $x^{(i)}$ into an embedding $f_\theta(x^{(i)})$ such that any embedding $f_\theta(x^{(j)})$ of the same video lies closer to this embedding than the embeddings of streams of different videos. The 'closeness' can be defined through any differentiable distance metric. We have opted for the Euclidean distance as it is often the most effective [81]. We define the formula for the triplet loss in Equation (5.1).

This *metric learning approach* differs fundamentally from classification approaches [150]. Classification approaches teach a model of the similarity of a stream to a pre-defined video. They are directly optimised to determine which video stream belongs to which videos. Classification approaches are thus often more accurate, as they are optimised for that purpose. A metric learning approach teaches a model which streams are similar but not explicitly which videos they belong to. A major advantage of such an approach is that the resulting model is not dependent on which videos it sees during training. It can classify any arbitrary video, which has made this approach popular in other fields for n-shot learning.

$$\begin{aligned}\mathcal{L}_{triplet}(f_\theta;\mathbb{T};m) = \frac{1}{|\mathbb{T}|}\sum_{(a,p,n)\in\mathbb{T}} \max\{0, m+ \\ ||f_\theta(\boldsymbol{x}^{(a)}) - f_\theta(\boldsymbol{x}^{(p)})||_2 - ||f_\theta(\boldsymbol{x}^{(a)}) - f_\theta(\boldsymbol{x}^{(n)})||_2\}.\end{aligned} \tag{5.1}$$

We perform our triplet selection in an online fashion similar to [81]. This means that we first generate the embeddings from a randomly sampled balanced batch. A balanced batch entails having a subset of all classes and sampling an equal amount of samples for each of those classes. After generating the embeddings, we perform triplet selection and compute the loss within this batch.

To select the triplets, we use two strategies. In the first ten epochs, we use a semi-hard negative strategy. In this strategy, we randomly select for each positive-anchor pair a random negative such that the triplet is semi-hard. A semi-hard triplet is a triplet where the distance of the anchor to the negative is larger than to the positive, but the difference is not larger than the margin parameter yet. This strategy is easy to optimize and helps avoid model collapse. After the tenth epoch, we switch to the hardest negative strategy. In this strategy, a set of triplets is formed by selecting the negative closest to the anchor for each positive-anchor pair in a batch. We found this strategy the most effective during tuning.

5.3.2 Inference

While the triplet loss defines a training procedure, we still need to define how to perform classification with the trained model. There are multiple approaches possible here, but we have opted for an approach similar to [81]. We take the mean embedding for every class, dubbed the centroid, and treat the distance of the centroid to new data points as the class-conditional score. We define the set of embeddings $\mathbb{Z}_k$ belonging to class k given some (training) dataset $\mathbb{X}$ in Equation (5.2).

$$\mathbb{Z}_k = \{f_\theta(\boldsymbol{x}) \mid (\boldsymbol{x}, y) \in \mathbb{X}, y = k\}\,. \tag{5.2}$$

We can then use these embeddings to compute the centroid $\boldsymbol{c}_k$ for class k using Equation (5.3) below.

$$\boldsymbol{c}_k = \frac{1}{|\mathbb{Z}_k|} \sum_{f_\theta(\boldsymbol{x}) \in \mathbb{Z}_k} f_\theta(\boldsymbol{x}). \tag{5.3}$$

Using centroids is an intuitive solution, as the centroid is the point that minimizes the distance to every point in $\mathbb{Z}_k$. Given the definition in Equation (5.3), we define the video-conditional score of some stream $\boldsymbol{x}$ given model f_θ with respect to video k in Equation (5.4) below.

$$s(\boldsymbol{x}; f_\theta, k) = \frac{\exp\left(-||\boldsymbol{c}_k - f_\theta(\boldsymbol{x})||_2\right)}{\sum_{j=1}^{K} \exp\left(-||\boldsymbol{c}_j - f_\theta(\boldsymbol{x})||_2\right)}. \tag{5.4}$$

This corresponds to the softmax over the negative distances of $f_\theta(x)$ to all centroids. We perform prediction by taking the argmax of this score with respect to video k as defined in Equation (5.5) below.

$$\text{prediction}(\boldsymbol{x}; f_\theta) = \text{argmax}_{j \in \{1,\ldots,K\}}\, s(\boldsymbol{x}; f_\theta, j). \tag{5.5}$$

We can also use the score function as a measure of confidence for the model's prediction. A confidence measure is important when considering the OOD component of our problem. By applying a threshold to such a measure, we can determine whether a prediction is correct or not. We have defined this confidence measure in Equation (5.6) below.

$$\text{confidence}(\boldsymbol{x}; f_\theta) = \max_{j \in \{1,\ldots,K\}}\, s(\boldsymbol{x}; f_\theta, j). \tag{5.6}$$

5.3.3 Outlier Leveraging

Like the standard cross-entropy loss, the triplet loss is not generally trained for an open-world setting. A third party may only be interested in a handful of videos and network conditions but needs to handle them regardless. That is why we also introduce a generalisation (OL) of the triplet loss function that allows incorporating streams of OOD videos during training. The loss function $\mathcal{L}_{OL}$ is similar to Equation (5.1) but instead of sampling negatives from other classes, they are sampled from an extra tuning dataset. Intuitively, we now also maximize the distance between the OOD streams and the anchors by using triplet loss. We combine the losses in Equation (5.7).

$$\mathcal{L} = \mathcal{L}_{triplet} + \lambda \mathcal{L}_{OL}. \tag{5.7}$$

The idea behind using two separate loss functions is to keep the sampling process of video streams we wish to distinguish from each other ($\mathcal{L}_{triplet}$) separate from the sampling process of video streams we want to reject ($\mathcal{L}_{OL}$). This ensures that in every training step, the model is guaranteed to concurrently learn to be discriminative and robust.

5.4 Data Collection

To test our method, we first gather data for our experiments. We opted to gather our own data as opposed to using datasets from prior work. We mainly do so as web technology moves quickly. Results from datasets from years ago may no longer represent the current state of affairs. For example, Google has been increasingly pushing the quick udp internet connections (QUIC) standard, so the dataset we gathered using Google Chrome only contains QUIC streams. An added benefit is that it gives us full control of our experimental setup and thus allowed us to construct a large, and diverse video streaming dataset.

For our data collection, we consider one of the simplest scenarios: a third party has access to the line and can observe all incoming and outgoing traffic. This scenario applies to ISPs and government agencies. This scenario is likely generalizable to other scenarios, e.g., when considering a side-channel attack [150] or when performing over-the-air captures [111].

To collect the data, we first gathered the video IDs we would like to observe using two approaches. The first approach consisted of scraping 20 videos from the YouTube trending page like [150]. The second approach entailed sampling videos pseudo-randomly from YouTube. We did so by taking a random string of size 4 and querying 50 videos using the YouTube API.

For each of the collected videos, we gathered samples as follows. We opened a Selenium browser instance that opens the link to the video. We kept the browser open for 60 seconds and recorded the network traffic using tcpdump. All browser instances used AdBlock Plus to handle advertisements. We filtered out the video component by looking at the domain name system (DNS) requests made and then filtering on the IP address that corresponds to the video component. This is possible, as most DNS requests are not yet encrypted. In case DNS requests are not available, picking connections that transmit a lot of data over a large window of time is an effective alternative.

Given a connection, we created our features as follows. We initialized a vector of size 240 where each element refers to a bucket with a timespan of 1/4 second in the total recording of 60 seconds. For every 1/4 second time window, we recorded the number of bytes that were transmitted during that time. After completion, we standardized each sample individually. Standardization can be helpful when dealing with different-quality video streams. We also found it to perform better than normalization. We repeated this procedure for both the incoming and outgoing packets, resulting in the final 2×240 bivariate feature vector. The gathered datasets are as follows.

1. $\mathbf{D_{small}}$: A dataset consisting of 20 different videos with 100 streams per video. This dataset was collected in a manner similar to [150]; we took 20 videos from the trending page and streamed all of them using a Chrome browser. We use this dataset to evaluate whether our method is performant in a setting known to be 'solved.' For every video, we captured 80 streams for training and 20 streams for evaluation.

2. $\mathbf{D_{large}}$ and $\mathbf{D_{firefox}}$: Two larger datasets which comprise of 1087 different videos with 10 streams and 4 streams each. We gathered them using the pseudo-random approach described earlier. We recorded them using a Chrome and Firefox browser instance respectively. We intended to capture as many different videos as we could with a small number of samples per video to evaluate the scalability of our methods. We split the data in a balanced manner with the proportions 8/2 and 3/1 respectively.

3. $\mathbf{D_{tune}}$ and $\mathbf{D_{out}}$: Two datasets comprising 1000 and 8000 different videos with 1 stream each. This dataset is comprised of different videos than in D_{large} and $D_{firefox}$. The former is used for the OL tuning and the latter for the OOD evaluation. The streaming process was done using a Chrome browser.

5.5 Experimental Setup

After defining the methods and data we use for our experiments, we discuss what experiments we have performed to evaluate these as well as the settings we have used to run our experiments.

5.5.1 Architecture

We have opted for a CNN with three blocks and a fully connected hidden layer. Each block consists of two 1D convolutional filters of size 7 followed by a ReLU activation function and batch normalization[88]. At the end of the block, max pooling is used to reduce the size of the feature map for the first two blocks with a global average pooling block used for the last block. The number of channels for the three blocks is 128, 256, and 512 respectively. A one-layer feed-forward neural network (FFNN) with a hidden layer size of 1024 is used as the classification head. A helpful feature of a CNN is its translation invariance. This helps our network with aligning sequences, which means that our method is not limited to classifying only the start of a video.

5.5.2 Hyperparameter Selection

To optimize our network, we used the Adam [96] optimizer with decoupled weight decay regularization [119]. We used this optimizer alongside a learning rate of 3×10^{-4} and a weight decay of 0.01. We found that the models are sufficiently regularised and that tuning does not result in drastically better models. A benefit of using default hyperparameters is that our results should be straightforward to reproduce. We note the hyperparameters for the models used to classify D_{small} and D_{large} in Table 5.5.1.

5.5.3 Benchmark Models

We evaluated our method empirically alongside two benchmark models. The first benchmark is a kNN approach applied to the input features. While similar to the

Table 5.5.1: The hyperparameter selection for the triplet model and benchmark (where relevant) when trained on D_{small} and D_{large}. Where applicable, the hyperparameters are D_{small}/D_{large} separated. Note that the videos per batch and samples per video together determine the batch size of a balanced batch. The architecture for D_{small} is identical to [150] with lower dropout values for the triplet loss model. For D_{large}, we use the architecture described in the previous architecture subsection.

	Selected
Architecture	CNN
Optimiser	AdamW
Number of Epochs	250
Learning Rate	3×10^{-4}
Weight Decay	0.01
Embedding Size	1024
Videos per Batch	5/25
Samples per Video	25/5
Number of Runs	10

approach in [49], we found that their approach is not as effective as their original method when unable to filter out transport layer security (TLS) retransmissions. We found that simply applying kNN to the input features yields surprisingly good results and thus used that approach. The kNN approach provides a lower bound for our method, as it has a trivial training time and is thus able to include new videos on the fly. We select the number of neighbors through 5-fold cross-validation. The optimal number of neighbors for the kNN approaches for all our experiments was 1.

A second benchmark we include is the convolutional neural network approach proposed in [150]. As this is the most accurate approach we are aware of, it is an obvious benchmark. We maintain a similar number of parameters for this benchmark compared to our method to ensure a fair comparison. For the sampling strategy, we do not have to perform triplet mining but instead sample batches of size 128 at random. We refer to the cross-entropy model as CNN in later sections. We can compute the scores and prediction for the benchmark model by taking the max and argmax respectively of the final scores outputted by the CNN.

5.5.4 Experiments

To evaluate our experiments, we use three metrics. First, we use accuracy computed over the videos we wish to classify. This metric tells how well the model can distinguish videos of interest. Second, we compute the mAP (*mean average precision*) between correctly classified videos and videos from D_{out}. We pose this as a binary classification problem where we wish to detect the correctly classified videos. We use the confidence score function in equation 5.6 to determine the ranking. Third, we compute the recall at 100% precision alongside the mAP. Both metrics together give an overview of how well the method performs in the presence of out-of-distribution data. We then performed four experiments to evaluate our four challenges.

1. **Robustness**. In these experiments, we evaluate how robust our models are when evaluated on D_{small} and D_{out}. We train all models and compare them using the metrics as described in this section. We also investigate the effect of OL on performance.
2. **Generalisability**. In these experiments, we evaluate how well our triplet-loss approach performs if we add videos that were not seen during training.
3. **Scalability**. In these experiments, we include the dataset D_{large} to evaluate whether the conclusions we found in experiments 1 and 2 apply in more challenging settings.
4. **Transferability**. In these experiments, we evaluate whether our models trained on Chrome can be used for Firefox data as well.

5.6 Results

Having defined the setup we use to evaluate our methods, we discuss the results we gathered to answer our research question.

5.6.1 Robustness

We thus first evaluated the robustness of our methods compared to the kNN and CNN baselines described earlier. To do so, we evaluated our method on how well it classifies the streams in D_{small}, as well as whether it can distinguish D_{small} from D_{out}. Otherwise, we used the setup as described in the Experimental Setup and report the result of this evaluation in Table 5.6.1.

First, we found that using kNN alone is sufficient to achieve an accuracy above 99%. This indicates, first of all, that kNN is sufficient for a small and clean dataset of encrypted video streams to achieve a high test accuracy. However, an mAP of 5% indicates that the kNN algorithm is unable to cope with new video

Table 5.6.1: The results for the accuracy and robustness metrics for the models trained on D_{small}. The kNN model is the kNN model with a selected number of neighbors of 1. The CNN model is a softmax cross-entropy classifier as in [150]. The triplet and triplet + OL models are our proposed triplet model as well as the outlier leveraging extension proposed. Recall is measured at 100% precision.

	Accuracy	mAP	Recall
kNN	0.99 ± 0	0.05 ± 0	0 ± 0
CNN	1.00 ± 0.00	0.76 ± 0.20	0.19 ± 0.17
Triplet	1.00 ± 0.00	0.98 ± 0.01	0.59 ± 0.32
Triplet + OL	1.00 ± 0.00	1.00 ± 0.00	0.89 ± 0.13

streams; it is not robust. Based on this finding, we opted to experiment with the number of neighbors beyond the results of cross-validation. We found that increasing the number of neighbors can improve the robustness quite drastically. However, it also reduces the accuracy slightly. Consequently, to get an accurate and robust kNN algorithm, it would be necessary to tune using out-of-distribution data.

However, both the CNN and triplet loss models do not require such tuning to achieve much better open-world scores. Both the CNN and triplet loss models respectively represent big steps in terms of open-world detection performance. The CNN model already achieved an mAP of 76%. Intuitively, the mAP is a form of summary statistic to measure the average precision over a set of thresholds based on the recall. Using a neural network seems to benefit the robustness of the model amidst new video streams, especially as it does not require out-of-distribution data for tuning.

It was known that the CNN model is more effective than the kNN approach. Interestingly, the triplet loss model is quite a bit more robust than the CNN. The mAP of 98% indicates that it can retrieve most of the videos we want to classify without making errors. Concretely, the recall at 100% precision metric indicates that it can identify 59% correctly without making any errors. The triplet loss model has access to the same data as the CNN and, unlike the kNN model, has a similar number of parameters as the CNN. The triplet loss function seems to force the backbone to extract more information from the training data. It seems to get penalized much more for being uncertain as opposed to the cross-entropy loss used by the CNN benchmark. Consequently, it is better able to recognize which video streams belong to one of the known videos and which do not. Using the triplet loss function thus results in more robust models in an open-world setting.

5.6.2 Generalisability

We have thus shown that on the small dataset, our method is at least as accurate while being more robust than previously known methods. In theory, it also provides the same flexibility of including new videos on the fly. To test this, we randomly selected and left out 2 of the videos in D_{small}, and trained on the remaining 18 videos. We demonstrate what the impact is of incorporating n shots with varying numbers for n in Figure 5.6.1. Interestingly, adding more shots strictly improves the performance of the model despite not using any retraining steps. When having access to the full 80 shots, the classification of the newly added videos is almost as accurate as the triplet loss model, and more accurate than kNN. This suggests that our model may be able to not only detect OOD data but classify them as well.

5.6.3 Scalability

The dataset D_{small} is fairly small and we cannot say much yet about whether our results would generalize to larger datasets. It can be argued that the dataset is

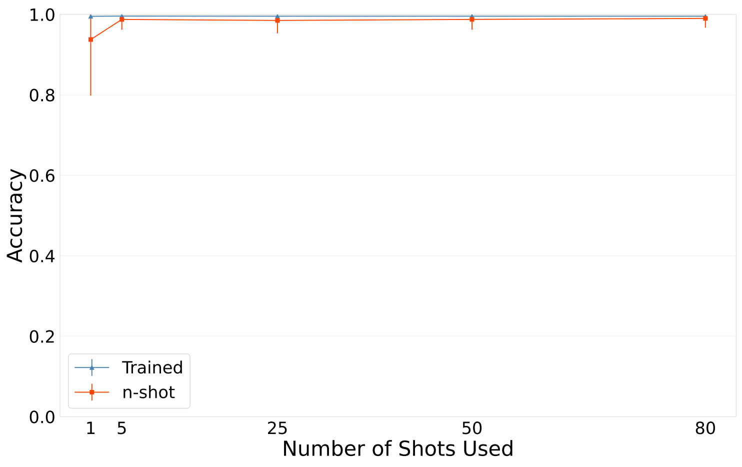


Figure 5.6.1: The accuracy for the models trained using 18 of the videos (leave out a fraction of 0.1) but where we include a variable number of shots. It seems that generating a centroid using more data has a positive effect on the accuracy.

Table 5.6.2: The results for the accuracy and robustness metrics for the models trained on D_{large}. The kNN model is the kNN model with a selected number of neighbors of 1. The CNN model is a softmax cross-entropy classifier as in [150]. The triplet and triplet + OL models are our proposed triplet model as well as the outlier leveraging extension proposed. Recall is measured at 100% precision.

	Accuracy	mAP	Recall
kNN	0.38 ± 0	0.10 ± 0	0 ± 0
CNN	0.86 ± 0.01	0.82 ± 0.01	0.03 ± 0.03
Triplet	0.86 ± 0.00	0.88 ± 0.00	0.08 ± 0.04
Triplet + OL	0.86 ± 0.01	0.87 ± 0.00	0.05 ± 0.02

also fairly simple, given how kNN is sufficient for a test accuracy of over 99%. That is why we have opted to gather the larger dataset D_{large} described earlier. This dataset consists of many different videos but few streams per video, as opposed to a few videos with many streams per video such as in D_{small}. This dataset gives us a better idea of what an attacker would be able to achieve at scale. We used the larger neural network architectures described in the experimental setup for these experiments. We quickly found that the small architectures were insufficient to properly learn the patterns in the data. We first evaluate the robustness again in Table 5.6.2.

We found a few things interesting here. First, a significant gap opens up between the kNN approach and the neural network approaches. The best kNN model is unable to find a good class separation. On the other hand, while the neural network-based methods required more parameters to learn, they all achieved respectable accuracy scores of around 86%. They were also much more robust than the kNN approach. Again, we also found that the triplet loss-based approach is more robust than the CNN. However, adding outlier leveraging to the triplet loss function did not make the model more robust and even made it slightly worse. The collection strategy of D_{large} was the same as D_{tune}, so adding data using outlier leveraging only likely helps when the extra data brings additional information on the out-of-distribution data to expect at inference. Another explanation for the gap might be that OL is primarily useful when the original dataset contains only a small number of distinct videos. Using OL in such a situation provides the model with the diversity it does not have in the original dataset.

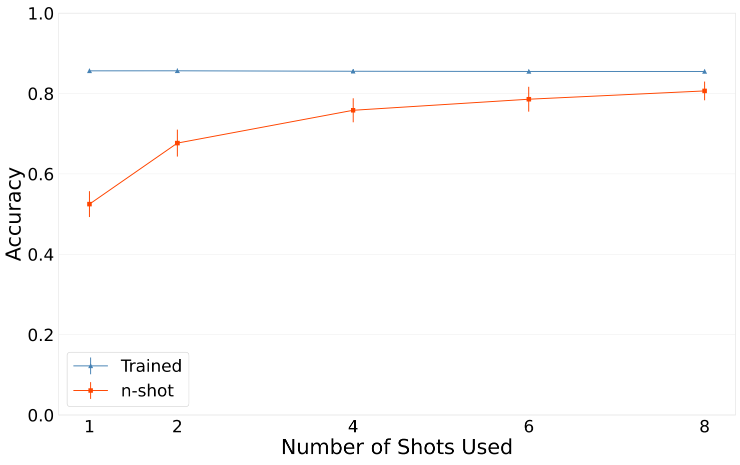


Figure 5.6.2: The accuracy for the models trained using 978 of the videos (leave out a fraction of 0.1) but where we include a variable number of shots. Using more shots increases accuracy quite drastically, achieving over 80% accuracy when adding all 8 available shots. This is despite that we do not touch the model parameters when adding these videos.

Given the disparity in accuracy between kNN and the triplet loss model on D_{large}, we performed the same experiment as in Section 5.6.3. We report the results of this in Figure 5.6.2. Again, we opted to leave out 10% of the videos and vary the number of shots we use. Interestingly, adding more data strictly improved the accuracy of the n-shot videos. If we use 8 shots to construct our centroids, we can achieve as high as 81% accuracy. This is significantly higher than the kNN approach while requiring almost no compute time as we do not retrain the model. Thus, our approach can be an interesting option for quickly including new videos.

An interesting question is how far this would scale in minimal data setups. In Figure 5.6.3, we vary how much data we leave out and use 1 shot for each video added at inference. What is interesting here is that the accuracy is still significantly higher than the kNN approach. When training with a leave-out fraction of 0.5 (a 544/543 training/inference split for video classes), the accuracy is only $2 - 3\%$ lower than when training with a fraction of 0.1 (a 978/109 training/inference

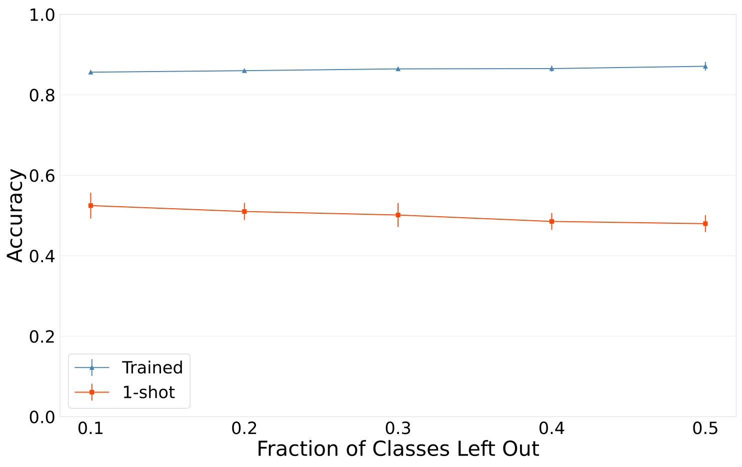


Figure 5.6.3: The accuracy for different models when varying the numbers of videos left out. A fraction of classes left out of 0.2, for example, means that the model is trained on 870 out of the original 1087 videos, and the remaining 217 are added during evaluation. Note that we do not retrain on these videos; we only add centroids for them and evaluate our models.

split for video classes). This implies that our model can potentially incorporate arbitrarily many new videos if we have a sufficiently strong base model. This makes our approach highly scalable.

5.6.4 Transferability

As we have shown that our model can generalize to videos not seen during training, we wondered whether this extends to different settings as well. To do so, we have gathered the dataset $D_{firefox}$. This dataset consists of the same videos as D_{large} but all streams were collected using the Firefox browser, which has a different implementation of the streaming controller. Thus, a model trained on Chrome is likely not transferable to Firefox. As the classification is done using centroids, we could swap the centroids of a model trained on Chrome with the centroids generated over Firefox data.

To evaluate this hypothesis, we first took the original models we trained on D_{large}. For the 0-shot setting, we use the original eight Chrome streams from D_{large} to form the centroids at inference time. We dub this setting 0-shot, as we do not use any streams from the Firefox setting. We use this setting as a baseline to evaluate how well a model trained on Chrome transfers to Firefox. For the 1-shot and 3-shot settings, we took our model trained on Chrome streams but evaluated it using Firefox centroids. The centroids of the 1-shot and 3-shot settings used one or three streams respectively. These settings demonstrate how well a model trained on Chrome streams transfers to classifying Firefox streams. Finally, we trained and evaluated our models on the Firefox data to serve as a benchmark.

