Optimizing Neural Networks for Low-Complexity Channel Estimation

van Lier, M.L.

Award date:
2020

Technische Universiteit
Eindhoven
University of Technology

The department of Electrical Engineering
Electronic Systems Research Group

Optimizing Neural Networks for Low-Complexity Channel Estimation

Master Thesis

Michel van Lier

Supervisors:
prof.dr. Henk Corporaal, (TU/e)
dr.ir. Zoran Zivkovic, (Intel Corporation)
dr.ir. Gijs Dubbelman, (TU/e)
dr. Alexios Balatsoukas Stimming, (TU/e)

Eindhoven, January 2020

Abstract

Due to the easy affordable access to powerful mobile devices, cloud services, and high global mobile broadband internet penetration, users consume more data than ever. This also makes new applications possible in new domains and scenarios that require high demands on the connection. Deep learning has not only gained a tremendous amount of interest as a dominant algorithm in computer vision applications but also as a promising technique for signal processing tasks such as (wireless) communication. However, these models typically require a significant amount of memory storage and are computationally demanding to run on embedded platforms, especially when supporting a wide range of signal-to-noise ratios.

In this work, we use a neural network model to perform pilot channel denoising in an OFDM receiver, to avoid the challenging task of estimating the channel covariance matrix and to improve the channel estimation performance. To avoid the large requirements introduced by these models, we present a framework for systematic design-space exploration of neural network configurations size, quantization, pruning, and layer sharing. On top of this, we propose two methods to maximize the channel denoise performance in a wide range of signal-to-noise ratio scenarios, while minimizing the computational complexity and memory storage requirements.

Using this framework, we have evaluated the performance and requirements of several neural network architectures including; fully-connected, fully-connected complex-valued, convolutional, and binary neural networks. The results reveal that choosing an appropriate neural network configuration size is crucial to reduce the complexity of our neural-network channel filter methods. On top of these results, we show that similar to neural network quantization and pruning in other domains, careful quantization and pruning can lead to significant complexity reduction with negligible performance degradation. In total, the model memory size can be reduced up to 83.2%, while the computational complexity can be reduced up to 86.2%. Finally, the results suggest that using a solution with multiple distinct neural networks trained for different signal-to-noise ratios leads to lower overall computational complexity and storage requirements compared to using a single bigger neural network trained for the entire signal-to-noise ratio (SNR) range. Furthermore, the results show that the fully-connected neural networks are the best-performing ones while having lower overall requirements. These results indicate that the proposed framework is an effective way to reduce the requirements, find multiple configurations for a broad range of targets, with minimal performance decrease.

Contents

Chapter 1

Introduction

5G, Artificial Intelligence, and Mobile First are one of the top tech buzzwords of 2019 and 2020 [1, 2]. Nowadays it is hard to miss any media information related to these topics and scientific papers in these fields have massively grown [3]. While "Mobile First" has taken a visible place in our lives, this is not yet the case for Artificial Intelligence and 5G. It will revolutionize our lives said Meredith Attwell Baker, president, and CEO of the wireless communications trade group CTIA, in a typical observation about 5G. While current Google CEO Sundar Pichai declared Artificial Intelligence to be probably the most important thing humanity has ever worked on. This Chapter will introduce these topics about (5G) mobile communications with Artificial Intelligence and the need to improve these. Furthermore, this chapter will introduce the problem statement to this and our contributions in these fields, followed by the outline of the rest of this report.

1.1 Introduction

Over the last decade, the increasing popularity of mobile-connected devices has led to a steady increase in global internet penetration. The penetration of the internet globally increased seven times, for the period 2000-2015, from 6.5% to 52%. In 2020 it is expected, that the number of mobile phone users will rise to 4.78 billion, with penetration of 63% globally (ITU, 2015). Over the next decades, the global market penetration of connected devices is expected to grow in emerging markets such as India and Africa and will serve mobile broadband internet to more people than ever.

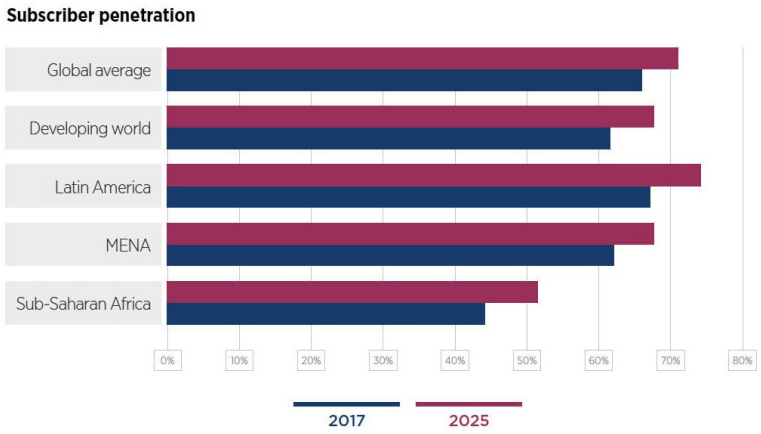

Figure 1.1: Cellular mobile network subscriptions expectation for 2025 (image source: [4]).