Table 5.6.3: The metrics when transferring from the Chrome dataset to the Firefox dataset. The models with (0-shot) perform inference using the Chrome data entirely. The 1-shot and 3-shot settings use 1 and 3 streams from Firefox to construct the centroids. The model is trained using the Chrome streams. The models with the trained label were trained from scratch on the Firefox data. Recall is measured at 100% precision.

	Accuracy	mAP	Recall
CNN (0-shot)	0.07 ± 0.00	0.03 ± 0.01	0.00 ± 0.01
Triplet (0-shot)	0.09 ± 0.01	0.02 ± 0.00	0.00 ± 0.00
Triplet (1-shot)	0.25 ± 0.01	0.17 ± 0.01	0.02 ± 0.01
Triplet (3-shot)	0.29 ± 0.01	0.19 ± 0.01	0.02 ± 0.01
kNN (Trained)	0.12 ± 0	0.02 ± 0	0 ± 0
CNN (Trained)	0.28 ± 0.01	0.13 ± 0.02	0.00 ± 0.01
Triplet (Trained)	0.34 ± 0.01	0.22 ± 0.02	0.03 ± 0.01

We report our results in Table 5.6.3.

As expected, there is a transfer gap between Chrome and Firefox. Hence the setting without any shots performs poorly. What is surprising is the performance of the 1-shot and 3-shot transfer settings for the triplet loss model. They are both more accurate and robust than training a kNN approach entirely on Firefox data. This demonstrates that the model has learned patterns from the Chrome traffic that also generalize well to Firefox streams. Furthermore, while the accuracy of 29% might be too low for a successful attack, it is clear from the model trained on Firefox that this setting is challenging. Nonetheless, this indicates that it is possible to train a model on streams from Chrome and use it for Firefox data as well.

Chrome to Firefox is just one case of conversion, however. Our method is likely transferable to other settings as well. The implication of this is powerful; a sufficiently complex base model might be able to perform this side-channel attack for any arbitrary network setting.

5.7 Discussion and Conclusion

We set out to devise a side-channel attack for encrypted video streams that satisfies four properties. The attack should be robust, generalisable, scalable and transferable. Using a triplet loss-based approach alongside an extension we call outlier leveraging, we were able to address each of these issues. Our attack either beats the state-of-the-art or introduces new possibilities for the attack.

First, the triplet loss approach makes the least mistakes when dealing with out-of-distribution data. Second, it generalises well to new videos without requiring retraining. Third, these conclusions hold when the number of videos becomes large. Finally, a single model is sufficient to classify both Chrome and Firefox streams and is thus likely sufficient for any arbitrary network condition. Interesting future work would be to look further into the transferability our method permits and how to improve on it. Defences against this attack would also be a valuable angle.

The practical implication of our work is that large-scale monitoring of video streaming content over HTTPS is likely possible. Quick solutions to alleviate the vulnerability include adjusting MPEG-DASH encoding settings to make burst patterns less identifiable, maintaining different encodings on different servers, and regularly regenerating encodings for existing videos. In the longer term, we hope that our work helps provide urgency for embedding preventative measures in the HTTPS and/or MPEG-DASH protocols to guarantee the privacy of internet users.

CHAPTER 6

Robustly Overfitting Latents for Flexible Neural Image Compression

Based on [7]:
Yura Perugachi-Diaz, Arwin Gansekoele and Sandjai Bhulai
Robustly overfitting latents for flexible neural image compression
Presented as a poster at the Neural Information Processing Systems 2024 (Neurips) conference

Abstract

Neural image compression has made a great deal of progress. State-of-the-art models are based on variational autoencoders and are outperforming classical models. Neural compression models learn to encode an image into a quantized latent representation that can be efficiently sent to the decoder, which decodes the quantized latent into a reconstructed image. While these models have proven successful in practice, they lead to sub-optimal results due to imperfect optimization and limitations in the encoder and decoder capacity. Recent work shows how to use SGA to refine the latents of pre-trained neural image compression models. We extend this idea by introducing SGA+, which contains three different methods that build upon SGA. We show how our method improves the overall compression performance in terms of the R-D trade-off, compared to its predecessors. Additionally, we show how refinement of the latents with our best-performing method improves the compression performance on both the Tecnick and CLIC dataset. Our method is deployed for a pre-trained hyperprior and for a more flexible model. Further, we give a detailed analysis of our proposed methods and show that they are less sensitive to hyperparameter choices. Finally, we show how each method can be extended to three- instead of two-class rounding.

6.1 Introduction

Image compression allows efficient sending of an image between systems by reducing their size. There are two types of compression: lossless and lossy. Lossless image compression sends images perfectly without losing any quality and can thus be restored in their original format, such as the PNG format. Lossy compression, such as BPG [27], JPEG [171] or JPEG2000 [154], loses some quality of the compressed image. Lossy compression aims to preserve as much of the quality of the reconstructed image as possible, compared to its original format, while allowing a significantly larger reduction in required storage.

Traditional methods [154, 171], especially lossless methods, can lead to limited compression ratios or degradation in quality. With the rise of deep learning, neural image compression is becoming a popular method [159, 163]. In contrast with traditional methods, neural image compression methods have been shown to achieve higher compression ratios and less degradation in image quality [24, 104, 124]. Additionally, neural compression techniques have shown improvements compared to traditional codecs for other data domains, such as video. [10, 71, 121].

In practice, neural lossy compression methods have proven to be successful and achieve state-of-the-art performance [24, 41, 104, 124]. These models are frequently based on variational autoencoders (VAEs) with an encoder-decoder structure [97]. The models are trained to minimize the expected rate-distortion (R-D) cost: $R + \lambda D$. Intuitively, one learns a mapping that encodes an image into a compressible latent representation. The latent representation is sent to a decoder and is decoded into a reconstructed image. The aim is to train the compression model in such way that it finds a latent representation that represents the best trade-off between the length of the bitstream for an image and the quality of the reconstructed image. Even though these models have proven to be successful in practice, they do have limited capacity when it comes to optimization and generalization. For example, the encoder's capacity is limited which makes the latent representation sub-optimal [43]. Recent work [35, 68, 187] proposes procedures to refine the encoder or latents, which lead to better compression performance. Furthermore, in neural video compression, other work focuses on adapting the encoder [19, 120] or finetuning a full compression model after training to improve the video compression performance [167].

The advantage of refining latents [35, 187] is that improved compression results per image are achieved while the model does not need to be modified. Instead, the latent representations for each individual image undergo a refining procedure. This results in a latent representation that obtains an improved bitstream and image quality over its original state from the pre-trained model. As mentioned in [187], the refining procedure for stochastic rounding with SGA considerably improves performance.

In this paper, we introduce SGA+, an extension of SGA that further improves compression performance and is less sensitive to hyperparameter choices. The main contributions are: *(i)* showing how changing the probability space with more natural methods instead of SGA boosts the compression performance, *(ii)* proposing the SSL, which can smoothly interpolate between the approximate atanh, linear, cosine and round, *(iii)* demonstrating a generalization to rounding to three classes, that contains the two classes as a special case, and *(iv)* showing that SGA+ not only outperforms SGA on a similar pre-trained mean-scale hyperprior model as in [187], but also achieves an even better performance for the pre-trained models of [41]. Further, we show how SSL outperforms baselines in an R-D plot on the Kodak dataset, in terms of PSNR vs BPP and in terms of true loss curves. Additionally, we show how our method generalizes to the Tecnick and CLIC dataset, followed by qualitative results. We analyze the stability of all functions and show the effect of interpolation between different methods with SSL. Lastly, we analyze a refining procedure at compression time that allows moving along the R-D curve when refining the latents with another λ than a pre-trained model is trained on [61, 184]. [1]

6.2 Preliminaries and Related Work

In lossy compression, the aim is to find a mapping of image x where the distortion of the reconstructed image $\hat{x}$ is as little as possible compared to the original one while using as little storage as possible. Therefore, training a lossy neural image compression model presents a trade-off between minimizing the length of the bitstream for an image and minimizing the distortion of the reconstructed image [23, 104, 124, 159].

[1]The code can be retrieved from: `https://github.com/yperugachidiaz/flexible_neural_image_compression`.

Neural image compression models from [23, 41, 124, 159], also known as hyperpriors, accomplish this kind of mapping with latent variables. An image x is encoded onto a latent representation $y = g_a(x)$, where $g_a(\cdot)$ is the encoder. Next, y is quantized $Q(y) = \hat{y}$ into a discrete variable that is sent losslessly to the decoder. The reconstructed image is given by: $\hat{x} = g_s(\hat{y})$, where $g_s(\cdot)$ represents the decoder. The rate-distortion objective that needs to be minimized for this specific problem is given by:

$$\begin{aligned}\mathcal{L} &= R + \lambda D \\ &= \underbrace{\mathbb{E}_{x\sim p_x}\left[-\log_2 p_{\hat{y}}(\hat{y})\right]}_{\text{rate}} + \lambda \underbrace{\mathbb{E}_{x\sim p_x}\left[d(x,\hat{x})\right]}_{\text{distortion}},\end{aligned} \tag{6.1}$$

where λ is a Lagrange multiplier determining the rate-distortion trade-off, R is the expected bitstream length to encode $\hat{y}$ and D is the metric to measure the distortion of the reconstructed image $\hat{x}$ compared to the original one x. Specifically for the rate, p_x is the (unknown) image distribution and $p_{\hat{y}}$ represents the entropy model that is learned over the data distribution p_x. A frequently used distortion measure for $d(x, \hat{x})$, is the mean squared error (MSE) or PSNR.

In practice, the latent variable y often consists of multiple levels in neural compression. Namely, a smaller one named z, which is modeled with a relatively simple distribution $p(z)$, and a larger variable, which is modeled by a distribution for which the parameters are predicted with a neural network using z, the distribution $p(y|z)$. We typically combine these two variables into a single symbol y for brevity. Furthermore, a frequent method of quantizing $Q(\cdot)$ used to train hyperpriors consists of adding uniform noise to the latent variable.

6.2.1 Latent optimization

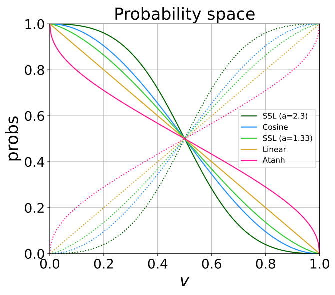
Neural image compression models have been trained over a huge set of images to find an optimal encoding. Yet, due to difficulties in optimization or due to constraints on the model capacity, model performance is sub-optimal. To overcome these issues, another type of optimizing compression performance is proposed in [35, 187] where they show how to find better compression results by utilizing pre-trained networks and keeping the encoder and decoder fixed but only adapting the latents. In these methods, a latent variable y is iteratively adapted using differentiable operations at test time. The aim is to find a more optimal discrete latent representation $\hat{y}$. Therefore, the following minimization problem needs to be solved for an image x:

$$\arg\min_{\hat{y}} \left[-\log_2 p_{\hat{y}}(\hat{y}) + \lambda d(x, \hat{x})\right]. \tag{6.2}$$

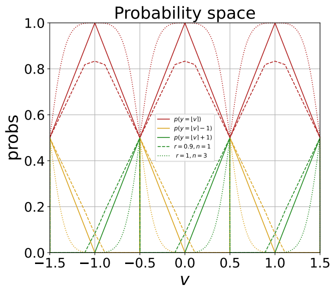
This is a powerful method that can fit to a test image x directly without the need to further train an entire compression model.

6.2.2 Stochastic Gumbel Annealing

Campos et al. [35] proposes to optimize the latents by iteratively adding uniform noise and updating its latents. While this method proves to be effective, there is



(a) Probability space for several SGA+ methods, along with atanh. Solid lines denote the probability of flooring $\lfloor v \rfloor$ and dotted lines the probability of ceiling $\lceil v \rceil$.



(b) Three-class rounding for the extended version of linear. Solid lines denote two-class rounding, dashed lines denote three-class rounding and dotted lines indicate the smoothness.

Figure 6.2.1: Probability space for (a) Two-class rounding (b) Three-class rounding

still a difference between the true rate-distortion loss ($\hat{\mathcal{L}}$) for the method and its discrete representation $\hat{y}$. This difference is also known as the discretization gap. Therefore, Yang et al. [187] propose the SGA method to optimize latents and show how it obtains a smaller discretization gap. SGA is a soft-to-hard quantization method that quantizes a continuous variable v into the discrete representation for which gradients can be computed. A variable v is quantized as follows. First, a vector $\mathbf{v}_r = (\lfloor v \rfloor, \lceil v \rceil)$ is created that stacks the floor and ceil of the variable, also indicating the rounding direction. Next, the variable v is centered between $(0, 1)$ where for the flooring: $v_L = v - \lfloor v \rfloor$ and ceiling: $v_R = \lceil v \rceil - v$. With a temperature rate $\tau \in (0, 1)$, that is decreasing over time, this variable determines the soft-to-hardness where 1 indicates training with a fully continuous variable v and 0 indicates training while fully rounding variable v. To obtain unnormalized log probabilities (logits), the inverse hyperbolic tangent (atanh) function is used as follows:

$$logits = (-\operatorname{atanh}(v_L)/\tau, -\operatorname{atanh}(v_R)/\tau). \tag{6.3}$$

To obtain probabilities a softmax is used over the *logits*, which gives the probability $p(y)$ which is the chance of v being floored: $p(y = \lfloor v \rfloor)$, or ceiled: $p(y = \lceil v \rceil)$. This is approximated by the Gumbel-softmax distribution. Then, samples are drawn: $\mathbf{y} \sim \text{Gumbel-Softmax}(logits, \tau)$ [90] and are multiplied and summed with the vector $\mathbf{v}_r$ to obtain the quantized representation: $\hat{v} = \sum_i (v_{r,i} * y_i)$. As SGA aids the discretization gap, this method may not have optimal performance and may not be as robust to changes in its temperature rate τ.

Besides SGA, [187] propose deterministic annealing [11], which follows almost the same procedure as SGA, but instead of sampling stochastically from the Gumbel Softmax, this method uses a deterministic approach by computing probabilities with the Softmax from the *logits*. In practice, this method has been shown to suffer from unstable optimization behavior.

6.2.3 Other methods

While methods such as SGA aim to optimize the latent variables for neural image compression at inference time, other approaches have been explored in recent research. [70] proposed a soft-then-hard strategy alongside a learned scaling factor for the uniform noise to achieve better compression and a smoother latent. These methods are used to fine-tune network parameters but not the latents directly. [199] proposed using Swin-transformer-based coding instead of ConvNet-based coding. They showed that these transforms can achieve better compression with fewer parameters and shorter decoding times. [167] proposed to also fine-tune the decoder alongside the latent for video compression. While accommodating the additional cost of saving the model update, they demonstrated a gain of $\sim 1dB$. [194] and [50] proposed using implicit neural representations for video and image compression, respectively. [75] proposed an improved context model (SCCTX) and a change to the main transform (ELIC) that achieve strong compression results together. [52] revisited vector quantization for neural image compression and demonstrated it performs on par with hyperprior-based methods. [106] proposed a method to incorporate trellis-coded quantization in neural codecs. While these approaches change the training process, our work differs in that we only consider the inference process. [21] proposes latent shift, a method that can further optimize latents using the correlation between the gradient of the reconstruction error and the gradient of the entropy.

6.3 Methods

As literature has shown, refining the latents of pre-trained compression models with SGA leads to improved compression performance [187]. In this section, we extend SGA by introducing SGA+ containing three other methods for the computation of the unnormalized log probabilities (*logits*) to overcome issues from its predecessor. We show how these methods behave in probability space. Furthermore, we show how the methods can be extended to three-class rounding.

6.3.1 Two-class rounding

Recall from SGA that a variable v is quantized to indicate the rounding direction to two classes and is centered between (0,1). Computation of the unnormalized log probabilities is obtained with atanh from Equation (6.3). Recall, that in general the probabilities are given by a softmax over the *logits* with a function of choice. As an example, for SGA the logits are computed with atanh. The corresponding probabilities for rounding down is then equal to:

$$\frac{e^{\operatorname{atanh}(v_L)}}{e^{\operatorname{atanh}(v_L)} + e^{\operatorname{atanh}(v_R)}}.$$

Then looking at the probability space from this function, see Figure 6.2.1a, the atanh function can lead to sub-optimal performance when used to determine rounding probabilities. The problem is that gradients tend to infinity when the function approaches the limits of 0 and 1, see Appendix 6.A for the proof that gradients at 0 tend to ∞. This is not ideal, as these limits are usually achieved when the discretization gap is minimal. In addition, the gradients may become larger towards the end of optimization. Further analyzing the probability space, we find that there are a lot of possibilities in choosing probabilities for rounding to two classes. However, there are some constraints: the probabilities need to be monotonic functions, and the probabilities for rounding down (flooring) and up (ceiling) need to sum up to one. Therefore, we introduce SGA+ and propose three methods that satisfy the above constraints and can be used to overcome the sub-optimality that the atanh function suffers from. We opted for these three as they each have their own interesting characteristics. However, there are many other functions that are also valid and would behave similarly to these three.

We will denote the probability that v is rounded down by:

$$p(y = \lfloor v \rfloor), \tag{6.4}$$

where y represents the random variable whose outcome can be either rounded down or up. The probability that v is rounded up is conversely: $p(y = \lceil v \rceil) = 1 - p(y = \lfloor v \rfloor)$.

Linear probabilities To prevent gradient saturation or vanishing gradients completely, the most natural case would be to model a probability that linearly increases or decreases and has a gradient of one everywhere. Therefore, we define the linear:

$$p(y = \lfloor v \rfloor) = 1 - (v - \lfloor v \rfloor). \tag{6.5}$$

It is easy to see that: $p(y = \lceil v \rceil) = v - \lfloor v \rfloor$. In Figure 6.2.1a, the linear probability is shown.

Cosine probabilities As can be seen in Figure 6.2.1a, the atanh tends to have gradients that go to infinity for v close to the corners. Subsequently, a method that has low gradients in that area is by modeling the cosine probability as follows:

$$p(y = \lfloor v \rfloor) = \cos^2\left(\frac{(v - \lfloor v \rfloor)\pi}{2}\right). \tag{6.6}$$

This method aids the compression performance compared to the atanh since there is less probability of overshooting the rounding value.

Sigmoid scaled logit There are a lot of possibilities in choosing probabilities for two-class rounding. We introduced two probabilities that overcome sub-optimality issues from atanh: the linear probability from Equation (6.5), which has equal gradients everywhere, and cosine from Equation (6.6), that has little gradients at the corners. Besides these two functions, the optimal probability might follow a different function from the ones already mentioned. Therefore, we introduce the sigmoid scaled logit (SSL), which can interpolate between different probabilities with its hyperparameter a and is defined as follows:

$$p(y = \lfloor v \rfloor) = \sigma(-a\sigma^{-1}(v - \lfloor v \rfloor)), \tag{6.7}$$

where a is the factor determining the shape of the function. SSL is exactly the linear for $a = 1$. For $a = 1.6$ and $a = 0.65$ SSL roughly resembles the cosine and atanh. For $a \to \infty$ the function tends to shape to (reversed) rounding.

Note that the main reason behind the linear version is the fact that it is the only function with constant gradients which is also the most robust choice, the cosine version is approximately mirrored across the diagonal of the $-\operatorname{atanh}(x)$ which shows that it is more stable compared to the $-\operatorname{atanh}(x)$, and the reason behind the SSL is that it is a function that can interpolate between all possible functions and can be tuned to find the best possible performance when necessary.

6.3.2 Three-class rounding

As described in the previous section, the values for v can either be floored or ceiled. However, there are cases where it may help to round to an integer further away. Therefore, we introduce three-class rounding and show three extensions that build on top of the linear probability Equation (6.5), cosine probability Equation (6.6), and SSL from Equation (6.7).

The probability that v is rounded is denoted by: $p(y = \lfloor v \rceil) \propto f_{3c}(w|r, n)$, where $w = v - \lfloor v \rceil$ is centered around zero. Further, we define the probability that v is rounded $+1$ and rounded -1 is respectively given by: $p(y = \lfloor v \rceil - 1) \propto f_{3c}(w - 1|r, n)$ and $p(y = \lfloor v \rceil + 1) \propto f_{3c}(w + 1|r, n)$. Recall, that $v_L, v_R \in [0, 1]$, whereas $w \in [-0.5, 0.5]$. Defining w like this is more helpful for the 3-class since it has a center class. The general notation for the three-class functions is given by:

$$f_{3c}(w|r, n) = f(\operatorname{clip}(w \cdot r))^n, \tag{6.8}$$

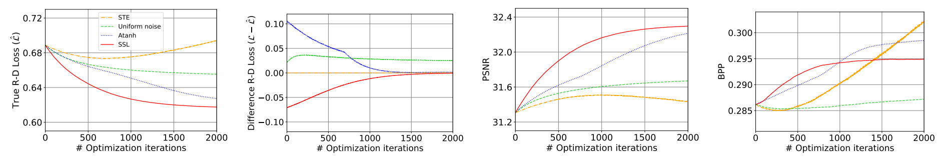
where $\operatorname{clip}(\cdot)$ clips the value at 0 and 1, r is the factor determining the height and steepness of the function and power n controls the peakedness of the function. Note that n can be fused with temperature τ together, to scale the function. This only accounts for the computation of the logits and not to modify the Gumbel temperature, therefore, τ still needs a separate definition.

Extended linear Recall that the linear probability can now be extended to three-class rounding as follows:

$$f_{linear}(w) = |w|. \tag{6.9}$$

A special case is $f_{3c,linear}(w|r = 1, n = 1)$, where the function is equivalent to the linear of the two-class rounding from Equation (6.5). For $r < 1$ this function rounds to three classes, and for $n \neq 1$ this function is not linear anymore.

In Figure 6.2.1b, three-class rounding for the extension of Equation (6.5) can be found. As can be seen, solid lines denote the special case of two-class rounding



(a) True R-D loss **(b)** Difference loss **(c)** PSNR **(d)** BPP

Figure 6.3.1: Performance plots of (a) True R-D Loss (b) Difference in loss (c) PSNR (d) BPP.

with $r = 1$ and $n = 1$, dashed lines denote three-class rounding with $r = 0.9$ and $n = 1$ and dotted lines denote the two-class rounding with $r = 1$ and $n = 3$, which shows a less peaked function. For an example of two- versus three-class rounding, consider the case where we have variable $v = -0.95$. For two-class rounding there is only the chance of rounding to -1 with $p(y = \lfloor v \rceil)$ (red solid line), a chance to round to 0 with $p(y = \lfloor v \rceil + 1)$ (green solid line) and zero chance to round to -2 with $p(y = \lfloor v \rceil - 1)$ (yellow solid line). For three-class rounding, with $r = 0.9$ and $n = 1$, when $v = -0.95$ we find a high chance to round to -1 with $p(y = \lfloor v \rceil)$ (red dashed line) and a small chance to round to 0 with $p(y = \lfloor v \rceil + 1)$ (green dashed line) and a tiny chance to round to -2 with $p(y = \lfloor v \rceil - 1)$ (yellow dashed line).

Extended cosine Similarly, we can transform the cosine probability from Equation (6.6) to three-class rounding:

$$f_{cosine}(w) = \cos\left(\frac{|w|\pi}{2}\right). \tag{6.10}$$

When $f_{3c,cosine}(w|r = 1, n = 2)$, this function exactly resembles the cosine for two-class rounding, and for $r < 1$ this function rounds to three classes.

Extended SSL Additionally, SSL from Equation (6.7) can be transformed to three-class rounding as follows:

$$f_{SSL}(w) = \sigma\left(-a\sigma^{-1}\left(|w|\right)\right), \tag{6.11}$$

where a is the factor determining the shape of the function. When $f_{3c,SSL}(w|r = 1, n = 1)$, this function exactly resembles the two-class rounding case, and for $r < 1$, the function rounds to three classes. Recall that this function is capable of exactly resembling the linear function and approximates the cosine from two-class rounding for $a = 1$ and $a = 1.6$, respectively.

6.4 Experiments

In this section, we show the performance of our best-performing method in an R-D plot and compare it to the baselines. Further, we evaluate and compare the methods with the true R-D loss performance ($\hat{\mathcal{L}}$), the difference between the method loss and true loss ($\mathcal{L} - \hat{\mathcal{L}}$), and corresponding PSNR and BPP plot that respectively expresses the image quality, and cost over t training steps. Finally, we show how our best-performing method performs on the Tecnick and CLIC dataset and show qualitative results.