5G New Radio (5G NR) cellular network technologies are expected to become a major driver in the coming decades for mobile communication, automotive, home internet and Internet of Things (IoT) and other domains due to their key features. These key features include ultra-lean transmission [5], support for low latency, advanced antenna technologies, and spectrum flexibility including operation in high-frequency bands, inter-working between high and low-frequency bands, and dynamic time-division multiplexing (TDD), which result in high bandwidths and others. For the interested reader, a summary overview of these key features is provided in [6]. Due to the easy affordable access to mobile devices and services and mobile broadband internet penetration, users consume more data than ever, in more scenarios than ever. The usage of high-resolution music streaming and video streaming, cloud apps, Connected Cars, and cloud gaming, real-time video/audio communication, typically require high bandwidth and real-time response. These types of applications are intolerant to latency issues and low data rate transmission which cause a low Quality of Service. Low Quality of Service is not desirable since it results in bad user experience or even unusability of the service. During communication between the transmitter and receiver, the received signal is usually distorted in the communication channel by channel characteristics which result in latency issues and low data rate transmission. To recover the transmitted data, and to achieve a high Quality of Service, many processing steps are involved including estimation of the channel characteristics and compensation at the receiver.

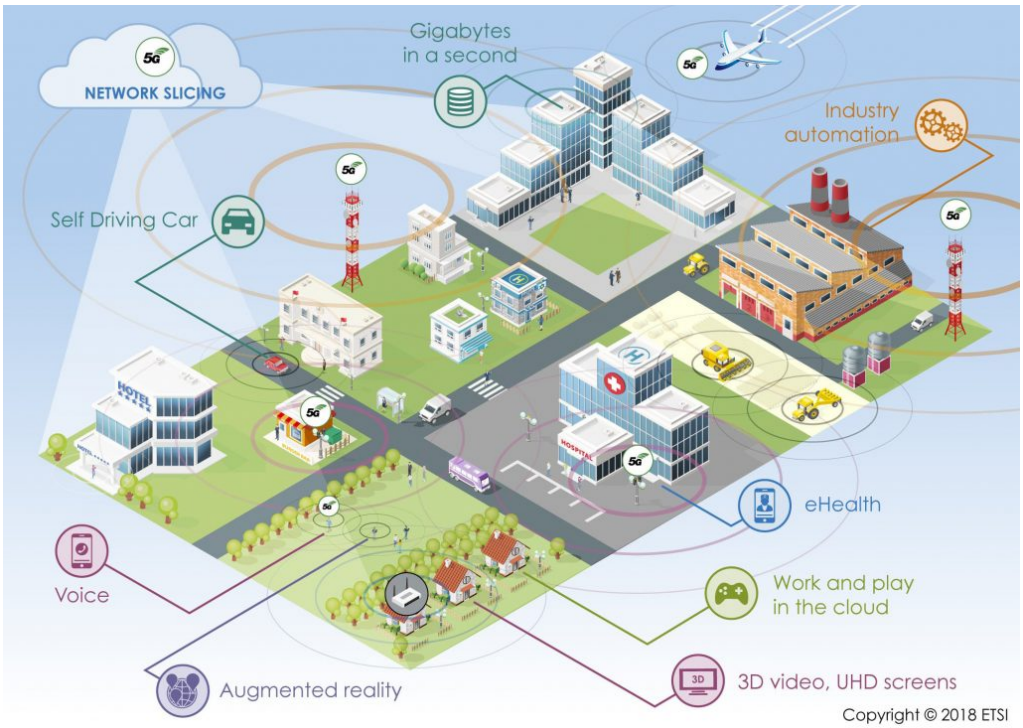

Figure 1.2: Example of possible 5G use cases (image source: [7]).

Deep Learning (DL) has gained a tremendous amount of interest as a promising Machine Learning (ML) paradigm since it outperforms traditional algorithms and even humans in visual object recognition [8]. This enables many useful applications in a wide range of fields. Examples are image and video classification/analysis/denoising/up-sampling, natural language processing, up to more specialized fields, to name a few. More specifically DL in communication has gained a tremendous amount of interest in recent years. Many approaches have been proposed specifically for improving the quality of service at the receiver side, from approaches that replace the total receiver chain [9], up to more fine-grained data/model-driven approaches which replace only parts of the receiver chain and focusing on channel estimation [10, 11, 12].

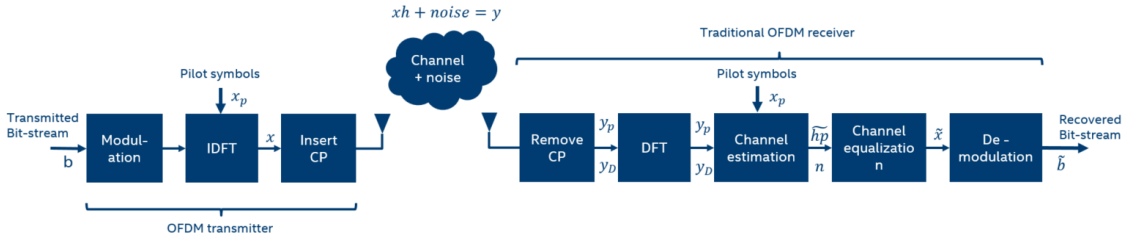

Figure 1.3: Example of an simplified OFDM system with a transmitter (left), channel (middle) and receiver (right).