Following [187], we run all experiments with temperature schedule $\tau(t) = \min(\exp\{-ct\}, \tau_{max})$, where c is the temperature rate determining how fast temperature τ is decreasing over time, t is the number of train steps for the refinement of the latents and $\tau_{max} \in (0, 1)$ determines how soft the latents start the refining procedure. Additionally, we refine the latents for $t = 2000$ train iterations, unless specified otherwise. See Section 6.4.2 for the hyperparameter settings.

Implementation details We use two pre-trained hyperprior models to test SGA+, for both models we use the package from CompressAI [26]. The first model is similar to the one trained in [187]. Note, that the refining procedure needs the same storage and memory as theirs. In Appendix 6.C, implementation details and results of this model can be found. All experiments in this section are run with a more recent hyperprior-based model which is based on the architecture of [41]. Additionally, this model reduces the R-D loss for atanh on average by 7.6% and for SSL by 8.6%, compared to the other model. The model weights can be retrieved from CompressAI [26]. The models were trained with $\lambda = \{0.0016, 0.0032, 0.0075, 0.015, 0.03, 0.045\}$. The channel size is set to $N = 128$ for the models with $\lambda = \{0.0016, 0.0032, 0.0075\}$, refinement of the latents on [98] with these models take approximately 21 minutes. For the remaining λ's channel size is set to $N = 192$ and the refining procedure takes approximately 35 minutes. We perform our experiments on a single NVIDIA A100 GPU.

Baseline methods We compare our methods against the methods that already exist in the literature. The **Straight-Through Estimator** (STE) is a method to round up or down to the nearest integer with rounding bound set to a half. This rounding is noted as $\lfloor \cdot \rceil$. The derivative of STE for the backward pass is equal to 1 [28, 168, 188]. The **Uniform Noise** quantization method adds uniform noise from $u \sim U(-\frac{1}{2}, \frac{1}{2})$ to latent variable y. Thus: $\hat{y} = y + u$. In this manner $\hat{y}$ becomes differentiable [23]. As discussed in Section 6.2.2, we compare against **Stochastic Gumbel Annealing**, which is a soft-to-hard quantization method that quantizes a continuous variable v into a discrete representation for which gradients can be computed.

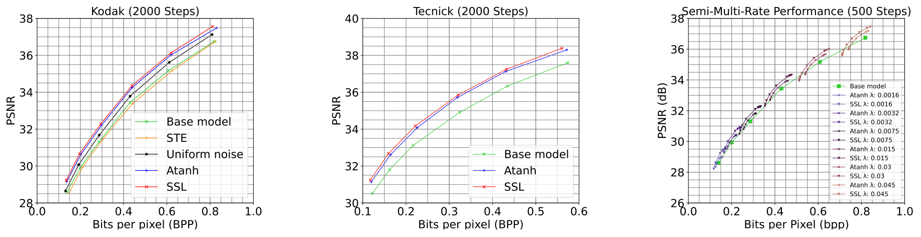
6.4.1 Overall performance

Figure 6.4.1a shows the rate-distortion curve, using image quality metric PSNR versus BPP, of the base model and for refinement of the latents on the Kodak dataset with method: STE, uniform noise, atanh and SSL. We clearly see how SSL outperforms all other methods. In Appendix 6.B.1a, the results after $t = 500$ iterations are shown, where the performance is more pronounced.

To show how each of the methods behave, we take a closer look at the performance plots shown in Figure 6.3.1. The refinement results are obtained from the pre-trained model, trained with $\lambda = 0.0075$. The true R-D loss in Figure 6.3.1a shows that STE performs worse and has trouble converging, which is reflected in the R-D curve. Uniform noise quickly converges compared to atanh and SSL. We find that SSL outperforms all other methods, including atanh in terms of the lowest true R-D loss at all steps. Looking at the difference between the method loss and true loss Figure 6.3.1b, we find that both SSL and atanh converge to 0. Yet, the initial loss difference is smaller and smoother for SSL, compared to atanh. Additionally, uniform noise shows a big difference between the method and true loss, indicating that adding uniform noise overestimates its method loss compared to the true loss.

Tecnick and CLIC To test how our method performs on other datasets, we use the Tecnick [18] and [203] dataset. We run baselines atanh and the base model and compare against SSL with $a = 2.3$. Figure 6.4.1b shows the corresponding R-D curves on the Tecnick dataset. Note that Appendix 6.B.3 contains the results of the CLIC dataset which shows similar behavior as on Tecnick. We find that both refining methods improve the compression performance in terms of the R-D trade-off. Additionally, our proposed method outperforms atanh and shows how it improves performance significantly at all points. The R-D plot after $t = 500$ iterations for both datasets can be found in Appendix 6.B.1. Note, that these results show even more pronounced performance difference.

BD-Rate and BD-PSNR To evaluate the overall improvements of our method, we computed the BD-Rate and BD-PSNR [29] for Kodak, Tecnick and CLIC in Table 6.4.1. We observe that across the board SSL achieves an improvement in both BD-PSNR and BD-Rate. After 500 steps, SSL achieves almost double the BD-Rate reduction as atanh. The difference between the two becomes smaller at 2000 steps. This underlines the faster convergence behavior of the SSL method.



(a) Kodak **(b)** Tecnick **(c)** Semi-multi-rate

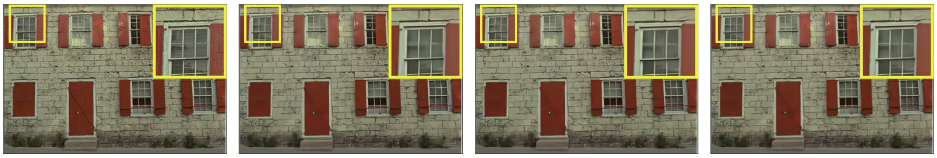
Figure 6.4.1: R-D performance for SSL on (a) Kodak with the baselines, (b) Tecnick with the base model and atanh and (c) Kodak for semi-multi-rate behavior with atanh. Best viewed electronically.

Table 6.4.1: Pairwise Comparison between atanh and SSL of BD-PSNR and BD-Rate.

	BD-PSNR (dB)						BD-Rate (%)					
	500 steps			2000 steps			500 steps			2000 steps		
	Kodak	Tecnick	CLIC	Kodak	Tecnick	CLIC	Kodak	Tecnick	CLIC	Kodak	Tecnick	CLIC
Base vs SSL	0.50	0.57	0.56	0.82	0.95	0.89	-10.30	-11.60	-13.18	-16.23	-18.77	-20.11
Base vs Atanh	0.26	0.28	0.31	0.69	0.79	0.78	-5.52	-5.91	-7.37	-13.82	-15.93	-17.70
Atanh vs SSL	0.24	0.28	0.26	0.14	0.16	0.11	-5.04	-5.97	-6.20	-2.86	-3.34	-2.76

6.4.2 Qualitative results

In Figure 6.4.2, we demonstrate the visual effect of our approach. We compare the original image with the compressed image from base model with $\lambda = 0.0016$, refinement method atanh, and SSL. For the image compressed by SSL, we observe that there are less artifacts visible in the overall image. For instance, looking at the window we see more texture compared to the base model and atanh method.



(a) Original Image

(b) Base Model (0.19 BPP vs 25.75 PSNR)

(c) atanh (0.19 BPP vs 25.91 PSNR)

(d) SSL (0.19 BPP vs 25.98 PSNR)

Figure 6.4.2: Qualitative comparison of a Kodak image from pre-trained model trained with $\lambda = 0.0016$. Best viewed electronically.

Hyperparameter settings Refinement of the latents with pre-trained models, similar to the one trained in [187], use the same optimal learning rate of 0.005 for each method. Refinement of the latents with the models of [41] use a 10 times lower learning rate of 0.0005. Following [187], we use the settings for atanh with temperature rate $\tau_{max} = 0.5$, and for STE we use the smaller learning rate of 0.0001, yet STE still has trouble converging. Note that, we tuned STE just as the other baselines. However, the STE method is the only method that has a lot of trouble converging. Even with smaller learning rates, the method performed poorly. The instability of training is not only observed by us, but is also observed in [187, 188]. For SGA+, we use optimal convergence settings, which are a fixed learning rate of 0.0005, and $\tau_{max} = 1$. Experimentally, we find approximately best performance for SLL with $a = 2.3$.

6.5 Analysis and Societal Impact

In this section we perform additional experiments to get a better understanding of SGA+. An in-depth analysis shows the stability of each proposed method, followed

Table 6.5.1: True R-D loss for different τ_{max} settings of: atanh(v), linear, cosine and SSL with $a = 2.3$. The lowest R-D loss per column is marked with: $\downarrow$. Note that the function containing atanh is unnormalized.

Function $\backslash\tau_{max}$	**0.2**	**0.4**	**0.6**	**0.8**	**1.0**
$\exp\{\text{atanh}(v)\}$	0.6301	0.6273	0.6267	0.6260	0.6259
$1 - v$ (linear)	0.6291 $\downarrow$	0.6229 $\downarrow$	0.6225	0.6222	0.6220
$\cos^2(\frac{v\pi}{2})$	0.6307	0.6233	0.6194 $\downarrow$	0.6186	0.6187
$\sigma(-a\sigma^{-1}(v))$	0.6341	0.6233	0.6196	0.6181 $\downarrow$	**0.6175** $\downarrow$
$\exp\{\text{atanh}(v)\}$	0.0010	0.0044	0.0073	0.0079	0.0084
$1 - v$ (linear)	0	0	0.0031	0.0041	0.0045

Table 6.5.2: True R-D loss of two- versus three-class rounding for SGA+ with the extended version of the linear, cosine, and SSL method at iteration 2000 and in brackets after 500 iterations.

Function $\backslash$**Rounding**	Two	Three
$f_{3c,\text{linear}}(w\|r = 0.98, n = 2.5)$	0.6220 (0.6594)	0.6175(0.6435)
$f_{3c,\text{cosine}}(w\|r = 0.98, n = 3)$	0.6187 (0.6516)	0.6175 (0.6449)
$f_{3c,\text{sigmoidlogit}}(w\|r = 0.93, n = 2.5)$	0.6175 (0.6445)	0.6203 (0.6360)

by an experiment that expresses changes in the true R-D loss performance when one interpolates between functions. Further, we evaluate three-class rounding for each of our methods. Finally, we show how SGA+ for semi-multi-rate behavior improves the performance of its predecessor and we discuss the societal impact. Note, the results of the additional experiments are obtained from the model trained with $\lambda = 0.0075$.

Temperature sensitivity Table 6.5.1 represents the stability of atanh and the SGA+ methods, expressed in true R-D loss, for different τ_{max} settings for the temperature schedule. As can be seen, the most optimal setting is around $\tau_{max} = 1$ for each of the methods. In the table we find that the linear function is least sensitive to changes in τ_{max}. To further examine the stability of the linear function compared to atanh, we subtract the best τ_{max}, column-wise, from the linear and atanh of that column. Also taking into account the sensitivity results of the one of [187] in Appendix 6.C.2 we find in general, that overall performance varies little compared to the best τ_{max} settings of the other methods and has reasonable performance. While SSL has the largest drop in performance when reducing τ_{max}, it achieves the highest performance overall for higher values of τ_{max}. If there is no budget to tune the hyperparameter of SGA+, the linear version is the most robust choice. Further, we evaluated the necessity of tuning both the latents and hyper-latents. When only optimizing the latents with the linear approach for $\tau_{max} = 1$, we found a loss of 0.6234. This is a difference of 0.0012, which implies that optimizing the hyper-latent aids the final loss.

Interpolation Table 6.5.3 represents the interpolation between different functions, expressed in true R-D loss. In Appendix 6.B.2 the corresponding loss plots can be found. Values for $a < 1$ indicate methods that tend to have larger gradients for v close to the corners, while high values of a represent a method that tends to a (reversed) step function. The smallest $a = 0.01$ diverges and results in a large loss value compared to the rest. For $a = 2.3$ we find a lowest loss of 0.6175 and for $a = 5$ we find fastest convergence compared to the rest. Comparing these model results with the model results with the one of [187], see Appendix 6.C.2, we find that the [41] model obtains more stable curves.

Table 6.5.3: True R-D loss results for the interpolation between different functions by changing a of the SSL.

a	R-D Loss
0.01	0.7323
0.3	0.6352
0.65 (approx atanh)	0.6260
0.8	0.6241
1 (linear)	0.6220
1.33	0.6199
1.6 (approx cosine)	0.6186
2.3	0.6175 $\downarrow$
5	0.6209

Three-class rounding In Table 6.5.2, the true R-D loss for two versus three-class rounding can be found at iteration $t = 2000$ and in brackets $t = 500$ iterations. For each method, we performed a grid search over the hyperparameters r and n. As can be seen in the table, most impact is made with the extended version of the linear of SGA+, in terms of the difference between the two versus three-class rounding at iteration $t = 2000$ with loss difference 0.0045 and $t = 500$ with 0.0159 difference. In Appendix 6.B.3 a loss plot of the two- versus three- class rounding for the extended linear method can be found. Concluding, the three-class converges faster. For the extended cosine version, there is a smaller difference and for SSL we find that the three-class extension only boosts performance when run for $t = 500$. In Appendix 6.C.2, the three-class experiments for the pre-trained model similar to [187] can be found. We find similar behavior as for the model of [41], whereas with the linear version most impact is made, followed by the cosine and lastly SSL. The phenomenon that three-class rounding only improves performance for $t = 500$ iterations for SSL may be due to the fact that SSL is already close to optimal. Additionally, we ran an extra experiment to asses what percentage of the latents are assigned to the 3-classes. This is run with the best settings for the linear version $f_{3c,\text{linear}}(w|r = 0.98, n = 2.5)$. At the first iteration, the probability is distributed as follows: $p(y = \lfloor v \rceil) = 0.9329$, for $p(y = \lfloor v \rceil - 1) = 0.0312$, and $p(y = \lfloor v \rceil + 1) = 0.0359$. This indicates that the class probabilities are approximately 3.12 for class -1 and 3.6 for class $+1$. This is a lot when taking into account that many samples are taken for a large dimensional latent. In conclusion, three-class rounding may be attractive under a constraint optimization budget, possibly because it is easier to jump between classes. Additionally, for the extended linear three-class rounding also results in faster convergence.

Semi-multi-rate behavior An interesting observation is that one does not need to use the same λ during refinement of the latents, as used during training. This is also mentioned in [61] for image and [184] for video compression. As a consequence of this approach, we can optimize to a neighborhood of the R-D curve without the need to train a new model from scratch. We experimented and analyze the results with methods: atanh and SSL. Figure 6.4.1c shows the performance after $t = 500$ iterations for the model of [41]. We find that SSL is moving further along the R-D curve compared to atanh. Note the refinement does not span the entire curve and that the performance comes closer together for running the methods longer, see Appendix 6.B.4. For future work it would be interesting how SGA+ compares to the methods mentioned in [61, 184], since SSL outperforms atanh.

Societal impact The improvement of neural compression techniques is important in our data-driven society, as it allows quicker development of better codecs. Better codecs reduce storage and computing needs, thus lowering costs. However, training these codes requires significant computational resources, which harms the environment through power consumption and the need for raw materials.

6.6 Discussion and Conclusion

In this paper we proposed SGA+, a more effective extension for refinement of the latents, which aids the compression performance for pre-trained neural image compression models. We showed how SGA+ has improved properties over SGA and we introduced SSL that can approximately interpolate between all of the proposed methods. Further, we showed how our best-performing method SSL outperforms the baselines in terms of the R-D trade-off and how it also outperforms the baselines on the Tecnick and CLIC dataset. Exploration of SGA+ showed how it is more stable under varying conditions. Additionally, we gave a general notation and demonstrated how the extension to three-class rounding improves the convergence of the SGA+ methods. Lastly, we showed how SGA+ improves the semi-multi-rate behavior over SGA. In conclusion, especially when a limited computational budget is available, SGA+ offers the option to improve the compression performance without the need to re-train an entire network and can be used as a drop-in replacement for SGA.

Besides being effective, SGA+ also comes with some limitations. Firstly, we run each method for 2000 iterations per image. In practice this is extremely long and time consuming. We find that running the methods for 500 iterations already has more impact on the performance and we would recommend doing this, especially when a limited computational budget is available. Future work may focus on reducing the number of iterations and maintaining improved performance. Note, higher values for a flatten out quickly, but they achieve much better gains with low-step budgets. Further, the best results are obtained while tuning the hyperparameter of SSL and for each of our tested models this lead to different settings. Note, that the experiments showed that the linear version of SGA+ is least sensitive to hyperparameter changes and we would recommend using this version when there is no room for tuning. Additionally, although three-class rounding improves the compression performance in general, it comes with the cost of fine-tuning extra hyperparameters. Finally, it applies for each method that as the temperature rate has reached a stable setting, the performance will be less pronounced, the longer you train, but in return requires extra computation time at inference.

Acknowledgments

This work was carried out on the Dutch national e-infrastructure with the support of SURF Cooperative.

Appendix

6.A Derivatives Proof

In this appendix we will proof that the normalization from the (Gumbel) softmax causes infinite gradients at 0.

Recall that the probability is given by a 2-class softmax is defined by:

$$K(v) = \frac{e^{f(v)}}{e^{f(v)} + e^{g(v)}},$$

where $f(v) = -\text{atanh}(v)$ and $g(v) = -\text{atanh}(1 - v)$. We will study the softmax for the first class 0, since the softmax is symmetric this also holds for the second class. The problem is that the gradients of the function $K(v)$ will tend to ∞ for both $v \to 1$ but also for $v \to 0$. Here we show that the gradients also tend ∞ for $v \to 0$, via the normalization with the term $g(v)$. First, take the derivative to v:

$$\frac{dK(v)}{dv} = \frac{dK(v)}{df(v)} \cdot \frac{df(v)}{dv} + \frac{dK(v)}{dg(v)} \cdot \frac{dg(v)}{dv},$$

where $\frac{dK(v)}{df(v)} = K(v)(1 - K(v))$ and $\frac{dK(v)}{dg(v)} = -K(v)\frac{e^{g(v)}}{e^{f(v)}+e^{g(v)}}$. Recall that $\frac{\text{datanh}(v)}{dv} = \frac{1}{1-v^2}$ therefore $\frac{df(v)}{dv} = -\frac{1}{1-v^2}$ and $\frac{dg(v)}{dv} = \frac{1}{1-(1-v)^2}$. Plugging this in and computing $\frac{dK(v)}{dg(v)}$ gives us:

$$\frac{dK(v)}{dv} = K(v)(1 - K(v))\left(-\frac{1}{1 - v^2}\right) - K(v) \cdot \frac{e^{g(v)}}{e^{f(v)} + e^{g(v)}} \cdot \frac{1}{1 - (1 - v)^2}.$$

Taking the limit to 0, (recall that $\lim_{v\to 0} K(v) = 1, \lim_{v\to 0} e^{f(v)} = 1$ and

$\lim_{v\to 0} e^{g(v)} = 0$) allows the following simplifications:

$$\lim_{v\to 0} 0 \cdot \left(-\frac{1}{1-0^2}\right) - 1 \cdot \frac{e^{g(v)}}{1+0} \cdot \frac{1}{1-(1-v)^2}$$

For simplicity we substitute $q = 1 - v$ (when $q \to 1$, then $v \to 0$) which will result in the following:

$$\lim_{v\to 0} -e^{-\text{atanh}(1-v)} \cdot \frac{1}{1-(1-v)^2} = \lim_{q\to 1} -e^{-\text{atanh}(q)} \cdot \frac{1}{1-q^2},$$

Recall, $-\text{atanh}(q) = -\frac{1}{2}\ln\frac{1+q}{1-q}$, so $e^{-\text{atanh}(q)} = 1/\sqrt{\frac{1+q}{1-q}}$ thus:

$$-\lim_{q\to 1}\sqrt{\frac{1-q}{1+1}} \cdot \frac{1}{1-q^2} = -\lim_{q\to 1}\sqrt{\frac{1}{2}\frac{(1-q)}{(1-q^2)^2}}$$

Since $\frac{1}{2}$ is a constant and $\lim_{x\to\infty}\sqrt{x} = \infty$ the final step is to simplify and solve:

$$-\lim_{q\to 1}\sqrt{\frac{(1-q)}{(1-q^2)^2}} = -\lim_{q\to 1}\sqrt{\frac{-1}{(q-1)(q+1)^2}} = -\infty.$$

This concludes the proof that the gradients tend to $-\infty$ for $v \to 0$.

6.B Additional Results

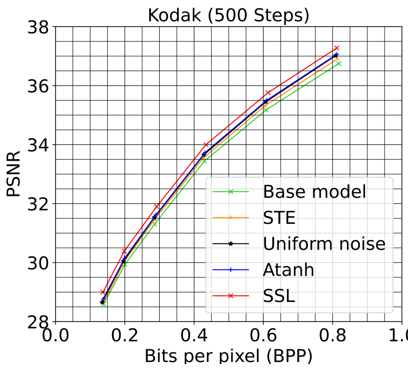
In this appendix, additional experimental results for refinement of the latents with the pre-trained models of [41] can be found.

Difference across runs Although we do not report the standard deviation for every run, the difference across runs is very small. For the model of [41] with $\lambda = 0.0032$, running five refinement procedures for 2000 iterations, results in a mean of 0.3969 and a standard deviation of $2.41 \cdot 10^{-5}$.

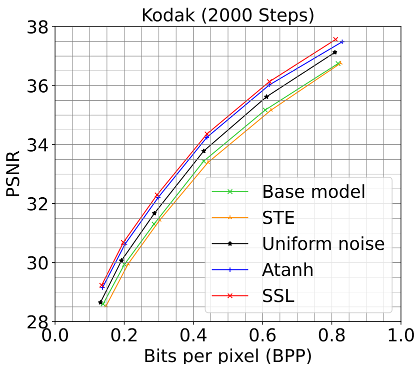
6.B.1 Additional overall performance

Figure 6.B.1 show the R-D curve for the Kodak dataset. We observe that atanh is similar to uniform noise at $t = 500$ iterations, while SSL manages to achieve better R-D gains. After $t = 2000$ iterations SLL achieves the best R-D trade-off, but the gain is a bit smaller compared to atanh at $t = 500$ iterations.

Tecnick and CLIC Figure 6.B.2 and Figure 6.B.3 show the R-D results for respectively, Tecnick and CLIC at $t = \{500, 2000\}$ iterations. We observe that SSL achieves best performance at $t = 2000$, compared to $t = 500$ iterations. However, running the method for 2000 iterations is in practice very long. Looking at the results at $t = 500$ iterations we see that there is already great improvement of performance for SSL, compared to the base model for both Tecnick and CLIC dataset.

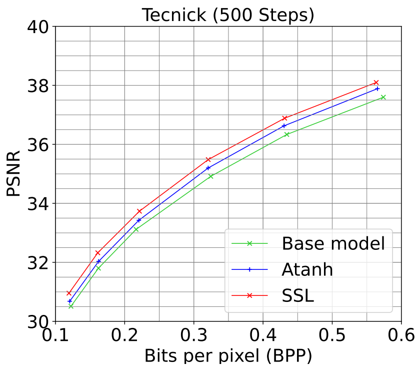


(a) Kodak ($t = 500$)

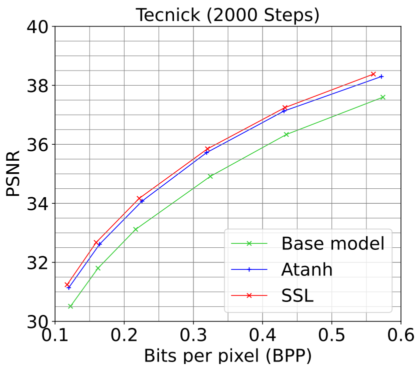


(b) Kodak ($t = 2000$)

Figure 6.B.1: Comparison of atanh and SSL on the Kodak dataset for $t = \{500, 2000\}$ iterations.



(a) Tecnick ($t = 500$)

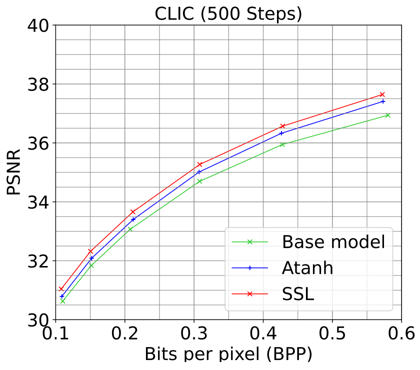


(b) Tecnick ($t = 2000$)

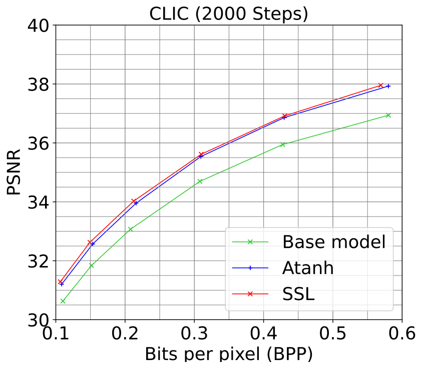
Figure 6.B.2: Comparison of atanh and SSL on the Tecnick dataset for $t = \{500, 2000\}$ iterations.