1.2 Problem Statement

Embedded mobile hardware platforms are constrained in memory and computation power and often use reduced-precision integer arithmetic. As a result, practical implementation of the above mentioned DL approaches, on modem platforms, does not fit the time and cost constraints, especially in wide signal to noise ratio scenarios, for broadband cellular network technologies such as 4G/LTE and 5G/NR. Although many DL computation and memory reduction techniques have been proposed, such as quantization [13, 14, 15], pruning [16, 17]. These compression methods focus typically on computer vision applications to solve classification problems. Their complexity reduction and performance impact are unknown on DL communication applications. As such, the performance impact on regression problems is less understood. This raises the research question; *How can we improve the channel estimation with deep learning, with lower computational complexity and memory size requirements than current approaches?* It should be known to what extent is it possible to reduce the memory and computation requirements of this DL approach with fixed-point arithmetic, with minimal impact on the quality of service in realistic physical wireless channel models. While most of the current DL-based Rx are usually trained for one specific scenario or signal-to-noise ratio value, the memory size and/or computational complexity massively increase when covering multiple scenarios. These cases should have therefore extra attention.

1.3 Contributions

To address these issues, this work is focusing on replacing only a sub-block of the channel estimation block from the orthogonal frequency-division multiplexing (OFDM) receiver Figure (1.3) by a deep residual de-noising neural network (NN). OFDM is a common modulation scheme between transmitters (Tx) and Rx. It was shown in [11] that focusing on the inference part instead of the standardized blocks, higher quality of service can be archived, with lower computation and memory requirements.

In this work we make the following contributions:

- A systematic design-space exploration of multiple channel denoising NN configurations for channel estimation, for different types of NN architectures. We show that a careful design of the NN configuration is a necessary requirement to obtain Pareto-optimal performance-complexity trade-offs. This is a rather well-known fact in the machine learning community, but so far has received little to no attention from the communications community.
- We show that complexity reduction by applying existing computer vision proven quantization and pruning methods work particularly well for this application with minor denoise performance decrease. Furthermore, due to fixed-point quantization, these models can run on fixed-point hardware. The computational complexity in multiply-accumulates (MACs) can be reduced up to 86.2% while the memory storage in KB can be reduced by 83.2%.
- Handling large signal-to-noise ration in data using multiple carefully designed NNs trained over distinct SNR ranges outperform a single NN trained over the union of the individual

SNR ranges both in terms of the computational complexity, reduced with 74% MACs and total memory size reduced with 23% and in terms of the achieved denoising performance of more than 1.62dB.

While the focus here is on the channel estimation, the presented methods can also be applied to any of the Rx DL approaches, e.g. [9], [11], [10].

1.4 Outline

The organization of this thesis is as follows: the next Chapter (2) discusses related works on the topic of channel estimation with DL, handling large SNR range in data with DL approaches and computational and memory complexity reduction strategies for DL models. Chapter (3) provides a brief introduction of OFDM communication, the wireless channel and classical channel estimation algorithms. Additionally, popular denoising NN architectures are discussed. In Chapter (4) we introduce generic channel denoising architecture who are explored with our framework. Chapter (5) introduces the optimization framework with a focus on optimizing the model size, memory and computation size by quantization and pruning and other methods. The evaluation methodology and results are discussed in Chapter (6). Chapter (7) contains the derived conclusions.

Chapter 2

Related Work

The field of ML and more specific DL has a long and successful history for over more than 70 years [18]. DL has shown its advantages in many areas, where it is normally difficult to find a concrete mathematical model for feature representation. In those areas, ML has proved to be a powerful tool as it does not require a comprehensive specification of the model. The domain of communications and its applications has mainly relied on theories and models, from information theory to channel modeling. These traditional approaches are showing limitations, especially when the applications and scenarios. Therefore, research on ML applied to communications, especially to wireless communications, is rising. This chapter will introduce the related work of DL-based OFDM receivers and their problems and how to tackle wide SNR problems which we will try to extend. This is followed by a brief discussion on memory and computation requirement reduction techniques and Complex-Valued Neural Networks (CVNNs) which are used to extend the previous related works.

2.1 Deep Learning-based Channel Estimation

The use of ML in wireless communication and in particular DL and NNs, to tackle communications tasks has attracted significant interest in the past few years [19, 20, 21] systems covers a wide range of applications. These comprise channel prediction, estimation, localization, equalization, coding/decoding, signal detection, to name a few. Jiang et al. [22] summarized the DL possibilities in 5G communication systems. Specifically for improving the performance of channel estimation in OFDM systems, ranging from approaches which replace the entire receiver chain [9], to model-based approaches which replace only parts of the receiver chain [12, 11, 10] .