6.B.2 Interpolation

In Figure 6.B.4a the true loss, difference in loss, BPP and PSNR curves can be found. As can be seen for $a = 0.01$, the function diverges and $a = 0.3$ does not seem to reach stable behavior, both resulting in large loss values. For $a \geq 1$, the difference in losses start close to zero (see Figure 6.C.7b). SSL with $a = 5$ results in the fastest convergence and quickly finds a stable point but ends at a higher loss than most methods.



(a) CLIC ($t = 500$)



(b) CLIC ($t = 2000$)

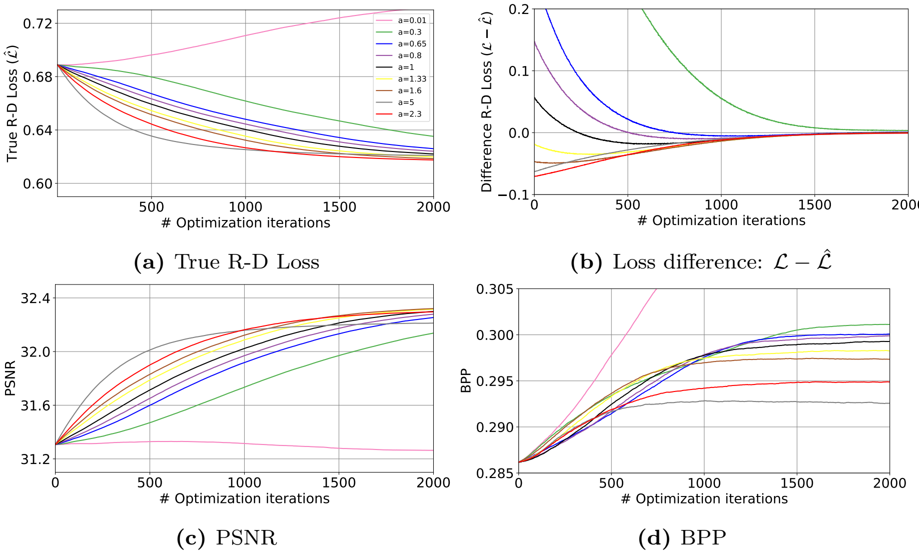
Figure 6.B.3: Comparison of atanh and SSL on the CLIC dataset for $t = \{500, 2000\}$ iterations.

6.B.3 Two- versus three- class rounding

Besides an improved RD performance for the three-class rounding, we also found that three-class rounding leads to faster convergence. In Figure 6.B.5 a true loss plot over the iterations for the linear method, can be found. As one can see, the three-class converges faster and especially on 500 iterations, boosts performance which makes it attractive under a constraint budget.

6.B.4 Semi-multi-rate behavior

In Figure 6.B.6, we have plotted the R-D curve of the base model (lime green line) and its corresponding R-D curves, obtained when refining the latents with the proposed λ's. For each model trained using $\lambda \in \{0.0016, 0.0032, 0.0075, 0.015, 0.03, 0.045\}$, we run atanh and SSL with $a = 2.3$ for $t = 2000$ iterations for all $\lambda \in \{0.0004, 0.0008, 0.0016, 0.0032, 0.0075, 0.015, 0.03, 0.045, 0.06, 0.09\}$. We depicted the base curve alongside the curves for each base model. We observe that the improved performance by SSL is especially noticeable at $t = 500$ iterations in Figure 6.B.6a.



(a) True R-D Loss **(b)** Loss difference: $\mathcal{L} - \hat{\mathcal{L}}$ **(c)** PSNR **(d)** BPP

Figure 6.B.4: Interpolation performance plots of different a settings for SLL (a) True R-D Loss (b) Difference in loss (c) PSNR (d) BPP.

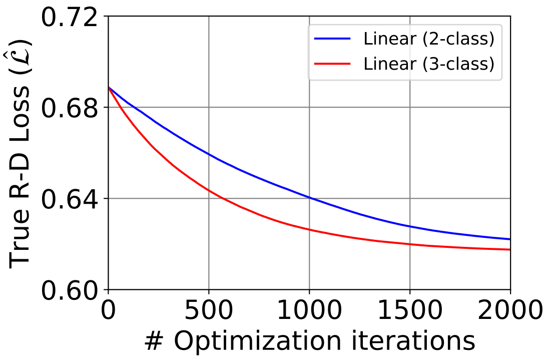
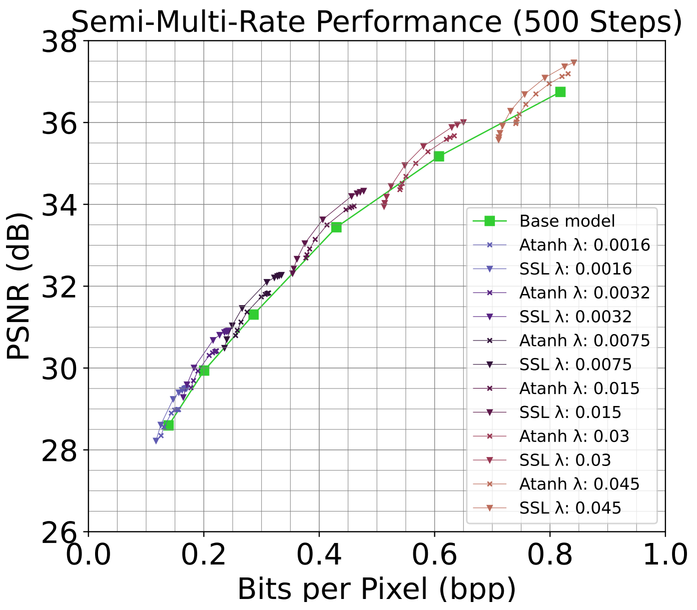
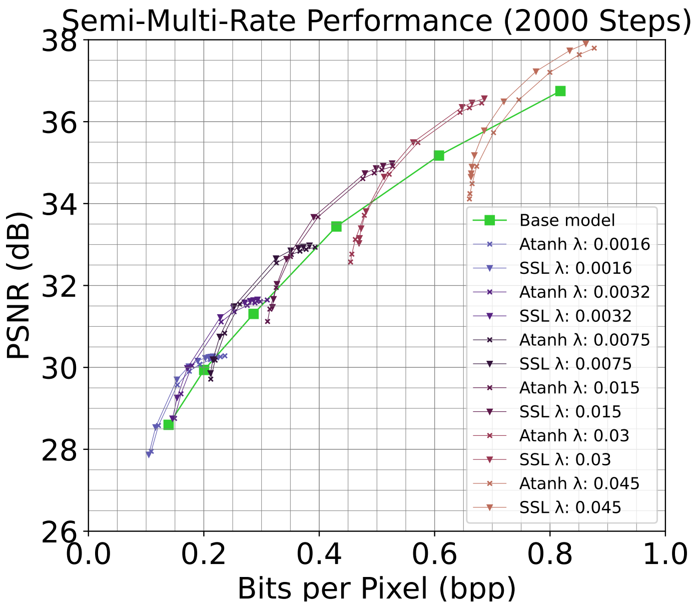


Figure 6.B.5: True R-D loss curves for two- versus three-class rounding of the linear method.



(a) $t = 500$ iterations



(b) $t = 2000$ iterations

Figure 6.B.6: R-D performance on Kodak of the [41] model when varying the target λ. Best viewed electronically.

6.C Mean-Scale Hyperprior

To make a clear comparison we trained a similar mean-scale hyperprior as in [187]. Therefore, we use the architecture of [124], except for the autoregressive part as a context model. Instead, we use the regular convolutional architecture of [24]. The model package for the mean-scale hyperprior is from CompressAI [26]. The details and results of this model can be found in this section.

Similar as for [41] we run all experiments with temperature schedule $\tau(t) = \min(\exp\{-ct\}, \tau_{max})$. Additionally, we refine the latents for $t = 2000$ train iterations, unless specified otherwise.

Implementations details The pre-trained mean-scale hyperpriors are trained from scratch on the full-size CLIC 2020 Mobile dataset [164], mixed with the ImageNet 2012 dataset [144] with randomly cropped image patches taken of size 256×256. For ImageNet, only images with a size larger than 256 for height and width are used to prevent bilinear up-sampling that negatively affects the model performance. During training, each model is evaluated on the Kodak dataset [98]. The models were trained with $\lambda = \{0.001, 0.0025, 0.005, 0.01, 0.02, 0.04, 0.08\}$, with a batch size of 32 and Adam optimizer with a learning rate set to $1e^{-4}$. The models are trained for 2M steps, except for model $\lambda = 0.001$, which is trained for 1M steps and model $\lambda = 0.08$, which is trained for 3M steps. Training runs took half a week for the 1M step model, around a week for the 2M step models, and around 1.5 weeks for the larger 3M step model. We ran all models and methods on a single NVIDIA A100 GPU. Further, the models for $\lambda = \{0.04, 0.08\}$ are trained with 256 hidden channels and the model for $\lambda = 0.001$ is trained with 128 hidden channels. The remaining models are trained with hidden channels set to 192.

Table 6.C.1: True R-D loss results for the interpolation between different functions by changing a of the SSL.

a	R-D Loss
0.01	1.15
0.3	0.7528
0.65 (approx atanh)	0.7410
0.8	0.7396
1 (linear)	0.7386
1.33	0.7380 $\downarrow$
1.6 (approx cosine)	0.7382
2.25	0.7388
5	0.7415

6.C.1 R-D Performance

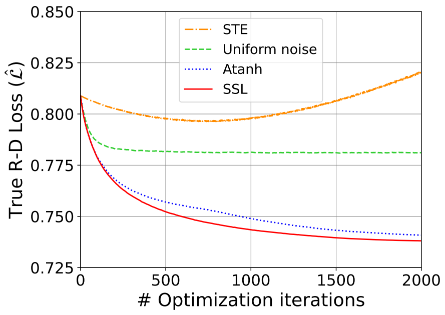
We evaluate our best-performing method SSL on the Kodak and Tecnick datasets, by computing the R-D performance, average over each of the datasets. The R-D curves use image quality metric PSNR versus BPP on the Kodak and Tecnick dataset. Recall that as base model we use the pre-trained mean-scale hyperprior, trained with $\lambda = \{0.001, 0.0025, 0.005, 0.01, 0.02, 0.04, 0.08\}$. For SSL we choose $a = \frac{4}{3}$ as we found that this setting achieves the best R-D loss overall at 2000 iterations. This is a lower setting compared to the model by [41]. The hyperparameters are similar to what we reported for [41] but with two main differences. We found that we could increase the learning rate by a factor 10 to 0.005 for atanh and SGA+. We also found that a lower $a \in [1.3, 1.4]$ was optimal.

Kodak Figure 6.C.2 shows the R-D curve for refining the latents, evaluated on Kodak. We compare SLL against baselines: STE, uniform noise, atanh and the base model at iteration $t = 500$ (see Figure 6.C.2a) and after full convergence at $t = 2000$ (see Figure 6.C.2b). As can be seen in Figure 6.C.2a, STE performs slightly better than the base model, while after $t = 2000$ iterations the method performs worse, this is also reflected in the corresponding true loss curve for $\lambda = 0.01$ (see Figure 6.C.1a), which diverges. Remarkably, for the smallest $\lambda = 0.001$, STE performs better than at $t = 500$. Adding uniform noise results in better performance when running the method longer. Comparing the R-D curves, Figure 6.B.1 to Figure 6.C.2, we find that most impact is made at $t = 500$ iterations. However, for the model similar to [187] the performance lay closer to the performance of atanh, than for the model of [41].

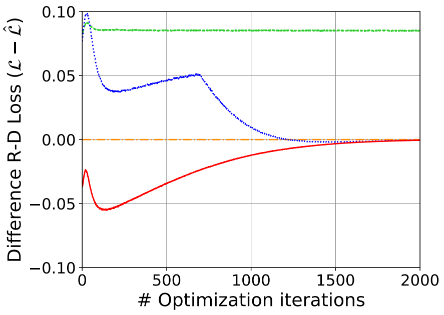
Tecnick Figure 6.C.3 shows the R-D curve when refining latents on the Tecnick dataset, after $t = 500$ and $t = 2000$ iterations. As can be seen in the plot, we find that the longer the methods run, the closer the performance lies to each other. The improvement by SSL compared to atanh is greater for Tecnick than for Kodak.

CLIC Figure 6.C.4 shows the R-D curve when refining latents on the CLIC dataset, after $t = 500$ and $t = 2000$ iterations. Similar to the previous results, we find that the longer the methods run, the closer the performance lies to each other and that running the method shorter already gives better performance, compared to the base model.

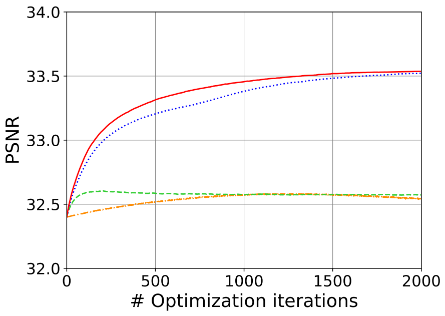
BD-Rate Gain In Table 6.C.2, we computed the change in BD-PSNR and BD-rate for the mean-scale hyperprior model. We observe that SSL is slightly better than atanh, although the difference between them is smaller than reported on the [41] model. The gap between 500 steps and 2000 steps is also smaller compared to the results in 6.4.1.



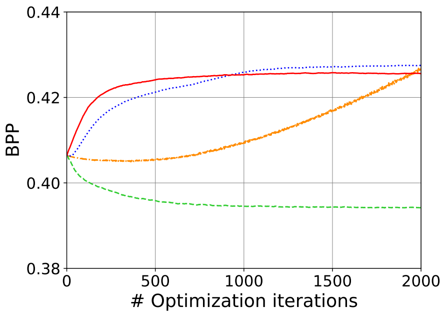
(a) True R-D loss



(b) Difference loss



(c) PSNR



(d) BPP

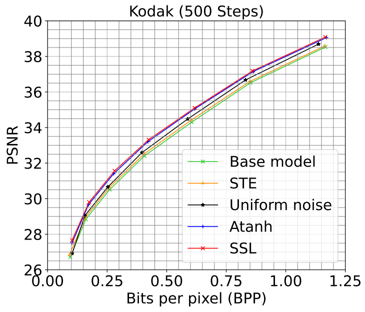
Figure 6.C.1: Performance plots of (a) True R-D Loss (b) Difference in loss (c) PSNR (d) BPP.

6.C.2 Analysis

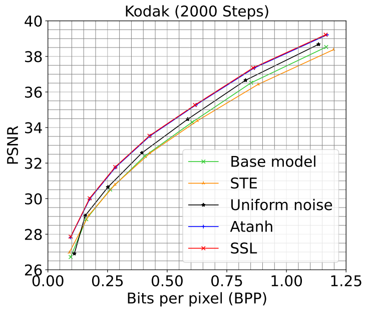
In this appendix, we analyze additional experiments for the model, similar to those in [187]. The results for the analysis are obtained from a pre-trained model trained with $\lambda = 0.01$.

Learning rates We run SSL and atanh with higher learning rate settings of 0.02 and 0.01 and compare it to the results obtained with learning rate 0.005. Figure 6.C.6 shows the corresponding loss curves and Table 6.C.3 shows the corresponding loss values at $t = \{500, 2000\}$ iterations. We find that for a learning rate of 0.01 the gap between atanh versus SSL at 500 iterations is only around 14.3% smaller and it remains pronounced, while with a learning rate of 0.01 the gap at 2000 iterations is around 28.6% better. This concludes that SSL benefits more than atanh at 2000 iterations, with a higher learning rate. More interestingly, for a learning rate of 0.02, atanh diverges whereas SSL still reaches comparable performance. This highlights the sensitivity of atanh.

Temperature sensitivity Table 6.C.4 represents the stability of atanh and the SGA+ methods, expressed in true R-D loss, for different τ_{max} settings for the

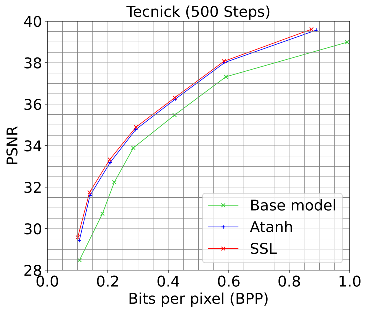


(a) R-D curve at iteration $t = 500$

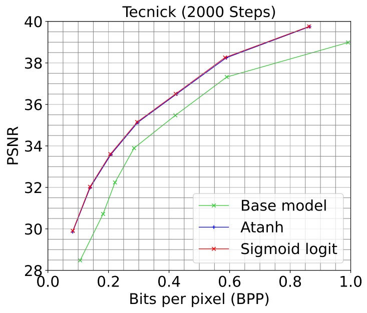


(b) R-D curve at iteration $t = 2000$

Figure 6.C.2: R-D performance on Kodak of the base mean-scale hyperprior model, STE, Uniform noise, SGA atanh and SSL with $a = \frac{4}{3}$.

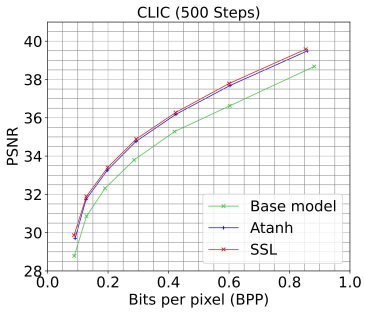


(a) R-D curve at iteration $t = 500$

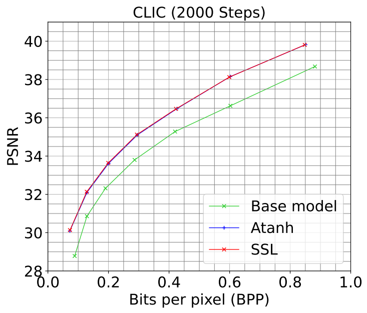


(b) R-D curve at iteration $t = 2000$

Figure 6.C.3: R-D performance on Tecnick of the base mean-scale hyperprior model, SGA atanh and SSL with $a = \frac{4}{3}$. Best viewed electronically.

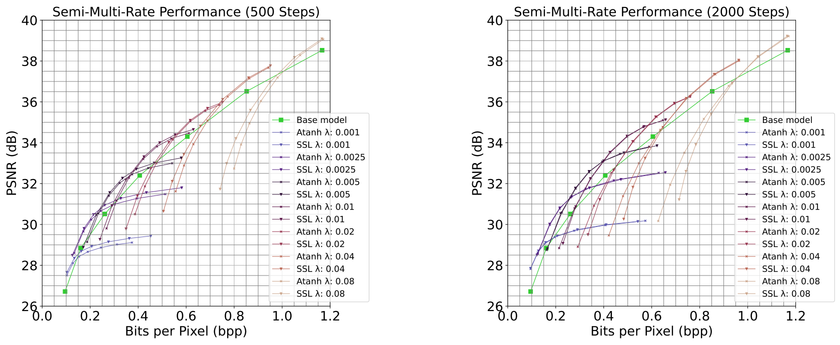


(a) R-D curve at iteration $t = 500$



(b) R-D curve at iteration $t = 2000$

Figure 6.C.4: R-D performance on CLIC of the base mean-scale hyperprior model, SGA atanh and SSL with $a = \frac{4}{3}$. Best viewed electronically.



(a) R-D semi-multi-rate curve at iteration $t = 500$

(b) R-D semi-multi-rate curve at iteration $t = 2000$

Figure 6.C.5: R-D performance on Kodak of the base mean-scale hyperprior model, SGA atanh and SSL with $a = \frac{4}{3}$. Each point is optimized with a different target $\lambda \in \{0.001, 0.0025, 0.005, 0.01, 0.02, 0.04, 0.08\}$.

Table 6.C.2: Pairwise Comparison of BD-PSNR and BD-Rate for the Kodak, Tecnick, and CLIC dataset on the Mean-Scale Hyperprior Model.

	BD-PSNR (dB)						BD-Rate (%)					
	500 steps			2000 steps			500 steps			2000 steps		
	Kodak	Tecnick	CLIC	Kodak	Tecnick	CLIC	Kodak	Tecnick	CLIC	Kodak	Tecnick	CLIC
Base vs SSL	0.68	1.21	1.03	0.91	1.50	1.33	-13.52	-22.77	-21.87	-17.69	-28.17	-27.86
Base vs Atanh	0.59	1.06	0.89	0.87	1.46	1.30	-11.90	-20.20	-19.14	-17.03	-27.49	-27.28
Atanh vs SSL	0.09	0.15	0.14	0.04	0.04	0.03	-1.80	-3.22	-3.14	-0.82	-1.03	-0.74

Table 6.C.3: True R-D loss for different learning settings of: atanh and SSL with $a = \frac{4}{3}$. At $t = 2000$ iterations and in brackets $t = 500$ iterations

Lr \Method	SSL	atanh
0.02	0.7386 (0.7506)	0.7491 (0.7627)
0.01	0.7375 (0.7498)	0.7411 (0.7540)
0.005 (base)	0.7380 (0.7521)	0.7408 (0.7570)

Table 6.C.4: True R-D loss for different τ_{max} settings of: atanh(v), linear, cosine and SSL with $a = \frac{4}{3}$. Lowest R-D loss per column is marked with: $\downarrow$. Note that the function containing atanh is unnormalized.

Function $\backslash\tau_{max}$	**0.2**	**0.4**	**0.6**	**0.8**	**1.0**
$\exp\{\text{atanh}(v)\}$	0.7445 $\downarrow$	0.7408	0.7412	0.7416	0.7418
$1 - v$ (linear)	0.7458	0.7406 $\downarrow$	0.7390 $\downarrow$	0.7386	0.7386
$\cos^2(\frac{v\pi}{2})$	0.7496	0.7414	0.7393	0.7387	0.7384
$\sigma(-a\sigma^{-1}(v))$	0.7578	0.7409	0.7391	0.7383 $\downarrow$	**0.7380** $\downarrow$
$\exp\{\text{atanh}(v)\}$	0	0.0002	0.0022	0.0033	0.0038
$1 - v$ (linear)	0.0013	0	0	0.0003	0.0006

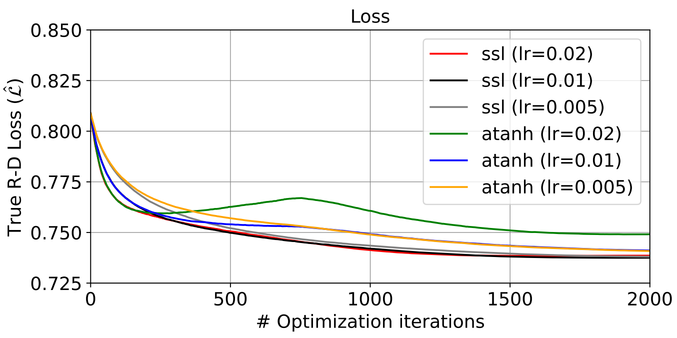


Figure 6.C.6: True R-D loss curves for different learning rates settings for method SSL and atanh.

temperature schedule. As can be seen, the most optimal setting is with $\tau_{max} = 1$ for each of the SGA+ methods. atanh obtains equal loss for $\tau_{max} \in [0.4, 0.5]$. In general, we find that the linear method of SGA+ is least sensitive to changes in τ_{max} and has equal loss between $\tau_{max} \in [0.7, 1]$. To further examine the stability of the linear function compared to atanh, we subtract the best τ_{max}, column-wise, from the linear and atanh of that column. We now see that the linear function is not only least sensitive to changes in τ_{max}, but overall varies little compared to the best τ_{max} settings of the other methods. While the SSL has the largest drop in performance when reducing τ_{max}, it achieves the highest performance overall for higher values of τ_{max}.

Interpolation Table 6.C.1 presents the true R-D loss results for the interpolation with different a settings for SSL for the mean-scale hyperprior model. In Figure 6.C.7, the corresponding overall performance of the methods can be found. As can be seen in Figure 6.C.7a, for $a = \{0.01, 0.30\}$, the functions diverge, resulting in large loss values. For $a = 0.65$, we find that the loss curve is slightly unstable at the beginning of training, which can be seen in the bending of the curve, indicating non-optimal settings. This may be due to the fact that we run all methods with the same $\tau_{max} = 1$ for a fair comparison. Additionally, note that SSL with $a = 0.65$ obtains a true R-D loss of 0.7410 compared to 0.7418 for atanh with the same settings. This is due to the fact that SSL, especially in the tails of the probability, is slightly more straight-curved compared to the atanh when looking at its probability space.