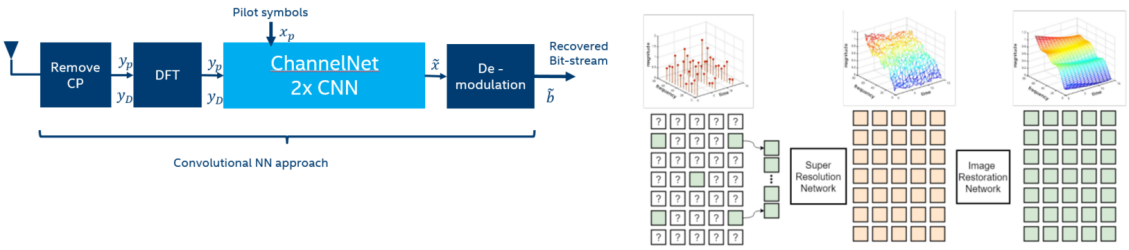

Figure 2.1: (Left) Partial DL based OFDM receiver, based on two CNN models as proposed by Sultani et. al. [12]. (Right) A visual example of the two step approach (a channel in frequency and time is seen as an image); "channel-image" up-scaling followed by de-noising.

Sultani et. al. [12] proposed to use a pipeline of two Convolutional Neural Network (CNN) models for channel estimation. They consider the time-frequency response of a channel as a low-

resolution image and use a super-resolution NN cascaded with a de-noising NN to estimate and interpolate the channel in both time and frequency. They showed that the presented approach is comparable to the ideal Linear Minimum Mean Square Error (LMMSE) solution and performs better than Approximated-LMMSE, in a realistic channel model.

Ye et al. [9] proposed a full replacement of the OFDM receiver. The OFDM receiver blocks from Figure (1.3), from the discrete Fourier transform block up to the demodulation block, are replaced by a five-layer fully connected DNN. The DNN model takes as input four data and four pilot symbols and recovers the transmitted data. Their initial experiment has shown to perform better than traditional Least Squares and LMMSE detection especially when fewer pilot symbols than data symbols used.

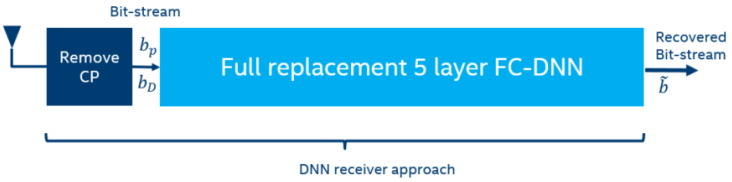

Figure 2.2: Full replacement DL based OFDM receiver, as proposed by, Ye et al. [9].

ComNet is proposed by Gao et al. [11] to replace the conventional or full replacement FC-DNN OFDM receiver and has shown to perform better. The proposed ComNet receiver uses DL to refine traditional channel estimation and equalization block and signal detection block in a hybrid way, rather than replacing the entire blocks. First, the channel at the pilot positions is estimated by Least Squares. Then with a single fully connected layer (LS-RefineNet), the channel at the data positions is estimated and refined. In signal detection, the Zero-Forcing solution is used to estimate the transmitted symbols and refined with a two-layer fully connected DNN (ZF-RefineNet) to demodulate and estimate the sent binary data stream.

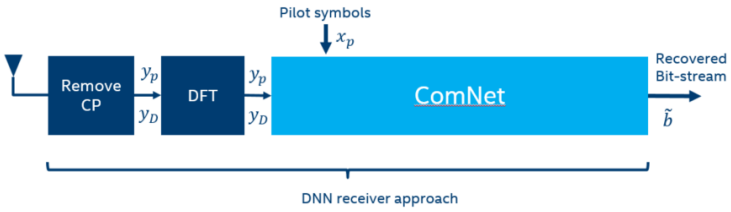

Figure 2.3: Partial replacement DL based OFDM receiver, as proposed by, Gao et al. [11].

SwitchNet is an improved approach based on ComNet by Jiang et al. [10]. In comparison with SwitchNet, they proposed to add a second fully connected layer with a trainable switch after LS-RefineNet in the channel estimation block. The first layer is trained on channels with a short delay, while the second layer is trained on channels with a large delay, to compensate for the mismatch of the first layer. The switch is trained to switch on the second layer when a large delay is detected, to minimize the channel mismatch, to avoid performance degradation.

While the FC-DNN, ChannelNet, ComNet, and SwitchNet outperform the traditional methods such as the Least Squares and derivative of the (simplified) Wiener filter, they require significantly more memory and computation as shown in Table (2.1). As a comparison, conventional LS and MMSE estimators require $\mathcal{O}(P)$ and $\mathcal{O}(P^3)$ MAC-like computations, respectively, and P is typically a relatively small value. Moreover, LS channel estimation requires no storage, while MMSE channel estimation requires storage of a $P \times P$ matrix. Typically wireless communications receivers contain low power, embedded hardware such as DSPs, which are computation and memory

constrained. Deploying these state-of-the-art algorithms on constrained hardware, while meeting the best and worst-case processing time deadlines for cellular network technologies becomes a challenge. We will extend this related work is an new DL-based channel OFDM Rx approach, which will significantly use less computation and memory than previous related works.

Table 2.1: Complexity Comparison of Neural-Network-Based pilot aided OFDM Channel Estimators.For 5G/NR the best case time constrains are P=48 within 71.35μs, while the worst-case time constrains are P=1650 within 8.92μs.

Model	MACs	Model size
Least Squares	$\mathcal{O}(P)$	0 MB
Wiener filter (MMSE)	$\mathcal{O}(P^3)$	$P \times P$
ChannelNet [12]	13.15 M	35.82 MB
ChannelNet (low+high SNR) [12]	13.15 M	71.65 MB
ComNet-BiLSTM [11]	10.40 M	2.4 MB
Full replacement FC-DNN [9]	6.96 M	27.85 MB
ComNet-FC [11]	0.37 M	1.25 MB
SwitchNet [10]	0.4 M	1.82 MB

2.2 Handling Large SNR Range in Data

Typically for the domain of wireless communication systems, are the large SNR range in which these systems work. Due to the free space path loss ($P_{sig} = 1/distance^2$), and other distortions (shadow/multi-path fading), when the signal propagates through the wireless channels, the signal power (P_{sig}) decreases massively with increasing distances between the TX and RX leading to wide range of signal to noise ratios ($SNR = P_{sig}/P_{noise}$).

Although DNN/CNNs have proven to have strong generalization capabilities in many applications, Liu et al. [23] have shown that for speech, that training on a single SNR, but validating on larger SNR range, results in a performance drop at the unseen noise levels, in comparison when trained on these levels. Xu et al. proposed to input the SNR level as a specific (learned) code for adaptation, together with dropout, to improve the generalization of the network. To further improve performance on large SNR ranges, Fu et al. [24] proposed an SNR-Aware CNN for speech enhancement, were, an SNR-depended CNN model is selected for denoising, based on the SNR level. Sultani et. al. [12] proposed a similar approach, to use two models for low and large SNR range in an OFDM receiver.

The above approaches show good performance on a wide range of signal-to-noise ratios in various domains and applications when multiple models are used. However, using multiple models greatly increases the total memory storage requirements. Reducing the model size is therefore crucial for a realistic implementation on a mobile modem platform. Our extension to this related work is to improve the performance in wide range signal-to-noise ratio scenarios while minimizing the memory storage and computational complexity requirements as proposed in Section 5.6.

2.3 Deep Learning Complexity Reduction

Due to the high computation and memory requirements of DL models in the computer vision domain, especially when deployed in real-time, resource-constrained situations, a large number of works have been devoted to reducing DL model size and improving inference efficiency, while keeping the accuracy as high as possible. There are mainly three design strategies that are discussed.

2.3.1 Compact Aware Network Design

Compact aware network design is a model-driven technique to design a compact network architecture from scratch, while it has to achieve as high as possible classification performance. E.g. Network Distillation [25] is a model compression approaches in which a pre-trained large model teaches a smaller model to achieve a similar prediction performance. Multiscale DenseNet [26] uses early-exit classifiers to enable anytime classification in the DNN pipeline. GoogLeNet [27] and SqueezeNet [28] use 1x1 convolution layers in the models to reduce the computation. Furthermore, MobileNet [29] utilizes separable convolutions in the filters and MobileNet V2 [29] stacks several layers of depthwise separable convolutions. ShuffleNet V2 [30] employs pointwise group convolutions with channel shuffling and split. These approaches can reduce the computation with a factor of up to 2, and memory size factor up to 8 without classification accuracy decrease.

2.3.2 Parameter Reduction

Parameter reduction is a strategy to reduce the number of parameters in new and existing networks while keeping accuracy as high as possible. With low-rank factorization, the matrix decomposition is used to estimate the informative parameters and can be applied on both from scratch and pre-trained DNNs [31, 32]. Hinton et. al [33] proposed knowledge distillation, were a compact version of the original DNN is trained, that distilled the knowledge of the larger high precision model equivalent trained on the same dataset. Furthermore, Chen et al. [34] showed a Hashing approach that reduced the total memory storage up to 64x, with minor classification accuracy loss of 2%. In the extreme case methods such as pruning can be used which deletes weights, channels, filters, and layers, that do not add to the discriminative power of the network. By pruning the convolutional layers [35] and/or the fully connected layers [17], reduction of 13 in model size and 5 in computing operations are achieved [16, 17]. By applying Huffman encoding on top of pruning, model size can be further increased by 1.5x without loss of accuracy [36]. But this approach comes at a cost of extra decoding hardware.

2.3.3 Quantization

Quantization can be applied to either the weights, biases, and activations or a combination of them. Next to this, the training parameters; gradients can be quantized, although this is only for speeding-up (distributed) DNN training. A lot of strategies have been proposed to quantize NNs, which can be divided into two subsections; deterministic and stochastic quantization. In deterministic quantization, there is a one-to-one mapping between the quantized value and the real value. While with stochastic quantization, the weights and activations are discretely distributed. The quantized value is sampled from the discrete distributions. Furthermore different kinds of codebooks can be used for quantization: fixed codebook quantization and adaptive codebook quantization. In fixed codebook quantization, the weights are quantized into a predefined codebook while in adaptive codebook quantization the codebook is learned from the data. The recent quantized NNs have achieved accuracy similar to their full-precision counterparts. Where typical approaches are; rounding [13], vector quantization [36] and quantization as optimization problem [15], wherein the extreme case both weights and activations are quantized up to one bit only [37].