Remarkably, for $a \geq 1$, the difference in losses start close to zero (see Figure 6.C.7b). SSL with $a = 5$ results in the fastest convergence and quickly finds a stable point but ends at a higher loss than most methods.

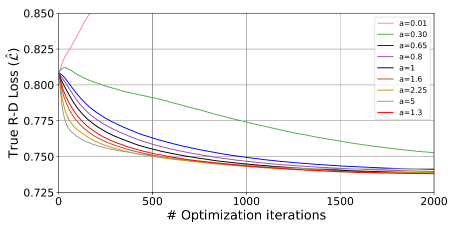
Three-class rounding Table 6.C.5 shows the true R-D loss for two- versus three-class rounding, at iteration $t = 500$ and in brackets $t = 2000$ iterations. For each method, we performed a grid search over the hyperparameters r and

Table 6.C.5: True R-D loss of two versus three-class rounding for SGA+ with the extended version of the linear, cosine, and SSL method at iteration 500 and in brackets after 2000 iterations.

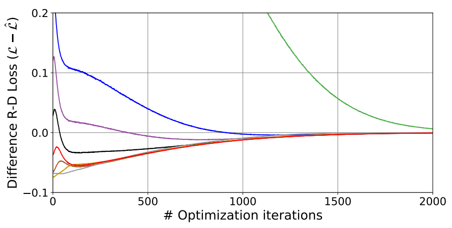
Function \Rounding	Two	Three
$f_{3c,\text{linear}}(w \mid r = 0.98, n = 1.5)$	0.7552 (0.7386)	0.7517(0.7380)
$f_{3c,\text{cosine}}(w \mid r = 0.98, n = 2)$	0.7512 (0.7384)	0.7513 (0.7379)
$f_{3c,\text{sigmoidlogit}}(w \mid r = 0.93, n = 1.5)$	0.7524 (0.7380)	0.7504 (0.7380)

n. Additionally, for the extended SSL, we also performed a grid search over a and found the best setting to be $a = 1.4$. As can be seen in the table, most impact is made with the extended version of the linear of SGA+, in terms of the difference between the two versus three-class rounding at iteration $t = 500$ with loss difference 0.0035 and $t = 2000$ with 0.0006 difference. There is a small difference at $t = 500$ for the extended cosine version. In general, we find that running models longer results in convergence to similar values. SSL converges to equal values for two- and three-class rounding.

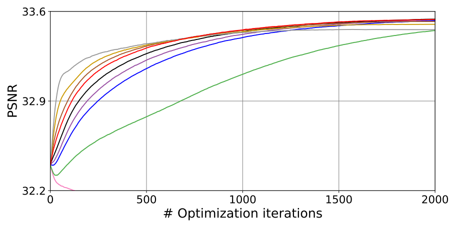
Semi-multi-rate behavior Similar as in Appendix 6.B.4, we experimented with different values for λ to obtain a semi-multi-rate curve. For every pre-trained model, we ran SSL and atanh using $\lambda \in \{0.001, 0.0025, 0.005, 0.01, 0.02, 0.04, 0.08\}$. In Figure 6.C.5, we have plotted the R-D curve of the base model (lime green line) and its corresponding R-D curves, obtained when refining the latents with the proposed λ's for atanh and SSL. As can be seen, running the methods for $t = 500$ iterations, SSL obtains best performance. While the longer you train, the closer together the performance will be.



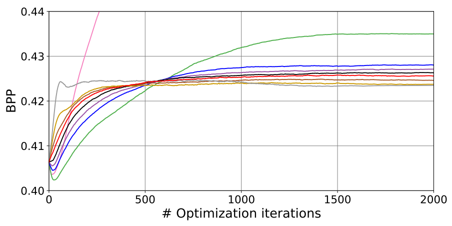
(a) True R-D Loss



(b) Loss difference: $\mathcal{L} - \hat{\mathcal{L}}$



(c) PSNR



(d) BPP

Figure 6.C.7: Interpolation performance plots of different a settings for SLL (a) True R-D Loss (b) Difference in loss (c) PSNR (d) BPP under the mean-scale hyperprior model.

CHAPTER 7

Meta-Learning for Federated Face Recognition in Imbalanced Data Regimes

Based on [3]:
Arwin Gansekoele, Emiel Hess, and Sandjai Bhulai
Meta-Learning for Federated Face Recognition in Imbalanced Data Regimes
Published in the 2024 IEEE International Conference on Federated Learning Technologies and Applications (FLTA) proceedings

Abstract

The growing privacy concerns surrounding face image data demand new techniques that can guarantee user privacy. One such face recognition technique that claims to achieve better user privacy is Federated Face Recognition (FRR), a subfield of FL. However, FFR faces challenges due to the heterogeneity of the data, given the large number of classes that need to be handled. To overcome this problem, solutions are sought in the field of personalized FL. This work introduces three new data partitions based on the CelebA dataset, each with a different form of data heterogeneity. It also proposes Hessian-Free Model Agnostic Meta-Learning (HF-MAML) in an FFR setting. We show that HF-MAML scores higher in verification tests than current FFR models on three different CelebA data partitions. In particular, the verification scores improve the most in heterogeneous data partitions. To balance personalization with the development of an effective global model, an embedding regularization term is introduced for the loss function. This term can be combined with HF-MAML and is shown to increase global model verification performance. Lastly, this work performs a fairness analysis, showing that HF-MAML and its embedding regularization extension can improve fairness by reducing the standard deviation over the client evaluation scores.

7.1 Introduction

During the past decade, significant advances have been made in AI and ML. The surge in available data and computing power has enabled the development of technologies such as large language models, image classification, and image generation. However, these advancements also bring challenges related to data collection and storage. Issues such as data ownership and privacy are becoming more relevant in a world that is becoming increasingly digital. A domain with a particular sensitivity to privacy concerns is *face recognition* (FR). Datasets such as LFW [85], MegaFace [128], MS-Celeb-1M [69], and IJB-C [122] have been instrumental in advancing FR technology. Of these datasets, both MegaFace and MS-Celeb-1M have been redacted due to privacy issues at the time of writing. Addressing these privacy concerns is essential for the ethical advancement of face recognition technologies.

An approach to using large-scale datasets while respecting privacy constraints is FL [123]. By aggregating data from multiple parties, we can share large amounts of information while alleviating privacy issues. In the context of FR, this approach is referred to as FFR. A significant challenge with FFR is that making i.i.d. data assumptions across clients is unrealistic. Different data owners typically have distinct identities, which means that everyone is solving their own local classification problem. Furthermore, identities may vary in terms of attributes depending on the part of the world the data originate from. Ideally, these algorithms should be not only effective but fair as well. Disparities in model performance could discourage participation from certain parties.

This challenge of data heterogeneity is not new in the field of FL [195, 198].

Attempts to address data heterogeneity can be dubbed personalized federated learning (PFL). Multiple authors have proposed algorithms to address this issue in FL and FFR [9, 115]. One approach that is underexplored, yet interesting in the context of FFR is meta-learning. Meta-learning algorithms, such as *model-agnostic meta learning* MAML [58], optimize the model in such a way that it can quickly adapt to any new task. MAML has previously been adapted to an FL setting by [55] and [91]. Their application in data heterogeneous settings is interesting, as meta-learning provides a natural multi-task framework. In this work, we propose the use of MAML in an FFR setting and analyze its performance. We make the following contributions.

1. We introduce the use of meta-learning in the context of FFR and propose an additional regularization term to alleviate the problem of global/local model mismatch.
2. We introduce two new splits for the dataset CelebA [117] that allow us to evaluate FFR models under data heterogeneity.
3. We evaluate our approach and demonstrate that meta-learning, in the absence of a global dataset, outperforms FedAvg [123] in both TAR@FAR and fairness under data heterogeneity.

7.2 Related Work

Federated Face Recognition While the naming of FFR is new, the CelebA dataset has long been included in FL benchmarks such as LEAF [32]. The introduction of face recognition functions such as CosFace [173] in an FL setup followed later [9, 115, 132]. FedFace [9] identified the issue of sensitive information being stored in the classification head of the global model. Another problem occurs when the assumption is made that every client only has one identity to train on; there are no negative samples to balance out the loss. They addressed this by pre-training a classifier on a global dataset and then regularizing the embeddings at the server during FL. FedFR [115] relaxed the assumption of a single identity per client, demonstrating improvements through a multi-task learning approach. However, note that both these approaches assume the availability of a large global dataset to start training. We drop this assumption in our work.

Meta Learning MAML was first introduced in [58] with [54] providing convergence guarantees for this approach. Work by [38], [55], and [91] demonstrated that meta-learning can be applied in an FL setting. Work by [91] showed that the FedAvg algorithm can be interpreted as a form of meta-learning. Work by [186], the authors proposed an adaptive method to divide clients into groups, where each group is selected to be similar in terms of data heterogeneity. An issue with MAML is that you need to compute an expensive second derivative. That is why [54] proposed Hessian-Free MAML (HF-MAML), which provides an approximation to the second derivative that avoids the need to compute the Hessian. We use this approximation in our meta-learning approach.

Non-IID Data The issue of data heterogeneity is well-known in FL. While FedAvg is robust to non-i.i.d. data in certain cases [123], other papers have reported accuracy reductions of up to 55% in highly imbalanced data regimes [195]. To properly evaluate our models, we need data partitions that allow exploring performance under non-i.i.d. settings. The most common data partition in FFR is one class per client [32]. Another commonly used partition was used by [115, 151], where each client received an equal number of classes > 1.

Fairness An algorithm needs to be fair to be compliant with legislation and for broader adoption. Multiple works have examined how to define fairness in an FL system [108, 109]. In work by [109], q-FedAvg is proposed to balance the performance across clients. They do so using a q parameter that weights the parameter updates by their loss values. Clients with a lower loss value are less influential for the final parameter update. They also proposed a variant with meta-learning.

7.3 Methods

7.3.1 Federated HF-MAML for FFR

To improve the personalization in an FL setup for FFR, we propose the use of HF-MAML. MAML is model-agnostic, which means it is compatible with any model that uses gradient descent [58]. MAML (and HF-MAML) has been shown to work well with classification models [55, 58, 91], but is more complicated in FR since we have to assume that we know the total number of identities of all clients combined beforehand.

Previous works have shown that one can naturally interpret meta-learning in an FL setup. Meta-learning normally works with different sets of tasks and computes the gradient of the model in such a way that the distance is minimal for all tasks. In FL, one can interchange 'task' with 'client' and take the same approach. FL is essentially multitask learning where each client forms their own task. To perform meta-learning on gradient-based approaches, one would normally compute the Hessian to minimize the needed gradient update, as this matrix represents the rate of change of functions. However, this is expensive in terms of computation and memory. We make an estimate of the Hessian by sampling multiple datasets and performing the following weight update.

$$w_{t+1}^k = w_t^k - \beta \nabla_{\tilde{w}_t^k} f(\tilde{w}_t^k, D'_k)[I - \alpha \nabla^2_{w_t^k} f(w_t^k, D''_k)]. \tag{7.1}$$

Here, we define D_k as the dataset of client k, w_t^k the weights at timestep t for client k, f the differentiable function. We denote D'_k and D''_k as two extra datasets sampled from client k to get an estimate for the 2nd derivative. We define $\tilde{w}_t^k$ as

$$\tilde{w}_t^k = w - \alpha \nabla f(w, D). \tag{7.2}$$

We then approximate $\nabla^2_{w_t^k} f(w_t^k, D_k'')$ using

$$\nabla^2_w f(w, D_k'')\nabla_{\tilde{w}} f(\tilde{w}, D_k') = \frac{1}{2\delta}\big[\nabla_w f(w + \delta\nabla_{\tilde{w}} f(\tilde{w}, D_k'), D_k'') - \nabla_w f(w - \delta\nabla_{\tilde{w}} f(\tilde{w}, D_k'), D_k'')\big]. \quad (7.3)$$

This provides us with the weight update that we need for our approaches.

Global Classification The classification layer of the model refers to the part of the model after the backbone. This module is usually optimized by a loss function, such as Softmax, CosFace, or ArcFace. If we send all clients these weights as well, they all get access to the embeddings within this module. This poses privacy risks, as it is known that the original face could be reconstructed from the final embedding. As this is often still considered a valid approach, we include this approach in our evaluation suite as a global classification.

Another issue with sharing the global classification layer occurs because not all clients possess data from all classes. Computing, e.g., a softmax over all classes, does not make sense when not all classes are seen during training. That is why if a client knows that they do not possess data from some class c_i, they set the logits for the class to $-\infty$. This essentially removes the influence of that class on the final classification problem, meaning they only use the local classes for optimization.

Local Classification A safer alternative to maintaining a global classification layer is to keep the classification layer local and not share the weights of this layer with the global model. This approach is intuitive in the context of FFR, as the assumption that no classes are shared between clients is realistic. Instead of sending weight updates for all layers, we only send weight updates for the backbone layer to the global model.

Embedding regularization Note that the lack of overlap between classes could cause local models to drift apart. We propose adding an additional loss term to the local classification layer to alleviate the model drift issue. Adding a regularization term to the local loss function in FFR is not new. Both [110] and [115] added regularization terms to reduce the divergence of local weights. Instead of adding a penalty to the divergence between weights, we propose to reduce the divergence between the embeddings generated as in Equation (7.4).

$$\text{Loss} = f(x) + C(1 - S_c(f_{\text{emb}}^{\text{global}}(x), f_{\text{emb}}^{\text{local}}(x))), \quad (7.4)$$

where S_c refers to the cosine similarity. With the regularization penalty, every client computes the cosine distance between the starting model's embeddings and the current model's embeddings. This term is weighted by a factor of C, which allows a trade-off between increasing personalization and reducing model divergence.

Combined algorithm Algorithm 2 shows the federated HF-MAML algorithm that we propose for FFR. The parts marked dark blue indicate algorithm additions for the implementation of the local classification layer, the parts marked green refer to using a global classification layer, the part marked red represents embedding regularization, and the parts with cyan color are algorithm additions to make HF-MAML work in an FFR setting. It describes our HF-MAML based approach that we use for effective personalized FFR.

Algorithm 2 Federated HF-MAML local regularized

```
Server
Initialize $w_0^{\text{server}}$ on the server
if Local classification layer then
    Initialize $a_0^k$ as the classification weights on each client$_k$, $1 \leq k \leq K$
end if
if Global classification layer then
    Initialize $a_0^{\text{server}}$ as the classification weights on the server
end if
for $t$ rounds do
    for client$_k$, $1 \leq k \leq K$ do
        Sample set $D_k$ from client$_k$ training dataset
        $w_t^k = w_t^{\text{server}}$
        if Global classification layer then
            $a_t^k = a_t^{\text{server}}$
        end if
        Define $F(w, a, D) = f(w, a) + C(1 - S_c(f_{\text{emb}}^{\text{global}}(w, D), f_{\text{emb}}^{\text{local}}(w, D)))$
        $(\tilde{w}_{t+1}^k, \tilde{a}_{t+1}^k) = (w_t^k, a_t^k) - \alpha \nabla F(w_t^k, a_t^k, D_k)$
        Sample set of images $D'_k$ and $D''_k$ from client$_k$ training dataset different from $D_k$
        $(w_{t+1}^k, a_{t+1}^k) = (w_t^k, a_t^k) - \beta \nabla_{\tilde{w}_t^k, \tilde{a}_t^k} f(\tilde{w}_t^k, \tilde{a}_t^k, D'_k)[I - \alpha \nabla^2_{w_t^k, a_t^k} f(w_t^k, a_t^k, D''_k)]$
        Send $w_{t+1}^k$ back to the server
        if Global classification layer then
            Send $a_{t+1}^k$ back to the server
        end if
    end for
    $w_{t+1} = \sum_{k=1}^K \frac{n_k}{\sum_{i=1}^K n_i} w_{t+1}^k$
    if Global classification layer then
        $a_{t+1} = \sum_{k=1}^K \frac{n_k}{\sum_{i=1}^K n_i} a_{t+1}^k$
    end if
end for
```

7.3.2 Data Partitions

We evaluate three types of data partitions, two of which we propose in this work. We assume the possibility of multiple identities per client, which gives us an equal

Table 7.3.1: Train and test sizes for each client for the lognormal client partition.

client	train data	test data	client	train data	test data
1	134	32	11	264	65
2	1,488	376	12	389	99
3	748	184	13	2,203	554
4	95	23	14	222	54
5	259	66	15	287	69
6	2,585	650	16	276	71
7	27	7	17	4,657	1,163
8	346	86	18	4,307	1,076
9	1,955	495	19	1,326	334
10	220	58	20	773	196

class partition as a starting point. This partition tries to divide the number of identities and examples as evenly as possible per client. To better explore the effect of data heterogeneity on different algorithms, we propose two new data partitions: the lognormal partition and the attribute-based partition.

Lognormal class partition

$$C\frac{S_i}{\sum_j S_j}, S_i \sim lognormal(\mu, \sigma), \quad 1 \leq i \leq n. \tag{7.5}$$

First, we propose the lognormal class partition as defined in Equation (7.5). For each client, we draw a sample from the lognormal distribution. We normalize these samples by the sum of all random samples and multiply them by the total number of identities. The result then gives the number of identities a client should be randomly assigned. Using the lognormal distribution in such a way gives us a few clients with many identities and many clients with few identities. The idea of using a lognormal distribution has been proposed previously [132]. However, they instead partition the data itself instead of the identities, which creates a quantity skewness in the data. We found it more intuitive for FFR to create an identity skew, as this classification problem already inherently consists of many classes with few samples per identity.

Attribute-based partition Finally, we propose a data partition type based on an attribute partition. The images in the CelebA dataset have not only been labeled based on identity but on more features, such as whether the person has a beard or not, or if the person is wearing make-up. These attributes can be used for classification or to improve face recognition based on the dependence of attributes [73]. Different attributes can be combined to create more specific partitions. For example, a client has a dataset that contains only images of men with beards and glasses.

This data partition is based on the idea of the FEMNIST dataset that is included in the LEAF benchmark. For FEMNIST, each client is assigned handwritten

Table 7.3.2: Different combinations of attributes that are assigned to each client.

Cl.	Gender	Glasses	Hat	Old/Yng	Hair	Attr 6	Attr 7	Attr 8	Size
1	Male	Yes	No	Young	All Types				1,066
2	Male	Yes	No	Old	Grey				1,007
3	Male	Yes	No	Old	All Types	Chubby			1,001
4	Female	Yes	No	Both	All Types				925
5	Both	No	Yes	Both	All Types				1,010
6	Female	No	No	Both	Blond	Oval Face	Rosy Cheeks		1,109
7	Male	No	No	Both	All Types	Goatee	Not bald	Smiling	1,044
8	Male	No	No	Both	All Types	No Goatee	Bald		1,019
9	Male	No	No	Both	Gray	No Goatee			1,040
10	Male	No	No	Old	All Types	No Goatee	Bushy Eyeb.		1,062
11	Male	No	No	Old	No black	No Goatee	Bushy Eyeb.		1,007
12	Male	No	No	Old	No blond	No Goatee	Bushy Eyeb.		1,062
13	Male	No	No	Young	Brown	No Goatee	Bushy Eyeb.		1,077
14	Female	No	No	Old	All Types	Oval Face	Lipstick	No R. Chks	1,076
15	Female	No	No	Young	All Types	No Oval Face	Lipstick	Rosy Chks	1,248
16	Male	No	No	Both	All Types	No Bushy Eyeb.	Mustache	No Bushy Eyeb.	1,077
17	Male	No	No	Young	All Types	Beard	No Mustache	Bushy Eyeb.	1,147
18	Male	No	No	Young	All Types	Beard	No Mustache	No Bushy Eyeb.	1,108
19	Male	No	No	Young	Black	Bags u. Eyes			1,039
20	Female	No	No	Old	All Types	Bags u. Eyes			1,029

letters according to the writer. This results in each client having data in different handwriting giving a form of feature-based skewness.

7.4 Experimental Setup

7.4.1 Data partitioning

This section provides the concrete partitions generated using our approach. We will make these partitions available for future research. In total, there are 20 clients, 15 of which are used for training and 5 for evaluation. A total of 1,500 classes were used for the experiment. Data for each client are divided into a train, validation and test set, in a 70%/10%/20% split. We used around 10% of the original dataset due to resource constraints. All samples were cropped using Haar cascades.

Equal class partition In the equal class partition, the data set for each client consists of images belonging to 75 randomly assigned identities, without overlap between clients. Since each class has a different sample size, the total number of images per client can differ.

Lognormal class partition For the lognormal distribution, we used $\mu = 3$ and $\sigma = 3$. We sample a partition once and store this partition for all runs. The amount of test data and training data per client is shown in Table 7.3.1.

Attribute-based partition The CelebA dataset has a set of 40 binary attributes per image. These attributes range from hair color to whether or not the person is wearing glasses. Since some attributes are more common than others,

Table 7.4.1: The model architecture.

Layer type	In	Out
ResNet 18 (pre-trained)	$3 \times 128 \times 128$	1,000
Linear	1,000	512
Arcface ($s = 8$, $m = 0.5$)	512	1,500

Table 7.4.2: The hyperparameters used for optimization per method.

All		HF-MAML	
Learning rate	0.01	α	0.01
Momentum	0.9	β	0.1
		δ	0.001

different combinations of attributes are used to create subsets of data that can be assigned to clients. Table 7.3.2 shows how we have assigned attributes for each client. For example, client 5 has images with both male and female images. It is the only client with people without glasses. They wear hats, they can be both old and young, and the client has images of people with all types of hair color. In short, the focus of client 5 is on people with hats. Client 12 only contains images of men who do not wear glasses or hats. The people in the images are old people without blond hair, which means they can have any hair color except blond. The people in the images do not have goatees, which means that all men with goatees are filtered out. Lastly, all have bushy eyebrows. When an attribute is not mentioned it means that the client is agnostic towards this attribute; it has both people with or without this attribute in its dataset. An identity may have images with different attribute distributions. As can be seen in Table 7.3.2, dataset sizes are kept as consistent as possible to reduce the effect of quantity skew as much as possible.

7.4.2 Model Setup

The model architecture used to train both the HF-MAML and FL models is shown in Table 7.4.1. We used the Arcface loss function with $s = 8$ and $m = 0.5$, which were chosen through tuning. Important to note is that Tab 7.4.1 depicts the global classification layer scenario with 1,500 identities. For the local classification layer, the output of the Arcface part depends on the number of local identities. The linear layer between ResNet and Arcface is used to cast the ResNet output vector of size 1,000 to the size required for the embedding vector. In this paper, the embedding vector will have a length of 512. This embedding vector is used to compute the cosine similarity. As described previously, this cosine similarity is used for model evaluation and embedding regularization.

For both training and tuning, we use a stochastic gradient descent optimizer with a learning rate of 0.01 and a momentum of 0.9. Table 7.4.2 shows the hyperparameters used for the HF-MAML algorithm. The chosen hyperparameters were found to be sufficient for our experiments.

7.4.3 Training and Evaluation

Training is performed for 30 rounds and local training is performed once on each client for 50 epochs. For the FedAvg algorithm, one epoch means 50 batches of size 64. For the HF-MAML algorithm, we used three sampling rounds with 64 images per sampling round. In each epoch, the algorithm sampled three batches of size 64. Both FL and HF-MAML were tuned to determine the optimal number of epochs used. Using 50 epochs seemed best in both cases. This process was repeated 10 times with different seeds to obtain our results.

During the evaluation, each training run is evaluated 5 times for the global model and 5 times for the tuned model. In scenarios with a local classification layer, the local classification weights are combined with the global backbone weights to tune the model. We report both the average and standard deviation, the latter serving as the backbone for our fairness analysis.

We perform our evaluation in a way similar to [115]. Our evaluations differ from the IARPA Janus Benchmark [122] as we are primarily interested in the performance per client. We perform a stratified split per client, which means that every identity is in both the train and the test set. We sample a batch of pairs where half the pairs have the same identity and the other half are different. The ones of a different origin may originate from other client test sets. Based on these pairs, we measure the verification performance in TAR@FAR.

7.5 Results

7.5.1 Convergence analysis

Figure 7.5.1 shows the average validation score per communication round for the first 15 clients. We compared the global model with the tuned model for both. Figure 7.5.2 shows the same validation score per round but for the 5 clients not included during the training. The tuning of the clients included during the training seems to improve the scores, while we do not see this for the clients included later. We found similar training curves for the other methods.

7.5.2 Performance Comparison

First, we compared the global and local models for both FedAvg and HF-MAML in Table 7.5.1 under an equal class partition. We observed that HF-MAML achieves a better TAR@FAR than FedAvg, especially after tuning locally. We also saw little difference between the local and global models, which implies that it is sufficient to keep the classification layer local.