Although the above-mentioned approaches show significant reductions often more than 60 times in both memory and computation requirements, with minor accuracy decrease in image classification. The performance impact is usually only known on computer vision algorithms. Additionally, these quantization methods focus typically on classification problems and do not take into account the multiple (overlapping) models in wide SNR range problems. Since there are already many different successful algorithms, we will not pose to improve them. The related work will help in selecting appropriate approaches to reduce memory storage and computational complexity requirements in normal and wide signal-to-noise scenarios. To the best of our knowledge, this work is first who is applying these complexity reduction methods on DL models in the communications

domain. The proposed reduction methods are proposed in Section (5.3.2), (5.4) and (5.5).

2.4 Complex-Valued Neural Networks

Most of the work in DL is focused on representing DL networks, with real-valued arithmetic, even for most of the DL networks used in the complex domain. As an example, all the DL based OFDM receivers discussed in Section (2.1), operate with real-valued networks. This is contradictory, as all the blocks in the OFDM receiver after the Discrete Fourier Transform operates in the frequency domain, where data is represented in the complex domain. Due to this, mobile hardware platforms in (OFDM receiver) modems often use only complex-valued arithmetic. Performing real arithmetic with complex comes with inefficiencies. Motivated by this, we explore the related work of CVNNs.

The use of CVNNs is not new and has been investigated long before the earliest DL breakthroughs. For the interested reader, Hirose [38] reviews the long history of the field of CVNNs. Recently Reichert et al. [39] Bruna et al. [40] Arjovsky et al. [41] Danihelka et al. [42] Wisdom et al. [43] have tried to bring more attention to CVNNs by providing theoretical and mathematical motivation. These CVNNs are of interest due to their generalization properties [44], and the ability to capture phase in signals [45]. [45] claims that the power of CVNNs lies in their ability to handle phase-amplitude transformations to signals.

For example, Trabelsi et al. [46] used Complex-Valued Convolutional Neural Networks (CV-CNNs) for image classification of the CIFAR-10, CIFAR-100 an SVHN dataset, but was slightly outperformed by the real-valued CNN counterpart. Anderson et al. came to a similar conclusion for the CIFAR-10 and MNIST dataset, while the CVNN models contained double the parameters. Applications in the complex domain showed more competitive results. Trabelsi et al. [46] showed state-of-the-art performance on the MusicNet dataset [47], with only 9/10 of the parameters, and state-of-the-art performance on the speech spectrum prediction on TIMIT. Moran et al. [48] showed that using a CVNN to invert the complex transmission effects of optical fiber distortion on an input image, totally outperforms the real-valued counterpart network. Gao et al. [49] showed that CV-CNNs can achieve state of the art performance in estimation distorted radar echo signal, in comparison with real-valued networks.

Despite these works of CVNNs, and its motivations, practical real-world applications show mixed results. While performance differs per application, the performance of channel estimation de-noising with complex-valued networks is unknown. Our contribution of this related work is not to improve these CVNNs, but to evaluate the performance and requirements of this architecture on our application, as OFDM Rx usually contains complex-arithmetic hardware.

Chapter 3

Background

OFDM is a digital modulation technique to modulate digital data in multiple parallel carriers, and is successfully widely used in many (wireless) popular communication systems such as for broadband internet communication; LTE, home internet; WLAN, WiMAX, digital radio; DAB, digital television; DVB-T/H, and others. These techniques are used worldwide by many customers, and devices such as smartphones, tablets, laptops, PCs, smart-televisions and many others. The received signal is usually distorted by channel characteristics, which is bad for the performance. To recover the transmitted symbols, many processing steps are involved including estimation of the channel characteristics and compensation at the receiver. Communication systems become more complex, and recovering data with traditional methods is becoming more challenging. Previous related work has shown that neural networks can be an alternative to these classical approaches. This chapter provides a summary of the OFDM wireless communication system, to provide a better understanding of the role of channel estimation in such a system. Furthermore, we provide a background on classical channel estimation solutions. We finish the chapter with the introduction of denoising NN and popular NN architectures.

3.1 OFDM Model

The concept of OFDM communication is transmitting and receiving parallel data in the frequency domain. The transmitted (binary) data symbol stream, is split into F parallel data streams, which are transmitted on equal spaced frequencies called sub-carriers. The transmission rate of each such subcarrier is $1/F$ times lower than the original (non-split) data rate. This is obtained by splitting the original data stream into multiple blocks, which are transmitted in consecutive time intervals, and where each symbol of a block is assigned to a specific subcarrier. In OFDM systems, this frequency-domain multiplexing is usually achieved by fast Fourier transform (FFT), as shown in Figure (1.3). By using orthogonal subcarriers, OFDM communication systems avoid (partially) the intercarrier interference (ICI) among the data symbols of the same OFDM block. ICI is the effect when symbols received from multiple paths are delayed and overlap each other in time. The ICI causes high transmission error rates because the symbols from multiple received paths interfere with each other and become indistinguishable by the receiver. The main cause of ICI is introduced by the wireless channel: multipath fading and scattering environments. The basic method to avoid this ICI is by adding a cyclic extension to each OFDM block, usually noted as cyclic prefix (CP). But, when the channel experiences a minor time variation, each subcarrier will experience a Doppler spreading effect that will destroy the subcarrier orthogonality, producing significant ICI [50, 51]. Furthermore, due to the fast-changing channel, recovering of this effect becomes more complicated. And usually, equalizers are used to mitigate ISI and improve receiver performance.

Such a typical OFDM wireless communication system is shown in Figure 1.3, as an end-to-end model. The binary data stream to be transmitted from the transmitting (Tx) device is inserted

in the first block, where after a chain of operations, the data is sent over a wireless channel to the receiving (Rx) device. At the Rx side, the received data is going through a chain of the inverse operations that are applied until the received data is recovered and demodulated to the original send data.

3.1.1 OFDM Transmitter

Figure (1.3) presents a simplified overview of the OFDM Tx. The input of this transmitter is the bit-stream that should be transmitted. The first step is to modulate the digital signal. Digital modulation is methods to transfer a digital bit stream over an analog baseband channel. To accomplish this the modulator changes the characteristics of the carrier waveform, popular techniques are e.g. PSK and QAM. Hereafter, the signal is converted from serial-to-parallel, and the stream of symbols is split into data blocks. Each OFDM block, of size F, can contain either data symbols or pilot symbols, or both data and pilots, depending on the training pattern. These pilot symbols are known at both the Tx and Rx and assist the Rx side e.g. in channel estimation. The OFDM symbol is transmitted on the fth subcarrier of the kth OFDM block and is denoted by $x[f, k]$. Eventually, a CP is inserted to compensate for the ICI. The resulting OFDM symbols are then converted from parallel-to-serial, usually with the inverse discrete Fourier transform (IDFT) on f parallel subcarriers. and transmitted over the wireless channel.

3.1.2 Wireless Channel Model

In the above-described OFDM communication system, such as shown in Figure. (1.3), typically sends complex data symbols in parallel over a set of frequencies F. Let the complex data OFDM symbol $x[f] \in \mathbb{C}$. The symbols are modified by the characteristics of wireless signal as it travels from the transmitter antenna to the receiver antenna. These characteristics depend upon the distance between the two antennas, the path(s) taken by the signal, and the environment (objects) around the path to name a few. But also by the analog electronics characteristics.

Analytically it is useful to distinguish between three different effects on the channel, which result in an overall attenuation of the transmitted signal:

1. Multipath Fading is on of the main causes of ICI, caused by a multipath propagation environment, therefore by an environment reflecting the transmitted electromagnetic waves such that multiple copies of this wave interfere at the receiving antenna. Multipath Fading is a stochastic effect and leads to significant attenuation changes in the short term.

2. Shadow Fading is the stochastic effect when some part of the transmitted signal is lost through absorption, reflection, scattering, and diffraction if there are objects along the path of the signal, which causes short term fluctuations in the attenuation.

3. Path loss is a deterministic effect, depending only on the distance between the Tx and the RX, and is a long term attenuation, since the distance between transmitter and receiver in most situations does not change significantly on smaller time scales.

All three attenuating effects combined, result in the experienced attenuation of the wireless channel model:

$$h[f] = a_{pl}[t] \times a_s[t] \times sf[t] \tag{3.1}$$

In a linear system with no interference between the symbols, the transmission can be modeled in the frequency domain as:

$$y[f] = h[f]x[f] + w[f], \quad f \in \{0, \ldots, F-1\}, \tag{3.2}$$

where $y[f] \in \mathbb{C}$, $h[f] \in \mathbb{C}$, and $w[f] \in \mathbb{C}$ are the frequency-domain representations of the received signal, the transmission channel, and the additive noise on subcarrier f, respectively. The noise $w[f]$ is typically modeled as additive white Gaussian noise (AWGN) distributed according

$\mathcal{CN}(0, \sigma^2)$. The channels $h[f]$ at different subcarriers, however, are typically correlated. If we define $\mathbf{h} = \begin{bmatrix} h[0] & h[1] & \dots & h[F-1] \end{bmatrix}$, then a common model for correlated channels is $\mathbf{h} \sim \mathcal{CN}(0, \mathbf{C_h})$, where $\mathbf{C_h}$ is the covariance matrix of $\mathbf{h}$.