Second, we looked at the performance under the lognormal partition in Table 7.5.2. We observed that the difference between HF-MAML and FedAvg is greater than

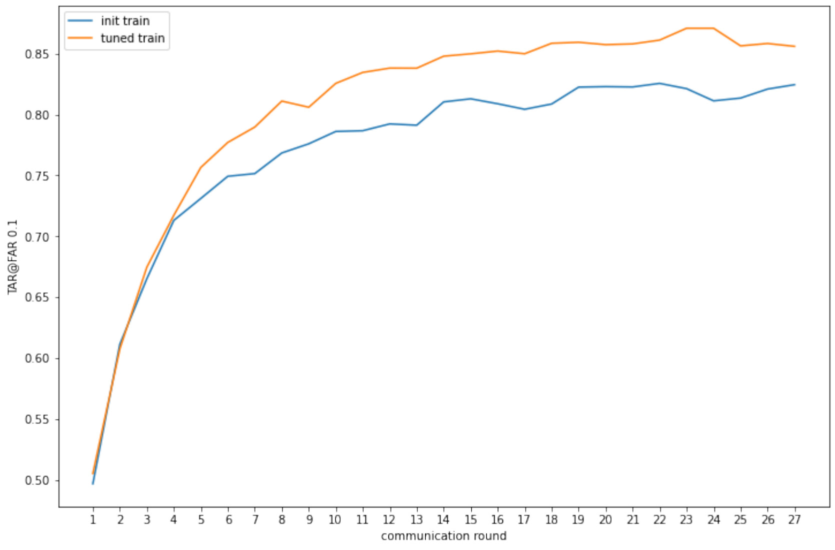


Figure 7.5.1: HF-MAML with equal class partition. TAR@FAR 0.1 results after each communication round with the clients 1-15; before and after tuning with 5 batches.

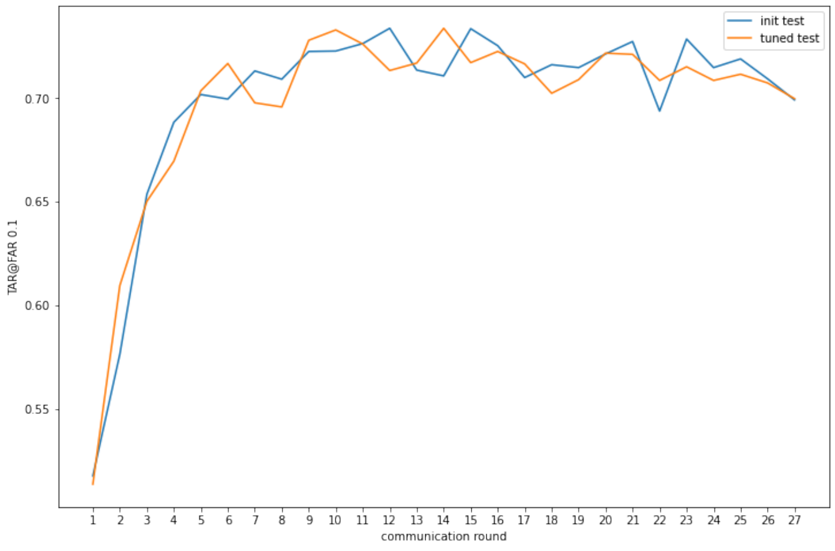


Figure 7.5.2: HF-MAML model with equal class partition. TAR@FAR 0.1 results after each communication round with the clients 16-20; before and after tuning with 5 batches.

Table 7.5.1: TAR@FAR 0.1 for the equal class partition dataset. Bold implies the best result.

	Train	Test
	Untuned	
FedAvg - Global	0.783 ± 0.006	0.710 ± 0.016
FedAvg - Local	0.779 ± 0.006	0.718 ± 0.013
HF-MAML - Global	0.795 ± 0.007	0.705 ± 0.016
HF-MAML - Local	$\mathbf{0.804} \pm 0.007$	$\mathbf{0.737} \pm 0.014$
	Tuned	
FedAvg - Global	0.836 ± 0.006	0.760 ± 0.012
FedAvg - Local	0.817 ± 0.006	0.740 ± 0.013
HF-MAML - Global	$\mathbf{0.855} \pm 0.005$	$\mathbf{0.762} \pm 0.013$
HF-MAML - Local	0.852 ± 0.006	0.768 ± 0.013

Table 7.5.2: TAR@FAR0.1 for the lognormal class partition dataset. Bold implies the best result.

	Train	Test
	Untuned	
FedAvg - Global	0.707 ± 0.013	0.675 ± 0.013
FedAvg - Local	0.715 ± 0.013	0.686 ± 0.013
FedAvg - Regularization	0.709 ± 0.009	0.704 ± 0.013
HF-MAML - Global	0.722 ± 0.012	0.695 ± 0.012
HF-MAML - Local	0.725 ± 0.014	0.705 ± 0.016
HF-MAML - Regularization	$\mathbf{0.734} \pm 0.011$	$\mathbf{0.726} \pm 0.011$
	Tuned	
FedAvg - Global	0.788 ± 0.012	0.715 ± 0.016
FedAvg - Local	0.786 ± 0.013	0.707 ± 0.014
FedAvg - Regularization	0.773 ± 0.008	0.715 ± 0.017
HF-MAML - Global	$\mathbf{0.810} \pm 0.014$	$\mathbf{0.737} \pm 0.013$
HF-MAML - Local	0.802 ± 0.014	0.731 ± 0.012
HF-MAML - Regularization	0.798 ± 0.011	0.735 ± 0.013

Table 7.5.3: TAR@FAR 0.1 for the attribute-based partition dataset. Bold implies the best result.

	Train	Test
	Untuned	
FedAvg - Global	0.718 ± 0.007	0.572 ± 0.014
FedAvg - Local	0.754 ± 0.008	0.636 ± 0.013
FedAvg - Regularization	0.764 ± 0.009	0.673 ± 0.014
HF-MAML - Global	0.784 ± 0.008	0.653 ± 0.015
HF-MAML - Local	$\mathbf{0.789} \pm 0.009$	0.679 ± 0.011
HF-MAML - Regularization	$\mathbf{0.789} \pm 0.009$	$\mathbf{0.708} \pm 0.014$
	Tuned	
FedAvg - Global	0.788 ± 0.009	0.672 ± 0.017
FedAvg - Local	0.772 ± 0.008	0.695 ± 0.013
FedAvg - Regularization	0.779 ± 0.008	0.694 ± 0.015
HF-MAML - Global	$\mathbf{0.838} \pm 0.008$	0.755 ± 0.016
HF-MAML - Local	0.816 ± 0.006	$\mathbf{0.759} \pm 0.011$
HF-MAML - Regularization	0.812 ± 0.006	0.729 ± 0.011

for the equal partition. While there was initially no difference in TAR@FAR0.1 for the clients added after training under the equal partition, the lognormal partition showed a larger and more significant difference between FedAvg and HF-MAML. This suggests that HF-MAML may be more effective under quantity skew.

Another interesting result is that adding model regularization seems best when one does not tune locally afterward. Embedding regularization seems to be effective in addressing embedding drift, especially for new clients.

Finally, we compared our test bench under the attribute-based partition in Table 7.5.3. This partition has the same amount of data per client but has mixed attributes. This partition thus allows us to evaluate under feature skew. In this partition, we see a clear improvement from FedAvg to HF-MAML after tuning the local data. This effect is visible for both the clients known during training and the clients added after training. We also see the same effect as observed under lognormal for embedding regularization.

7.5.3 Fairness Analysis

Another vital aspect of PFR is fairness between clients. If clients know in advance that they will mostly contribute and do not receive much from collaborative training, they are unlikely to participate. To evaluate the fairness of our approaches, we note the standard deviation of our TAR@FAR 0.1 scores. The variance of the evaluation metrics is commonly used to measure the fairness of an approach.

In Table 7.5.4, we noted the standard deviation of the compared methods under

Table 7.5.4: Standard deviation of client's evaluation scores for the equal partition dataset. Bold implies the best result.

	Train	Test
	Untuned	
FedAvg - Global	0.031 ± 0.005	0.021 ± 0.010
FedAvg - Local	0.033 ± 0.006	0.020 ± 0.009
HF-MAML - Global	$\mathbf{0.025} \pm 0.005$	0.010 ± 0.008
HF-MAML - Local	0.031 ± 0.006	$\mathbf{0.009} \pm 0.009$
	Tuned	
FedAvg - Global	0.027 ± 0.005	0.012 ± 0.010
FedAvg - Local	0.031 ± 0.004	0.018 ± 0.009
HF-MAML - Global	$\mathbf{0.023} \pm 0.005$	$\mathbf{0.007} \pm 0.008$
HF-MAML - Local	0.028 ± 0.005	0.012 ± 0.010

the equal partition. We see that local tuning of the global model improves the fairness of the model for both the FedAvg and HF-MAML models. We also see that the difference between the two under equal class partitions is small. This implies that using meta-learning when the data is approximately balanced does not add much to fairness.

In Table 7.5.5, we show the fairness results for the lognormal partition. We see that the variance between clients is overall higher than what we observed in Table 7.5.4, which is a logical consequence of the skew in the quantity between clients. However, this partition shows a difference between FedAvg and HF-MAML. After tuning locally, the trained clients achieve a significantly lower standard deviation than under FedAvg. Interestingly, we do not see this effect for the clients added later.

However, we note that some of these test clients, particularly 17 and 18, possess some of the largest local datasets in our evaluation. On average, their variance is also lower than that of the clients involved in training.

Finally, we noted the fairness results for the attribute-based partition in Table 7.5.6. Here, we observe the largest difference between FedAvg and HF-MAML. The difference is small but significant for the 15 trained clients before tuning. After tuning, the standard deviation of HF-MAML becomes smaller, and the difference between FedAvg and HF-MAML becomes even larger. Under feature skew, HF-MAML seems to achieve the fairest results.

Finally, to better evaluate why our HF-MAML-based approach achieved fairer results, we compared the per-client performance for both FedAvg and HF-MAML in Figure 7.5.3. We observed that HF-MAML achieved a higher TAR@FAR 0.1 for all clients. However, the weakest clients under FedAvg achieved the greatest improvement percentage-wise with HF-MAML. This indicates that HF-MAML gives fairer results primarily by improving the performance of the weakest clients.

Table 7.5.5: Standard deviation of client's evaluation scores for the lognormal partition dataset. Bold implies the best result.

	Train	Test
	Untuned	
FedAvg - Global	**0.121** ± 0.022	0.005 ± 0.009
FedAvg - Local	0.129 ± 0.018	0.006 ± 0.012
FedAvg - Regularization	0.145 ± 0.020	0.008 ± 0.008
HF-MAML - Global	0.134 ± 0.016	0.006 ± 0.008
HF-MAML - Local	0.124 ± 0.021	**0.004** ± 0.009
HF-MAML - Regularization	0.127 ± 0.016	0.013 ± 0.008
	Tuned	
FedAvg - Global	0.115 ± 0.026	0.039 ± 0.012
FedAvg - Local	0.114 ± 0.019	0.029 ± 0.011
FedAvg - Regularization	0.127 ± 0.024	**0.021** ± 0.013
HF-MAML - Global	**0.088** ± 0.028	0.043 ± 0.016
HF-MAML - Local	0.105 ± 0.027	0.039 ± 0.010
HF-MAML - Regularization	0.107 ± 0.022	0.021 ± 0.010

Table 7.5.6: Standard deviation of client's evaluation scores for the attribute-based partition dataset. Bold implies the best result.

	Train	Test
	Untuned	
FedAvg - Global	0.098 ± 0.009	0.070 ± 0.020
FedAvg - Local	0.099 ± 0.008	0.079 ± 0.013
FedAvg - Regularization	0.103 ± 0.010	0.070 ± 0.013
HF-MAML - Global	0.088 ± 0.013	0.070 ± 0.013
HF-MAML - Local	**0.083** ± 0.009	**0.060** ± 0.013
HF-MAML - Regularization	0.095 ± 0.006	0.063 ± 0.013
	Tuned	
FedAvg - Global	0.078 ± 0.009	0.080 ± 0.014
FedAvg - Local	0.093 ± 0.010	0.068 ± 0.013
FedAvg - Regularization	0.095 ± 0.008	0.072 ± 0.008
HF-MAML - Global	**0.063** ± 0.011	0.069 ± 0.015
HF-MAML - Local	0.074 ± 0.007	**0.062** ± 0.011
HF-MAML - Regularization	0.081 ± 0.008	0.065 ± 0.010

Note that the last 5 clients were not seen during training, which implies that HF-MAML is a better alternative when new clients also want a personalized model.

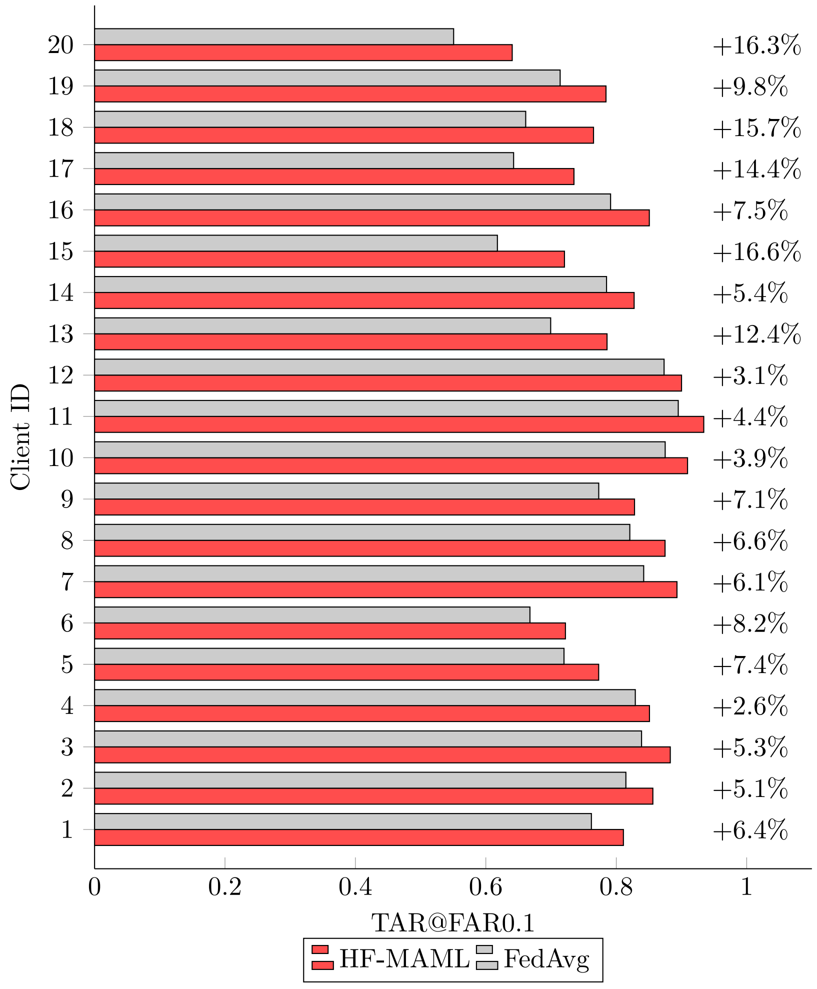


Figure 7.5.3: The mean TAR@FAR0.1 per client for FedAvg and HFMAML using the local approach on the attributes partition. Percentage improvement by HFMAML is presented at the end of the bar. Clients with weak performance on average have a greater improvement with HFMAML.

7.6 Discussion and Conclusion

In general, our results indicate that the effectiveness of HF-MAML varies depending on the level and type of data heterogeneity. Under the equal partition, HF-MAML was similar to FedAvg. Thus, it is hard to justify the additional cost and complexity of approximating the second derivative in this scenario. However, if we assume that there is some form of data heterogeneity, HF-MAML is beneficial compared to FedAvg. We observe that, especially after tuning locally, HF-MAML achieves better local TAR@FAR scores and lower variance between clients compared to FedAvg.

When it comes to embedding regularization, we observe that it helps to transfer the model to new clients. By not allowing the model to drift too much, we avoid a model that is too specific to a set of clients and, thus, transfers better to new clients. This improvement is not retained under local tuning, so regularization is primarily interesting when local tuning is not an option.

In conclusion, we propose using HF-MAML in the context of FFR. We proposed new data partitions based on the CelebA dataset that help evaluate FR in a data-heterogeneous setting and demonstrated that HF-MAML can add value in such settings. The limitations of the work entail the limited performance of our approach. By not including a global dataset, our work is unable to reach practically useful TAR@FAR scores. Our work is intended as an investigation of whether HF-MAML can be useful in FFR and should provide a stepping stone in cases where a global dataset is no longer feasible. Future work should also look at applying our approach to more datasets with different types of data heterogeneity.

CHAPTER 8

Discussion

In summary, this thesis discussed six works on different aspects of DL within the broader domain of digital communications. These works were divided into two parts. The first part addressed physical layer challenges and the second part applied layer challenges. A common denonimator throughout every chapter is the use of DL. First, a short discussion of the points per chapter is provided.

In Chapter 2, an issue of modularity was addressed. Traditional receivers are generally fully modular, an important trait in ensuring that communication systems are practical. A naively implemented DNN receiver loses modularity. This chapter demonstrated that it is possible to reintroduce modularity into a DNN receiver. Using the structure of both constellations and the DNN, a module was introduced that allows mapping independence and multiple constellations. This chapter addresses part of a wider issue in adopting deep learning techniques. Networking systems require modularity and backwards compatibility for adoption. This work demonstrates that there are custom solutions that respect both the data-driven principle and modularity.

In Chapter 3, a different aspect of the effective application of deep learning for the physical layer was presented. This chapter used principles from geometric deep learning to improve the efficiency of deep neural receivers. It differs subtly from the chapter before it. Reducing parameter count is a very broad issue, whereas reintroducing a specific modularity is quite specific. This chapter demonstrated instead that many of the findings in computer vision and NLP can be translated to physical layer communications. In this case, the incorporation of inductive biases into a network achieved a significant parameter reduction. It is thus prudent to be aware of the broader field of DL to be able to translate this to physical layer communications. However, identifying the correct inductive bias required expert knowledge of physical layer communications.

In Chapter 4, an exploratory work on contrastive learning for UATR was presented. This work shows the possibility of building an underwater acoustic model without explicit supervision. This is a highly interesting topic for effective applications of deep learning for underwater communications as well. It demonstrates the effectiveness of combining techniques commonly used in CV in domains tangential to the physical layer.

In Chapter 5, metric learning was used to further exploit a known vulnerability in MPEG-DASH. This work demonstrates that this attack can be made more practical by building a triplet loss model. The resulting model generalizes well to other settings with a small number of additional samples from these settings. This work demonstrates that adjusting the DL modeling objective can directly result in a more practical network system.

In Chapter 6, a method for neural image compression was proposed to perform further inference time optimizations. The concept of neural image compression in networking is underexplored, despite compression being a key component of the presentation layer. This line of work indicates that certain presentation layer tasks can also be adopted by DL. By improving the overall compression factor, the work demonstrates improved effectiveness for eventual adoption in future networking systems.

In Chapter 7, a federated learning method for fair training in FR was presented. Ensuring that connections are private is already an important task in networking. Considering FL as part of the networking stack to consider privacy, fairness, and efficient model training may be prudent. It may prove to be an important step in the integration of DL into the networking stack.

8.1 Research Questions

Based on the content chapters, **RQ1** and **RQ2** are addressed.

RQ1: *How can deep learning techniques be effectively applied across different parts of the networking stack?*

Note that we only addressed parts of this question, as the question is quite broad. The following three common components were identified based on the content of this thesis.

The need for modularity A core issue remains in making DL systems practical enough for networking systems. One component in making networking systems is modularity. Both chapters 2 and 5 addressed issues related to modularity. The naive inclusion of end-to-end learning introduces issues with e.g. backward compatibility. These chapters show that modularity can be reintroduced by constructing DNNs in such a way that certain components can be swapped depending on need. These works show that modularity and end-to-end learning are not mutually exclusive.

Translating DL improvements to communications Another common theme throughout the chapters is the incorporation of improvements in other domains of DL. Advancements in one technical subdomain of DL are often applicable throughout other subdomains of DL as well. Chapter 3 demonstrates how the inclusion of inductive biases, as originally proposed for CV, also enables more efficient deep receivers. It is important to note that these translations are not necessarily straightforward; identifying inductive biases for deep receivers requires expert knowledge. This chapter also introduced improvements to deep neural receivers based on improvements to CV models. Certain model improvements are translated across domains, the most important example being residual connections [77]. Improvements in DL techniques are also demonstrated in Chapter 4. The concept of unsupervised learning is not new. However, the ability to construct unsupervised models that match the ability of supervised models is quite recent. This improvement enables new possibilities for UATR and potentially underwater communications. Both chapters show how advances in other areas of DL enable solving issues with DL for networking.

A broader view of DL for networking Arguably the most important point is the conclusion that a large portion of DL research directly applies to networking. Broadly speaking, advances in e.g. LLMs build on and require advances in networking. For instances, although mainly approached from the area of CV, developing better compression techniques is in many ways a networking task. Chapter 6 proposed a practical extension to neural image compression. By extension, this work makes neural compression for networking more practical. Similarly, the work on FL as in Chapter 7 is by extension work on networking. When distributed training techniques such as FL become commonplace, it is imperative that the network scale along with it. Work on efficient FL directly aids in this task, despite not necessarily being considered as a networking domain. Potentially, these lines of work could converge with the work that is being done on AI for networking.

The overall answer to this question is therefore nuanced, but boils down to the importance of a nice interplay between DL and communications. A practitioner of DL for communications should be aware of advances within the broader domain of DL. These are often directly useful in the domain of communications. However, there is a clear gap for the inclusion of expert knowledge in DL methods to allow them to better fit communications systems. Naive implementation of better DL techniques for CV can result in better models, as demonstrated with the ConvNeXt adaptation in Chapter 3. However, expert knowledge of both domains is required to achieve the full potential of DL for digital communications.

RQ2: *What are the advantages and challenges of implementing deep learning in the physical layer versus the application layer?*

Addressing deep learning challenges for both the physical layer and the application layer provides a unique perspective. There are notable differences in approaches, but also clear similarities.

Difference: data modality An important difference to consider is the data modality for both layers. The physical layer, in our experience, is a more natural fit to consider implementing DL. Although the amount of data is large in the application layer, the packets are well defined at that point. Algorithms exist or are possible that do not require the automatic feature extraction of DL. The features are often possible to extract directly from the data. The method in Chapter 5 is only useful when the assumption is made that the data stream is encrypted.

Physical layer tasks operate primarily on signals where the data is no more than a bit stream. Such tasks align more closely with the domains in which DL traditionally achieved success. Something that is taken for granted in the current physical layer literature is that deep receivers achieve better gains than, e.g. a LS estimate. The transition to DL is in some sense a transition from model-based assumptions to data-based assumptions. Instead of entirely modeling the physical phenomena based on a set of assumptions, the assumptions are automatically learned based on the provided data.

Similarity: solution avenue While the data modality differs, there are great similarities in the work for both layers. Every chapter uses some form of CNN as a solution avenue. This is despite the fact that each chapter addresses different, sometimes drastically different topics. Although the choice for a CNN is flexible, all problems share some form of time-domain challenge. This property accelerates research for both layers, as advancements in one area are often (partially) applicable. Thus, it also allows for the possibility of building joint latent spaces across layers.

It is important to relate this last statement to the concept of end-to-end learning. The work of [84] proposes complete end-to-end physical layer systems. If it is possible to construct latents in a similar fashion for both layers, this idea could be extended further across the layers. For example, one could think of a process similar to encapsulation in networking. However, instead of adding purely headers, latents could be added of end-to-end learning systems. Although modularity becomes an issue, certain task-specific loads may prefer potential performance improvements over modularity.

Thus, it becomes clear that there is a clear benefit in considering research of DL for both the application layer and the physical layer. Although research on DL for the physical layer is necessary, it has also become clear that potential lies in combining further layers with the physical layer. In particular, a broader view of the application layer provides greater potential than a traditional view of the application layer. This becomes especially true when considering the increased networking needs of CV foundation models and LLMs such as ChatGPT, which are also trainable end-to-end. A broader look at the application layer with fully end-to-end systems may prove to be advantageous in the future.

8.2 Open Problems

While this thesis demonstrates there is a lot of potential in DL for networking, there are a few open problems that must be addressed before considering adoption.

Energy efficiency and speed A major concern that this thesis only partially addresses in Chapter 3 is the high compute requirements of DL. High compute demands result in high energy requirements and lack of speeds. It is good to note that recent advances in software [83] have enabled real-time physical layer network links that only use DNNs. Hardware advancements will likely also enable approaches that are not currently viable.

Nevertheless, a lot more work needs to be done to enable systems that are practical. While a transition to fully end-to-end systems is highly interesting as a DL practitioner, it is also good to question to what extent this is useful. Identifying elements within communications that are better done with DL, but also how they would integrate with existing systems, is a key problem for the future to ensure that energy and speed issues are manageable.

Backward compatibility How to integrate DL systems in existing infrastructure is a major point. Given that the problems of energy efficiency and speed have been addressed, adoption would still be limited to new custom networking solutions. There is a lot of legacy hardware that AI-based systems need to operate with nicely. This constrains development of DNN-based receivers, and without proper care, may see adoption entirely omitted or limited to niche functionalities. This thesis has presented some work on modularity in Chapters 2 and 5 address some modularity issues but yet more work remains to be done.

Practitioner skill maturity Both of the previous points are related to knowledge and skill as well. The domains of networking and telecommunications consist of many extensive and well-established fields. While DL is relatively new, it has rapidly advanced over the years. A key factor in this advancement has been that many researchers have chosen to adopt DL as their main topic of research. These resulted in transformative changes to the CV and NLP domains. Both of these domains are now almost entirely DL-based and form much of the basis of modern DL theory books. This thesis in part demonstrates the potential of DL modeling for networking, but also the challenges in the modeling aspects. Interdisciplinary research between communications and DL practitioners has already enabled incredible progress. This thesis posits, however, that a group of researchers focused purely on DL for communications is necessary to fully unlock the potential.

8.3 Concluding Remarks

This thesis presented a variety of works on DL for communications. The set of work consisted of two parts, the first addressing physical layer challenges and the second a broad range of application layer challenges. Through these works, two research questions on the applicability of DL in communications and networking were addressed. This thesis demonstrated the usefulness of DL in communications and explored shared challenges across networking layers.

DL for communications has proven an interesting topic so far and will likely remain so in the near future. Although it is unclear whether it is viable, it is definitely possible to treat a network link as an end-to-end system. A lot of exciting work remains to be done in realizing the full potential of DL for networking and communications.

Publications

[1] A. Gansekoele et al. ‘A Machine Learning Approach for Simultaneous Demapping of QAM and APSK Constellations’. In: *2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*. 2024, pages 13–18.

[2] A. Gansekoele et al. ‘Joint Demapping of QAM and APSK Constellations Using Machine Learning’. In: *IEEE Open Journal of the Communications Society* 6 (2025), pages 1695–1709.

[3] A. Gansekoele et al. ‘Meta-Learning for Federated Face Recognition in Imbalanced Data Regimes’. In: *2024 IEEE International Conference on Federated Learning Technologies and Applications (FLTA)*. 2024.

[4] A. Gansekoele et al. ‘Relative Phase Equivariant Deep Neural Systems for Physical Layer Communications’. In: *Transactions on Machine Learning Research* (2025).

[5] A. Gansekoele et al. ‘Unveiling the Potential: Harnessing Deep Metric Learning to Circumvent Video Streaming Encryption’. In: *2023 IEEE/WIC International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*. 2023, pages 163–170.

[6] H. Hummel et al. ‘The Computation of Generalized Embeddings for Underwater Acoustic Target Recognition using Contrastive Learning’. Under review.

[7] Y. Perugachi-Diaz et al. ‘Robustly overfitting latents for flexible neural image compression’. To appear in the 2024 Neural Information Processing Systems (Neurips) proceedings. 2024.

Bibliography

[8] 3GPP. *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Requirements for Evolved UTRA (E-UTRA) and Evolved UTRAN (E-UTRAN) (Release 7)*. Technical report TR 25.913 V7.3.0 (2006-03). 3GPP, 2006.

[9] D. Aggarwal et al. ‘FedFace: Collaborative Learning of Face Recognition Model’. In: *2021 IEEE International Joint Conference on Biometrics (IJCB)*. 2021, pages 1–8.

[10] E. Agustsson et al. ‘Scale-space flow for end-to-end optimized video compression’. In: *IEEE conference on Computer Vision and Pattern Recognition*. 2020.

[11] E. Agustsson et al. ‘Soft-to-hard vector quantization for end-to-end learning compressible representations’. In: *Advances in neural information processing systems* 30 (2017).

[12] K. Åhlander and H. Munthe-Kaas. ‘Applications of the Generalized Fourier Transform in Numerical Linear Algebra’. In: *BIT Numerical Mathematics* 45.4 (Dec. 2005), pages 819–850.

[13] F. Ait Aoudia and J. Hoydis. ‘End-to-End Learning for OFDM: From Neural Receivers to Pilotless Communication’. In: *IEEE Transactions on Wireless Communications* 21.2 (2022), pages 1049–1063.

[14] F. Ait Aoudia and J. Hoydis. ‘Waveform Learning for Next-Generation Wireless Communication Systems’. In: *IEEE Transactions on Communications* 70.6 (June 2022), pages 3804–3817. (Visited on 14/06/2024).

[15] S. H. Alkarni. ‘Statistical applications for equivariant matrices’. In: *International Journal of Mathematics and Mathematical Sciences* 25.1 (2001), page 303787.

[16] A. Alkhateeb. ‘DeepMIMO: A Generic Deep Learning Dataset for Millimeter Wave and Massive MIMO Applications’. In: *Proc. of Information*

Theory and Applications Workshop (ITA). San Diego, CA, Feb. 2019, pages 1–8.

[17] R. Amin et al. ‘A Survey on Machine Learning Techniques for Routing Optimization in SDN’. In: *IEEE Access* 9 (2021), pages 104582–104611.

[18] N. Asuni and A. Giachetti. ‘TESTIMAGES: A large-scale archive for testing visual devices and basic image processing algorithms (SAMPLING 1200 RGB set)’. In: *STAG: Smart Tools and Apps for Graphics*. 2014.

[19] C. Aytekin et al. ‘Block-optimized variable bit rate neural image compression’. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pages 2551–2554.

[20] S. Bae et al. ‘Watching the Watchers: Practical Video Identification Attack in LTE Networks’. In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pages 1307–1324.

[21] M. Balcilar et al. ‘Latent-Shift: Gradient of Entropy Helps Neural Codecs’. In: *2023 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2023, pages 920–924.

[22] E. Balevi et al. ‘Massive MIMO Channel Estimation with an Untrained Deep Neural Network’. In: *IEEE Transactions on Wireless Communications* 19.3 (2020), pages 2079–2090.

[23] J. Ballé et al. ‘End-to-end optimized image compression’. In: *International Conference on Learning Representations* (2017).

[24] J. Ballé et al. ‘Variational image compression with a scale hyperprior’. In: *arXiv preprint arXiv:1802.01436* (2018).

[25] A. Bardes et al. ‘Vicreg: Variance-invariance-covariance regularization for self-supervised learning’. In: *arXiv preprint arXiv:2105.04906* (2021).

[26] J. Bégaint et al. ‘CompressAI: a PyTorch library and evaluation platform for end-to-end compression research’. In: *arXiv preprint arXiv:2011.03029* (2020).

[27] F. Bellard. *BPG Specification*. (accessed June 3, 2020). 2014.

[28] Y. Bengio et al. ‘Estimating or propagating gradients through stochastic neurons for conditional computation’. In: *arXiv preprint arXiv:1308.3432* (2013).

[29] G. Bjontegaard. *Calculation of average PSNR differences between RD-curves*. VCEG-M33.

[30] G. Bokman et al. ‘ZZ-Net: A Universal Rotation Equivariant Architecture for 2D Point Clouds’. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2022), pages 10966–10975. (Visited on 24/07/2024).

[31] M. M. Bronstein et al. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. 2021.

[32] S. Caldas et al. ‘LEAF: A Benchmark for Federated Settings’. In: *CoRR* abs/1812.01097 (2018).

[33] S. Cammerer et al. ‘A Neural Receiver for 5G NR Multi-User MIMO’. In: *2023 IEEE Globecom Workshops (GC Wkshps)*. Dec. 2023, pages 329–334. (Visited on 17/06/2024).

[34] S. Cammerer et al. ‘Graph Neural Networks for Channel Decoding’. In: *2022 IEEE Globecom Workshops (GC Wkshps)*. Dec. 2022, pages 486–491. (Visited on 14/06/2024).

[35] J. Campos et al. ‘Content adaptive optimization for neural image compression’. In: *arXiv preprint arXiv:1906.01223* (2019).

[36] O. N. Canada. *Dataportal*. https://data.oceannetworks.ca/home. Accessed: 1 May 2023. 2007.

[37] R. Q. Charles et al. ‘PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation’. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017), pages 77–85. (Visited on 24/07/2024).

[38] F. Chen et al. *Federated Meta-Learning with Fast Convergence and Efficient Communication*. 2019. (Visited on 23/01/2023).

[39] H. Chen et al. ‘Equivariant Point Network for 3D Point Cloud Analysis’. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Nashville, TN, USA: IEEE, June 2021, pages 14509–14518. (Visited on 24/07/2024).

[40] T. Chen et al. ‘A simple framework for contrastive learning of visual representations’. In: *International conference on machine learning*. PMLR. 2020, pages 1597–1607.

[41] Z. Cheng et al. ‘Learned Image Compression with Discretized Gaussian Mixture Likelihoods and Attention Modules’. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[42] T. Cohen and M. Welling. ‘Group Equivariant Convolutional Networks’. In: *Proceedings of The 33rd International Conference on Machine Learning*. PMLR, June 2016, pages 2990–2999. (Visited on 24/07/2024).

[43] C. Cremer et al. ‘Inference suboptimality in variational autoencoders’. In: *International Conference on Machine Learning*. PMLR. 2018, pages 1078–1086.

[44] A. Dainotti et al. ‘Issues and future directions in traffic classification’. In: *IEEE Network* 26.1 (2012), pages 35–40.

[45] J. Deng et al. ‘Imagenet: A large-scale hierarchical image database’. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009.

[46] K. L. Dias et al. ‘An innovative approach for real-time network traffic classification’. In: *Computer Networks* 158 (2019), pages 143–157.

[47] S. Dörner et al. ‘Deep Learning-Based Communication Over the Air’. In: *IEEE Journal of Selected Topics in Signal Processing* 12.1 (Feb. 2018), pages 132–143. (Visited on 16/06/2023).

[48] S. Dorner et al. ‘Deep Learning Based Communication Over the Air’. In: *IEEE Journal of Selected Topics in Signal Processing* 12.1 (Feb. 2018), pages 132–143. (Visited on 14/08/2023).

[49] R. Dubin et al. ‘I Know What You Saw Last Minute—Encrypted HTTP Adaptive Video Streaming Title Classification’. In: *IEEE Transactions on Information Forensics and Security* 12.12 (2017), pages 3039–3049.

[50] E. Dupont et al. 'COIN: COmpression with Implicit Neural representations'. In: *CoRR* abs/2103.03123 (2021).

[51] N. Dym and H. Maron. 'On the Universality of Rotation Equivariant Point Cloud Networks'. In: *ArXiv* (Oct. 2020). (Visited on 24/07/2024).

[52] A. El-Nouby et al. 'Image Compression with Product Quantized Masked Image Modeling'. In: *Trans. Mach. Learn. Res.* 2023 (2023).

[53] C. Esteves et al. 'Equivariant Multi-View Networks'. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (Oct. 2019), pages 1568–1577. (Visited on 24/07/2024).

[54] A. Fallah et al. 'On the convergence theory of gradient-based model-agnostic meta-learning algorithms'. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pages 1082–1092.

[55] A. Fallah et al. 'Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach'. In: *Advances in Neural Information Processing Systems* 33 (2020), pages 3557–3568.

[56] Fayçal Ait Aoudia et al. 'Model-Free Training of End-to-End Communication Systems'. In: *IEEE Journal on Selected Areas in Communications* 37.11 (Aug. 2019), pages 2503–2516.

[57] A. Felix et al. 'OFDM-Autoencoder for End-to-End Learning of Communications Systems'. In: *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. June 2018, pages 1–5.

[58] C. Finn et al. 'Model-agnostic meta-learning for fast adaptation of deep networks'. In: *International conference on machine learning*. PMLR. 2017, pages 1126–1135.

[59] E. Fonseca et al. 'Unsupervised contrastive learning of sound event representations'. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, pages 371–375.

[60] A. Gansekoele et al. 'A Machine Learning Approach for Simultaneous Demapping of QAM and APSK Constellations'. In: *2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*. 2024, pages 13–18.

[61] C. Gao et al. 'Flexible neural image compression via code editing'. In: *Advances in Neural Information Processing Systems* 35 (2022), pages 12184–12196.

[62] J. Gao et al. 'Deep Learning for Spectrum Sensing'. In: *IEEE Wireless Communications Letters* 8.6 (2019), pages 1727–1730.

[63] I. Goodfellow et al. 'Generative adversarial nets'. In: *Advances in neural information processing systems* 27 (2014).

[64] M. Goutay et al. 'Machine Learning for MU-MIMO Receive Processing in OFDM Systems'. In: *IEEE Journal on Selected Areas in Communications* 39.8 (Aug. 2021), pages 2318–2332. (Visited on 14/06/2024).

[65] J.-B. Grill et al. 'Bootstrap your own latent-a new approach to self-supervised learning'. In: *Advances in neural information processing systems* 33 (2020), pages 21271–21284.

[66] J. Gu et al. ‘Walls Have Ears: Traffic-based Side-channel Attack in Video Streaming’. In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, pages 1538–1546.

[67] A. Gulati et al. ‘Conformer: Convolution-augmented transformer for speech recognition’. In: *arXiv preprint arXiv:2005.08100* (2020).

[68] T. Guo et al. ‘Variable rate image compression with content adaptive optimization’. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pages 122–123.

[69] Y. Guo et al. ‘Ms-celeb-1m: A dataset and benchmark for large-scale face recognition’. In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14*. Springer. 2016, pages 87–102.

[70] Z. Guo et al. ‘Soft then Hard: Rethinking the Quantization in Neural Image Compression’. In: *Proceedings of the 38th International Conference on Machine Learning*. Edited by M. Meila and T. Zhang. Volume 139. Proceedings of Machine Learning Research. PMLR, July 2021, pages 3920–3929.

[71] A. Habibian et al. ‘Video compression with rate-distortion autoencoders’. In: *IEEE International Conference on Computer Vision*. 2019.

[72] Q. Hamard et al. ‘A deep learning model for detecting and classifying multiple marine mammal species from passive acoustic data’. In: *Ecological Informatics* (2024), page 102906.

[73] E. Hand and R. Chellappa. ‘Attributes for improved attributes: A multi-task network utilizing implicit and explicit relationships for facial attribute classification’. In: *Proceedings of the AAAI conference on artificial intelligence*. Volume 31. 1. 2017.

[74] S. Hanna et al. ‘Signal Processing-Based Deep Learning for Blind Symbol Decoding and Modulation Classification’. In: *IEEE Journal on Selected Areas in Communications* 40.1 (2022), pages 82–96.

[75] D. He et al. ‘ELIC: Efficient Learned Image Compression with Unevenly Grouped Space-Channel Contextual Adaptive Coding’. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*. IEEE, 2022, pages 5708–5717.

[76] H. He et al. ‘Message Passing Meets Graph Neural Networks: A New Paradigm for Massive MIMO Systems’. In: *IEEE Transactions on Wireless Communications* 23.5 (May 2024), pages 4709–4723. (Visited on 14/06/2024).

[77] K. He et al. ‘Deep residual learning for image recognition’. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pages 770–778.

[78] K. He et al. ‘Momentum contrast for unsupervised visual representation learning’. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pages 9729–9738.

[79] D. Hendrycks and K. Gimpel. ‘A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks’. In: *International Conference on Learning Representations*. 2017.

[80] D. Hendrycks et al. ‘Deep Anomaly Detection with Outlier Exposure’. In: *International Conference on Learning Representations*. 2019.

[81] A. Hermans et al. ‘In Defense of the Triplet Loss for Person Re-Identification’. In: *ArXiv* abs/1703.07737 (2017).

[82] M. Honkala et al. ‘DeepRx: Fully Convolutional Deep Learning Receiver’. In: *IEEE Transactions on Wireless Communications* 20.6 (June 2021), pages 3925–3940. (Visited on 04/07/2023).

[83] J. Hoydis et al. ‘Sionna: An Open-Source Library for Next-Generation Physical Layer Research’. In: *arXiv preprint* (Mar. 2022).

[84] J. Hoydis et al. ‘Toward a 6G AI-native Air Interface’. In: *IEEE Communications Magazine* 59.5 (2021), pages 76–81.

[85] G. B. Huang et al. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. Technical report 07-49. University of Massachusetts, Amherst, Oct. 2007.

[86] H. I. Hummel et al. ‘A survey on machine learning in ship radiated noise’. In: *Ocean Engineering* 298 (2024), page 117252.

[87] J. M. J. Huttunen et al. ‘DeepTx: Deep Learning Beamforming With Channel Prediction’. In: *IEEE Transactions on Wireless Communications* 22.3 (Mar. 2023), pages 1855–1867. (Visited on 18/06/2024).

[88] S. Ioffe and C. Szegedy. ‘Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift’. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France: JMLR.org, 2015, pages 448–456.

[89] M. Irfan et al. ‘DeepShip: An underwater acoustic benchmark dataset and a separable convolution based autoencoder for classification’. In: *Expert Systems with Applications* 183 (2021), page 115270.

[90] E. Jang et al. ‘Categorical reparameterization with gumbel-softmax’. In: *arXiv preprint arXiv:1611.01144* (2016).

[91] Y. Jiang et al. *Improving Federated Learning Personalization via Model Agnostic Meta Learning*. 27th Sept. 2019. (Visited on 13/01/2023).

[92] J. Jumper et al. ‘Highly accurate protein structure prediction with AlphaFold’. In: *Nature* 596.7873 (Aug. 2021), pages 583–589.

[93] S. Kahl et al. ‘BirdNET: A deep learning solution for avian diversity monitoring’. In: *Ecological Informatics* 61 (2021), page 101236.

[94] P. Khosla et al. ‘Supervised contrastive learning’. In: *Advances in neural information processing systems* 33 (2020), pages 18661–18673.

[95] D. Kingma and J. Ba. ‘Adam: A Method for Stochastic Optimization’. In: *International Conference on Learning Representations (ICLR)*. San Diega, CA, USA, 2015.

[96] D. P. Kingma and J. Ba. ‘Adam: A Method for Stochastic Optimization’. In: *ICLR (Poster)*. 2015.

[97] D. P. Kingma and M. Welling. ‘Auto-encoding variational bayes’. In: *arXiv preprint arXiv:1312.6114* (2013).

[98] E. Kodak. *Kodak Lossless True Color Image Suite (PhotoCD PCD0992)*.

[99] D. Korpi et al. ‘Deep Learning-Based Pilotless Spatial Multiplexing’. In: *2023 57th Asilomar Conference on Signals, Systems, and Computers.* Oct. 2023, pages 1025–1029. (Visited on 17/06/2024).

[100] D. Korpi et al. ‘DeepRx MIMO: Convolutional MIMO Detection with Learned Multiplicative Transformations’. In: *ICC 2021 - IEEE International Conference on Communications.* 2021, pages 1–7.

[101] A. Krizhevsky et al. ‘ImageNet Classification with Deep Convolutional Neural Networks’. In: *Advances in Neural Information Processing Systems.* Edited by F. Pereira et al. Volume 25. Curran Associates, Inc., 2012.

[102] Y. Lecun et al. ‘Gradient-based learning applied to document recognition’. In: *Proceedings of the IEEE* 86.11 (1998), pages 2278–2324.

[103] Y. LeCun et al. ‘Deep learning’. In: *Nature* 521.7553 (May 2015), pages 436–444.

[104] J. Lee et al. ‘Context-adaptive Entropy Model for End-to-end Optimized Image Compression’. In: *the 7th Int. Conf. on Learning Representations.* May 2019.

[105] K. Lee et al. ‘Training Confidence-calibrated Classifiers for Detecting Out-of-Distribution Samples’. In: *International Conference on Learning Representations.* 2018.

[106] B. Li et al. ‘Deep learning-based image compression with trellis coded quantization’. In: *2020 Data Compression Conference (DCC).* IEEE. 2020, pages 13–22.

[107] J. Li et al. ‘Discrete Rotation Equivariance for Point Cloud Recognition’. In: *2019 International Conference on Robotics and Automation (ICRA)* (May 2019), pages 7269–7275. (Visited on 24/07/2024).

[108] T. Li et al. ‘Ditto: Fair and robust federated learning through personalization’. In: *International conference on machine learning.* PMLR. 2021, pages 6357–6368.

[109] T. Li et al. ‘Fair Resource Allocation in Federated Learning’. In: *8th International Conference on Learning Representations.* 2020.

[110] T. Li et al. ‘Federated optimization in heterogeneous networks’. In: *Proceedings of Machine learning and systems* 2 (2020), pages 429–450.

[111] Y. Li et al. ‘Deep Content: Unveiling Video Streaming Content from Encrypted WiFi Traffic’. In: *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA).* 2018, pages 1–8.

[112] A. Licciardi and D. Carbone. ‘WhaleNet: a Novel Deep Learning Architecture for Marine Mammals Vocalizations on Watkins Marine Mammal Sound Database’. In: *IEEE Access* (2024).

[113] L. Lin et al. ‘Underwater Passive Target Recognition Based on Self-Supervised Contrastive Learning’. In: *2024 3rd International Conference on Artificial Intelligence, Internet of Things and Cloud Computing Technology (AIoTC).* IEEE. 2024, pages 375–380.

[114] X. Lin et al. ‘A Deep Convolutional Network Demodulator for Mixed Signals with Different Modulation Types’. In: *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Comput-*

ing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech). Orlando, FL: IEEE, Nov. 2017, pages 893–896. (Visited on 16/06/2023).

[115] C.-T. Liu et al. ‘FedFR: Joint Optimization Federated Framework for Generic and Personalized Face Recognition’. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.2 (28th June 2022). Number: 2, pages 1656–1664.

[116] Z. Liu et al. ‘A convnet for the 2020s’. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 2022, pages 11976–11986.

[117] Z. Liu et al. ‘Deep Learning Face Attributes in the Wild’. In: *Proceedings of International Conference on Computer Vision (ICCV).* Dec. 2015.

[118] I. Loshchilov and F. Hutter. ‘Decoupled Weight Decay Regularization’. In: *International Conference on Learning Representations (ICLR).* 2017.

[119] I. Loshchilov and F. Hutter. ‘Decoupled Weight Decay Regularization’. In: *International Conference on Learning Representations.* 2019.

[120] G. Lu et al. ‘Content adaptive and error propagation aware deep video compression’. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16.* Springer. 2020, pages 456–472.

[121] G. Lu et al. ‘Dvc: An end-to-end deep video compression framework’. In: *IEEE conference on Computer Vision and Pattern Recognition.* 2019.

[122] B. Maze et al. ‘IARPA Janus Benchmark - C: Face Dataset and Protocol’. In: *2018 International Conference on Biometrics (ICB).* Gold Coast, QLD: IEEE, Feb. 2018, pages 158–165.

[123] B. McMahan et al. ‘Communication-efficient learning of deep networks from decentralized data’. In: *Artificial intelligence and statistics.* PMLR. 2017, pages 1273–1282.

[124] D. Minnen et al. ‘Joint autoregressive and hierarchical priors for learned image compression’. In: *Advances in neural information processing systems* 31 (2018).

[125] A. S. Mohammad et al. ‘Demodulation of Faded Wireless Signals Using Deep Convolutional Neural Networks’. In: *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC).* Las Vegas, NV: IEEE, Jan. 2018, pages 969–975. (Visited on 20/06/2023).

[126] A. Morello and V. Mignone. ‘DVB-S2: The Second Generation Standard for Satellite Broad-Band Services’. In: *Proceedings of the IEEE* 94.1 (2006), pages 210–227.

[127] N. Müller et al. ‘Navigating the Depths: A Comprehensive Survey of Deep Learning for Passive Underwater Acoustic Target Recognition’. In: *IEEE Access* (2024).

[128] A. Nech and I. Kemelmacher-Shlizerman. ‘Level playing field for million scale face recognition’. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2017, pages 7044–7053.

[129] Neev Samuel et al. ‘Learning to Detect’. In: *IEEE Transactions on Signal Processing* 67.10 (May 2019), pages 2554–2564.

[130] L. Nie et al. ‘A contrastive-learning-based method for the few-shot identification of ship-radiated noises’. In: *Journal of Marine Science and Engineering* 11.4 (2023), page 782.

[131] D. Niizumi et al. ‘BYOL for audio: Exploring pre-trained general-purpose audio representations’. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 31 (2022), pages 137–151.

[132] Y. Niu and W. Deng. *Federated Learning for Face Recognition with Gradient Correction.* 14th Dec. 2021. (Visited on 01/02/2023).

[133] T. O’Shea and J. Hoydis. ‘An Introduction to Deep Learning for the Physical Layer’. In: *IEEE Transactions on Cognitive Communications and Networking* 3.4 (2017), pages 563–575.

[134] Ö. Özdogan et al. ‘Performance of cell-free massive MIMO with Rician fading and phase shifts’. In: *IEEE Transactions on Wireless Communications* 18.11 (2019), pages 5299–5315.

[135] B. Ozpoyraz et al. ‘Deep Learning-Aided 6G Wireless Networks: A Comprehensive Survey of Revolutionary PHY Architectures’. In: *IEEE Open Journal of the Communications Society* 3 (2022), pages 1749–1809.

[136] J. Pihlajasalo et al. ‘Deep Learning OFDM Receivers for Improved Power Efficiency and Coverage’. In: *IEEE Transactions on Wireless Communications* 22.8 (Aug. 2023), pages 5518–5535. (Visited on 14/06/2024).

[137] T. Raviv and N. Shlezinger. ‘Data Augmentation for Deep Receivers’. In: *IEEE Transactions on Wireless Communications* (2023), pages 1–1.

[138] T. Raviv et al. ‘Online Meta-Learning for Hybrid Model-Based Deep Receivers’. In: *IEEE Transactions on Wireless Communications* (2023), pages 1–1.

[139] J Redmon. ‘You only look once: Unified, real-time object detection’. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016.

[140] A. Reed and B. Klimkowski. ‘Leaky streams: Identifying variable bitrate DASH videos streamed over encrypted 802.11n connections’. In: *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC).* 2016, pages 1107–1112.

[141] A. Reed and M. Kranch. ‘Identifying HTTPS-Protected Netflix Videos in Real-Time’. In: *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy.* CODASPY ’17. Scottsdale, Arizona, USA: Association for Computing Machinery, 2017, pages 361–368.

[142] D. W. Romero and M. Hoogendoorn. ‘Co-attentive equivariant neural networks: Focusing equivariance on transformations co-occurring in data’. In: *arXiv preprint arXiv:1911.07849* (2019).

[143] D. E. Rumelhart et al. ‘Learning representations by back-propagating errors’. In: *Nature* 323.6088 (Aug. 1986), pages 533–536.

[144] O. Russakovsky et al. ‘ImageNet Large Scale Visual Recognition Challenge’. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pages 211–252.

[145] A. Saeed et al. ‘Contrastive Learning of General-Purpose Audio Representations’. In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2021, pages 3875–3879.

[146] O. Salman et al. ‘A review on machine learning–based approaches for Internet traffic classification’. In: *Annals of Telecommunications* 75.11 (2020), pages 673–710.

[147] D. Santos-Domínguez et al. ‘ShipsEar: An underwater vessel noise database’. In: *Applied Acoustics* 113 (2016), pages 64–69.

[148] L. Sayigh et al. ‘The watkins marine mammal sound database: An online, freely accessible resource’. In: *Proceedings of Meetings on Acoustics.* Volume 27. 1. AIP Publishing. 2016.

[149] F. Schroff et al. ‘FaceNet: A unified embedding for face recognition and clustering’. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2015, pages 815–823.

[150] R. Schuster et al. ‘Beauty and the Burst: Remote Identification of Encrypted Video Streams’. In: *26th USENIX Security Symposium (USENIX Security 17).* Vancouver, BC: USENIX Association, 2017, pages 1357–1374.

[151] E. Shang et al. ‘FedFR: Evaluation and Selection of Loss Functions for Federated Face Recognition’. In: *Collaborative Computing: Networking, Applications and Worksharing.* Edited by H. Gao et al. Cham: Springer Nature Switzerland, 2022, pages 95–114.

[152] C. E. Shannon. ‘Communication in the presence of noise’. In: *Proceedings of the IRE* 37.1 (1949), pages 10–21.

[153] P. Sirinam et al. ‘Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-Shot Learning’. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security.* CCS ’19. London, United Kingdom: Association for Computing Machinery, 2019, pages 1131–1148.

[154] A. Skodras et al. ‘The JPEG 2000 still image compression standard’. In: *IEEE Signal Processing Magazine* (2001).

[155] M. Soltani et al. ‘Deep Learning-Based Channel Estimation’. In: *IEEE Communications Letters* 23.4 (Apr. 2019), pages 652–655. (Visited on 14/08/2023).

[156] S. Srivastava et al. ‘Conformer-based self-supervised learning for non-speech audio tasks’. In: *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE. 2022, pages 8862–8866.

[157] B. Sun and X. Luo. ‘Underwater acoustic target recognition based on automatic feature and contrastive coding’. In: *IET Radar, Sonar & Navigation* 17.8 (2023), pages 1277–1285.

[158] A. S. Tanenbaum and D. J. Wetherall. *Computer Networks.* 5th. USA: Prentice Hall Press, 2010.

[159] L. Theis et al. ‘Lossy image compression with compressive autoencoders’. In: *arXiv preprint arXiv:1703.00395* (2017).

[160] M. Thomas et al. ‘Marine mammal species classification using convolutional neural networks and a novel acoustic representation’. In: *Machine Learning*

and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part III. Springer. 2020, pages 290–305.

[161] N. Thomas et al. 'Tensor Field Networks: Rotation- and Translation-Equivariant Neural Networks for 3D Point Clouds'. In: *ArXiv* (Feb. 2018). (Visited on 24/07/2024).

[162] S. Tian et al. 'Few-shot learning for joint model in underwater acoustic target recognition'. In: *Scientific Reports* 13.1 (2023), page 17502.

[163] G. Toderici et al. 'Full resolution image compression with recurrent neural networks'. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pages 5306–5314.

[164] G. Toderici et al. *Workshop and Challenge on Learned Image Compression (CLIC2020)*. CVPR, 2020.

[165] S. Tonekaboni et al. 'Unsupervised representation learning for time series with temporal neighborhood coding'. In: *arXiv preprint arXiv:2106.00750* (2021).

[166] S. Valenti et al. 'Reviewing Traffic Classification'. In: *Data Traffic Monitoring and Analysis: From Measurement, Classification, and Anomaly Detection to Quality of Experience*. Edited by E. Biersack et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pages 123–147.

[167] T. van Rozendaal et al. 'Overfitting for Fun and Profit: Instance-Adaptive Data Compression'. In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[168] A. Van Den Oord, O. Vinyals et al. 'Neural discrete representation learning'. In: *Advances in neural information processing systems* 30 (2017).

[169] B. Van Merriënboer et al. 'Birds, bats and beyond: Evaluating generalization in bioacoustics models'. In: *Frontiers in Bird Science* 3 (2024), page 1369756.

[170] P. Velan et al. 'A Survey of Methods for Encrypted Traffic Classification and Analysis'. In: *International Journal of Network Management* 25.5 (Sept. 2015), pages 355–374.

[171] G. K. Wallace. 'The JPEG still picture compression standard'. In: *IEEE transactions on consumer electronics* 38.1 (1992), pages xviii–xxxiv.

[172] C. Wang et al. 'Adaptive Fingerprinting: Website Fingerprinting over Few Encrypted Traffic'. In: *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*. CODASPY '21. Virtual Event, USA: Association for Computing Machinery, 2021, pages 149–160.

[173] H. Wang et al. 'Cosface: Large margin cosine loss for deep face recognition'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pages 5265–5274.

[174] H. Wang et al. 'Deep Learning for Signal Demodulation in Physical Layer Wireless Communications: Prototype Platform, Open Dataset, and Analytics'. In: *IEEE access : practical innovations, open solutions* 7 (2019), pages 30792–30801. (Visited on 20/06/2023).

[175] Y. Wang et al. ‘Channel Estimation in IRS-enhanced mmWave System with Super-Resolution Network’. In: *IEEE Communications Letters* 25.8 (2021), pages 2599–2603.

[176] H. Wu et al. ‘Identification of Encrypted Video Streaming Based on Differential Fingerprints’. In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2020, pages 74–79.

[177] H. Wu et al. ‘Resolution Identification of Encrypted Video Streaming Based on HTTP/2 Features’. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 19.2 (Feb. 2023).

[178] T. Wu. ‘CNN and RNN-based Deep Learning Methods for Digital Signal Demodulation’. In: *Proceedings of the 2019 International Conference on Image, Video and Signal Processing*. Shanghai China: ACM, Feb. 2019, pages 122–127. (Visited on 16/06/2023).

[179] W. Xia et al. ‘A Survey on Software-Defined Networking’. In: *IEEE Communications Surveys & Tutorials* 17.1 (2015), pages 27–51.

[180] H. Xie et al. ‘Deep Learning Enabled Semantic Communication Systems’. In: *IEEE Transactions on Signal Processing* 69 (2021), pages 2663–2675.

[181] Y. Xie et al. ‘Guiding the underwater acoustic target recognition with interpretable contrastive learning’. In: *OCEANS 2023-Limerick*. IEEE. 2023, pages 1–6.

[182] Xisuo Ma et al. ‘Data-Driven Deep Learning to Design Pilot and Channel Estimator for Massive MIMO’. In: *IEEE Transactions on Vehicular Technology* 69.5 (Mar. 2020), pages 5677–5682.

[183] Q. Xu et al. ‘Self-supervised learning-for underwater acoustic signal classification with mixup’. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* (2023).

[184] T. Xu et al. ‘Bit allocation using optimization’. In: *International Conference on Machine Learning*. PMLR. 2023, pages 38377–38399.

[185] Xuanxuan Gao et al. ‘ComNet: Combination of Deep Learning and Expert Knowledge in OFDM Receivers’. In: *IEEE Communications Letters* 22.12 (Oct. 2018), pages 2627–2630.

[186] L. Yang et al. ‘Personalized Federated Learning on Non-IID Data via Group-based Meta-learning’. In: *ACM Trans. Knowl. Discov. Data* 17.4 (Mar. 2023).

[187] Y. Yang et al. ‘Improving inference for neural image compression’. In: *Advances in Neural Information Processing Systems* 33 (2020), pages 573–584.

[188] P. Yin et al. ‘Understanding straight-through estimator in training activation quantized neural nets’. In: *arXiv preprint arXiv:1903.05662* (2019).

[189] X. You et al. ‘Towards 6G Wireless Communication Networks: Vision, Enabling Technologies, and New Paradigm Shifts’. In: *Science China Information Sciences* 64 (2021), pages 1–74.

[190] Y. You et al. ‘Large batch training of convolutional networks’. In: *arXiv preprint arXiv:1708.03888* (2017).

[191] Yue Hao et al. ‘Power of Deep Learning for Channel Estimation and Signal Detection in OFDM Systems’. In: *IEEE Wireless Communications Letters* 7.1 (Feb. 2018), pages 114–117.

[192] C. Zhang et al. ‘Deep Learning in Mobile and Wireless Networking: A Survey’. In: *IEEE Communications Surveys & Tutorials* 21.3 (2019), pages 2224–2287. (Visited on 14/08/2023).

[193] M. Zhang et al. ‘Enhanced Efficiency BPSK Demodulator Based on One-Dimensional Convolutional Neural Network’. In: *IEEE access : practical innovations, open solutions* 6 (2018), pages 26939–26948.

[194] Y. Zhang et al. ‘Implicit Neural Video Compression’. In: *CoRR* abs/2112.11312 (2021).

[195] Y. Zhao et al. ‘Federated Learning with Non-IID Data’. In: (2018).

[196] Z. Zhao et al. ‘Deep-Waveform: A Learned OFDM Receiver Based on Deep Complex-Valued Convolutional Networks’. In: *IEEE Journal on Selected Areas in Communications* 39.8 (2021), pages 2407–2420.

[197] Zhijin Qin et al. ‘Deep Learning in Physical Layer Communications’. In: *IEEE Wireless Communications* 26.2 (Mar. 2019), pages 93–99.

[198] H. Zhu et al. ‘Federated learning on non-IID data: A survey’. In: *Neurocomputing* 465 (2021), pages 371–390.

[199] Y. Zhu et al. ‘Transformer-based Transform Coding’. In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022.* OpenReview.net, 2022.

[200] H. Zimmermann. ‘OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection’. In: *IEEE Transactions on Communications* 28.4 (Apr. 1980), pages 425–432.

Online References

[201] N. P. O. A. 2025. *The Nobel Prize in Chemistry 2024*. Retrieved on 14 01 2025. 2024. URL: https://www.nobelprize.org/prizes/chemistry/2024/summary/.

[202] N. P. O. A. 2025. *The Nobel Prize in Physics 2024*. Retrieved on 14 01 2025. 2024. URL: https://www.nobelprize.org/prizes/physics/2024/summary/.

[203] CLIC. *CLIC: Challenge on Learned Image Compression*. URL: http://compression.cc.

Summary

This thesis presents a set of works on the use of deep learning (DL) in the next generation of communications systems. Digital communications is a cornerstone technology of the modern information age. Having a conversation with someone on the other side of the planet is nowadays considered normal. The Internet is hard to think out of our daily lives and has also enabled tremendous scientific progress. One area that has seen tremendous progress as a result is the domain of artificial intelligence (AI). More specifically, the area of deep learning (DL) has seen immense progress in recent years. Tools such as ChatGPT and Dall-E 2 are only two of many examples of recent advancements in DL.

A natural question to ask is whether DL enables better solutions for common problems in communications systems. Here, communications systems refers to the entire concept of some device sending information to another device across a link. For example, sending an email can be done using a communications system. Many challenges pop up in such a system; where do we even begin with sending such an email? Modern communications system already have answers to this question. However, the use of DL could enable smarter and more efficient solutions. Such solutions are important to ensure, e.g., the Internet can operate well in the future.

A communications system can generally be split up in a set of layers. These layers each define an abstraction level within the network. Such a split makes it easier to think about how to transmit, e.g., an email. There is a layer that describes what an email should look like, a layer that makes sure all parts of the email arrives, a layer that decides what path the email should take through the network, and a layer that describes how the email should be sent physically. Each of these abstractions come with their own features and challenges.

In the context of these layers, this thesis considers the following two research questions on DL for next-generation communications systems:

RQ1: *How can deep learning techniques be effectively applied across different parts of the networking stack?*

RQ2: *What are the advantages and challenges of implementing deep learning in the physical layer versus the application layer?*

This thesis is divided into two parts with three chapters each. In total, there are eight chapters where the first and last chapter contain the introduction and conclusion. The first three content chapters discuss works that relate to the physical layer of the networking stack. The second three chapters contain works that relate to the application layer of the networking stack.

Chapter 2 focuses on improving deep learning in the physical layer of communication networks. We develop a more flexible neural receiver that can handle multiple types of modulations. A modulation type is a configuration that makes a trade-off between speed and reliability. This approach makes deep learning based communication systems modular, making them more adaptable and efficient.

Chapter 3 continues work on the physical layer, tackling the challenge of making deep learning receivers faster and more energy-efficient. We use a method of deep learning called group equivariant deep learning to build neural networks that inherently understand certain properties of radio signals. This results in smaller, more efficient networks that perform just as well as larger ones.

Chapter 4 shifts to underwater acoustics. We use a technique called contrastive learning to train neural networks without labeled data. This approach could improve underwater sound classification and potentially help simulate underwater communication channels more accurately.

Chapter 5 explores a security vulnerability in video streaming platforms like YouTube. Videos are generally streamed dynamically which is known to result in vulnerabilities. Using deep learning techniques, specifically deep-metric learning, we demonstrate that video IDs can be identified from encrypted streams with minimal examples. This work highlights how DL simplifies exploiting vulnerabilities between communication layers.

Chapter 6 focuses on improving neural image compression. Building upon the work of Yang et al. (2020), we propose SGA+, a set of functions that improve the weighting of Gumbel probabilities. Our approach, particularly the SSL function with hyperparameter a, converges faster and achieves better PSNR/BPP trade-offs than the original SGA method. This advancement in neural image compression is important for reducing internet congestion and improving edge device capabilities.

Chapter 7 investigates the application of Federated Learning (FL) in Face Recognition (FR) to address privacy concerns and issues with data heterogeneity. Given the challenges of non-shared identities across parties, we propose using federated meta-learning. Our approach demonstrates improved overall performance per client, especially under heterogeneous data splits. Notably, the performance gains primarily benefit weaker clients, reducing the variance in performance across clients.

Together, these works provide partial answers to the research questions posed above. Major factors in applying DL across network layers are modularity, translating DL research to communications, and integrating DL advancements into the communications stack of tomorrow. Notably, there are differences between the application layer and physical layer. This thesis found the physical layer to benefit more broadly from the inclusion of DL. However, the models used throughout this work were highly similar; they were all based on convolutional neural networks (CNNs). The future of DL for communications looks bright and much work remains to be done.

Samenvatting

Deze scriptie presenteert een reeks onderzoeken over het gebruik van deep learning (DL) in toekomstige communicatiesystemen. Digitale communicatie is een hoeksteen van het moderne informatietijdperk. Tegenwoordig wordt het als normaal beschouwd om een gesprek te voeren met iemand aan de andere kant van de planeet. Het internet is onmisbaar geworden in ons dagelijks leven en heeft ook enorme wetenschappelijke vooruitgang mogelijk gemaakt. Een gebied dat hierdoor veel vooruitgang heeft geboekt, is kunstmatige intelligentie (AI), met name deep learning (DL). Tools zoals ChatGPT en Dall-E 2 zijn slechts twee voorbeelden van recente ontwikkelingen in DL.

Een logische vraag is of DL betere oplossingen biedt voor veelvoorkomende problemen in communicatiesystemen. Hier verwijst 'communicatiesystemen' naar het hele concept van een apparaat dat informatie naar een ander apparaat stuurt via een verbinding, zoals bij het verzenden van een e-mail. Hoewel moderne communicatiesystemen al oplossingen bieden, kan DL slimmere en efficiëntere oplossingen mogelijk maken, wat belangrijk is om bijvoorbeeld de toekomstbestendigheid van het internet te waarborgen.

Communicatiesystemen kunnen doorgaans worden opgedeeld in lagen, die elk een abstractieniveau binnen het netwerk definiëren. Deze opsplitsing maakt het eenvoudiger om na te denken over transmissieprocessen, zoals bij een e-mail. Er zijn lagen die bepalen hoe een e-mail eruit moet zien, hoe alle onderdelen correct aankomen, welke route de e-mail door het netwerk moet volgen, en hoe deze fysiek wordt verzonden. Elke laag brengt unieke kenmerken en uitdagingen met zich mee.

In dit kader behandelt dit proefschrift twee onderzoeksvragen over DL voor communicatiesystemen van de volgende generatie:

RQ1: *Hoe kunnen deep learning-technieken effectief worden toegepast op verschillende netwerklagen?*

RQ2: *Wat zijn de voordelen en uitdagingen van het implementeren van deep learning in de fysieke laag versus de applicatielaag?*

De inhoud van het proefschrift is verdeeld in twee delen met elk drie hoofdstukken. In totaal zijn er acht hoofdstukken, waarvan het eerste en laatste respectievelijk de introductie en conclusie bevatten. De eerste drie inhoudelijke hoofdstukken richten zich op de fysieke laag van de netwerkstack, terwijl de laatste drie hoofdstukken betrekking hebben op de applicatielaag.

Hoofdstuk 2 richt zich op het verbeteren van deep learning in de fysieke laag van communicatienetwerken. We ontwikkelen een flexibelere neurale ontvanger die meerdere soorten modulaties kan verwerken. Een modulatie is een configuratie die een afweging maakt tussen netwerk snelheid en betrouwbaarheid. Deze aanpak maakt op deep learning gebaseerde communicatiesystemen modulair, waardoor ze aanpasbaarder en efficiënter worden.

Hoofdstuk 3 zet het werk aan de fysieke laag voort en pakt de uitdaging aan om deep learning ontvangers sneller en energiezuiniger te maken. We gebruiken een methode van deep learning genaamd groepsequivariante deep learning om neurale netwerken te bouwen die inherent bepaalde eigenschappen van radiosignalen begrijpen. Dit resulteert in kleinere, efficiëntere netwerken die net zo goed presteren als grotere.

Hoofdstuk 4 verschuift naar onderwaterakoestiek. We gebruiken een techniek genaamd contrastief leren om neurale netwerken te trainen zonder gelabelde data. Deze aanpak zou de classificatie van onderwatergeluiden kunnen verbeteren en mogelijk helpen bij het nauwkeuriger simuleren van onderwatercommunicatiekanalen.

Hoofdstuk 5 onderzoekt een beveiligingskwetsbaarheid in videostreamingplatforms zoals YouTube. Video's worden over het algemeen dynamisch gestreamd, wat bekend staat als een bron van kwetsbaarheden. Met behulp van deep learning-technieken, specifiek deep-metric learning, tonen we aan dat video-ID's kunnen worden geïdentificeerd uit versleutelde streams met minimale voorbeelden. Dit werk benadrukt hoe DL het uitbuiten van kwetsbaarheden tussen communicatielagen vereenvoudigt.

Hoofdstuk 6 richt zich op het verbeteren van neurale beeldcompressie. Voortbouwend op het werk van Yang et al. (2020), stellen we SGA+ voor, een set functies die de weging van Gumbel-waarschijnlijkheden verbeteren. Onze aanpak, met name de SSL-functie met hyperparameter a, convergeert sneller en bereikt betere PSNR/BPP-afwegingen dan de originele SGA-methode. Deze vooruitgang in neurale beeldcompressie is belangrijk voor het verminderen van internetcongestie en het verbeteren van de mogelijkheden van randapparaten.

Hoofdstuk 7 onderzoekt de toepassing van Federated Learning (FL) in gezichtsherkenning (FR) om privacyproblemen en kwesties met dataheterogeniteit aan te pakken. Gezien de uitdagingen van niet-gedeelde identiteiten tussen partijen, stellen we voor om federated meta-learning te gebruiken. Onze aanpak toont verbeterde algehele prestaties per cliënt, vooral bij heterogene dataverdelingen. Opmerkelijk is dat de prestatiewinst voornamelijk ten goede komt aan zwakkere cliënten, waardoor de variantie in prestaties tussen cliënten wordt verminderd.

Samen bieden deze werken gedeeltelijke antwoorden op de hierboven gestelde onderzoeksvragen. Belangrijke factoren bij het toepassen van deep learning op verschillende netwerklagen zijn modulariteit, het vertalen van DL-onderzoek naar communicatiesystemen en het integreren van DL-vooruitgangen in de communicatiestack van de toekomst. Opmerkelijk is dat er verschillen zijn tussen de applicatielaag en de fysieke laag. Dit proefschrift heeft vastgesteld dat de fysieke laag in bredere zin profiteert van de integratie van DL. De modellen die in dit werk zijn gebruikt, waren echter sterk vergelijkbaar; ze waren allemaal gebaseerd op convolutionele neurale netwerken (CNN's). De toekomst van DL voor communicatiesystemen ziet er veelbelovend uit, maar er blijft nog veel werk te doen.

About the Author

Adriaan Winand (Arwin) Gansekoele was born on June 24, 1997, in Almere. In 2010, he obtained a VWO diploma in the Science & Technology (N&T) track at the Oostvaarderscollege in Almere. He then started the BSc program in Business Analytics at VU Amsterdam. Out of interest in information systems, he also pursued a BSc. in Computer Science at VU. For his BSc. Business Analytics graduation project, he worked in a group to investigate the causes of long queues at Schiphol Airport, under the supervision of Prof. Dr. Rob van der Mei. For the BSc Computer Science, he completed a graduation internship supervised by dr. Karine Miras and prof. dr. A.E. Eiben. This resulted in the article "Insights in evolutionary exploration of robot morphology spaces." After completing both BSc programs, he attended the MSc Artificial Intelligence program from 2018 to 2020. For his graduation project, he interned at Prosus, with daily supervision by dr. Luis Armando Pérez Rey and dr. Dmitri Jarnikov. He was supervised research by dr. P.S.M. Mettes and prof. dr. C.G.M. Snoek. This resulted in the master's thesis titled "The Fisher-Watson Detector for Out-of-Distribution and Adversarial Detection."

In 2021, he started this PhD thesis at the Centrum Wiskunde & Informatica, pursuing his doctorate at the Vrije Universiteit Amsterdam. During this four-year period, he produced several peer-reviewed papers, many of which he presented at international conferences. Furthermore, he supervised several MSc students during their graduation internships. The resulting dissertation will be defended on September 17, 2025. At the time of writing, he has continued doing Deep Learning research in the role of Data Scientist at the Ministry of Defence.