3.1.3 OFDM Receiver

At the receiver, the received symbols are converted back to the parallel data stream, the CP is removed and eventually used to cancel any interference. The resulting OFDM symbols are sent through a DFT block to convert the data back from the time domain to the frequency domain, such that the corresponding symbols from the F orthogonal subcarriers could be obtained. The channel estimation consists of two steps: first, the pilot symbols are extracted on the agreed training pattern. The second step is to estimate the channel frequency response on the pilot and/or data places (this is discussed in more detail in Section (3.1.4)). These estimated channel frequency responses are used in the channel equalization block to recover the received distorted OFDM data symbols. To get CFR from the other subcarriers which dont have pilot symbols, and carries data, many approaches are used. One of them is interpolating

3.1.4 Classical Channel Estimation

Known pilot symbols $x[p]$, $p \in \mathcal{P} \subseteq \{0, \dots, P-1\}$, are sent occasionally for the purpose of estimating the channel. A simple channel estimation approach is the so-called *least-squares solution* obtained by element-wise division of the observed data and the known reference values. The Least-squares (LS) channel estimation is given by:

$$\hat{\mathbf{h}}_{\mathcal{P}}^{\mathrm{LS}} = \mathbf{y}_{\mathcal{P}} / \mathbf{x}_{\mathcal{P}}, \tag{3.3}$$

where $\mathbf{x}_{\mathcal{P}}$ and $\mathbf{y}_{\mathcal{P}}$ denote vectors containing the elements of $x[f]$ and $y[f]$, respectively, with indices in $\mathcal{P}$ and division is performed element-wise. The LS solution is noisy in the sense that it does not take the channel correlation into account. The minimum mean squared error (MMSE) solution [52], which exploits knowledge of the covariance matrix and the noise level σ to denoise the channel estimation, can be obtained as:

$$\hat{\mathbf{h}}_{\mathcal{P}}^{\mathrm{MMSE}} = \mathbf{C}_{\mathbf{h}_{\mathcal{P}}} (\mathbf{C}_{\mathbf{h}_{\mathcal{P}}} + \sigma^2 \mathbf{I})^{-1} \hat{\mathbf{h}}_{\mathcal{P}}^{\mathrm{LS}}, \tag{3.4}$$

where $\mathbf{h}_{\mathcal{P}}$ denotes a vector containing the elements of $\mathbf{h}$ with indices in $\mathcal{P}$ and $\mathbf{C}_{\mathbf{h}_{\mathcal{P}}}$ denotes the covariance matrix of $\mathbf{h}_{\mathcal{P}}$.

In practice, $\mathbf{C}_{\mathbf{h}_{\mathcal{P}}}$ and σ^2 are not known and need to be estimated from the received data. The noise level σ can be reasonably well known, however accurately estimating the covariance matrix $\mathbf{C}_{\mathbf{h}_{\mathcal{P}}}$, in particular, can be challenging in practice. Usually, the covariance matrix is estimated from previous T observed reference vectors. With many observed reference symbols, the covariance matrix estimate will become close to the ideal. However, this would not allow fast adaptation to changing (fast-fading) environments. Furthermore, with the 5G NR standard, the reference symbols are often provided specifically per user leading to an even larger need for the fast adaptation since the set $\mathcal{P}$ can change from one transmission to the next.

3.2 Channel Denoising Neural Networks

In the previous subsection, we described MMSE as a denoising channel estimator to make a natural connection with NNs that are used for denoising [53] in other areas, such as image denoising [54] and speech denoising [55]. Autoencoders are a feed-forward NN often used for explaining denoising in NNs [56]. Autoencoder consists of an encoder and decoder part. The encoder part is the deterministic mapping $f(\cdot)$ that transforms an n-dimensional input vector x into a hidden representation y. This is typically an affine mapping followed by a nonlinearity:

$$f(y) = s(Wx + b) \tag{3.5}$$

where W is a $d \times d$ weight matrix and b is an bias vector. The latent representation y is then mapped back (eventually after multiple hidden deep layers) to a reconstructed a m-dimensional vector z in the input space. This mapping is called the decoder part. The denoising autoencoder is a variant of the autoencoder. A denoising autoencoder is trained to reconstruct a clean repaired input from a corrupted/partially destroyed input. The performance of such a denoising autoencoder can be measured by checking the robustness to (partial) destruction of the input, where the output should yield a representation as close as possible, measured e.g. with mean squared error (MSE). The workings of such an denoising autoencoder is motivated by the following informal reasoning: "*a good representation is expected to capture stable structures in the form of dependencies and regularities characteristic of the (unknown) distribution of its observed input*" [53]. An autoencoder will learn a hidden latent-space representation and how to project noisy input to it. An example of such an NN is shown in Fig. 4.2, where a distorted channel estimate (in our case, the Least Squares estimate $\hat{\mathbf{h}}_{\mathcal{P}}^{\mathrm{LS}}$) is input into the NN and a denoised version is produced at the output. The mean squared error (MSE) between the NN output and the true channel frequency response $\mathbf{h}_{\mathcal{P}}$ is used to train the NN using backpropagation.

3.2.1 Fully-Connected Neural Network

Fully-Connected Neural Network (FC-NNs) is a class of machine-learning algorithms that work very well for a wide range of DL applications, in various fields. To be more specific, FC-NNs is the basic building block for the (denoising) autoencoder [53] and used in various channel estimation nets [9, 11, 10].

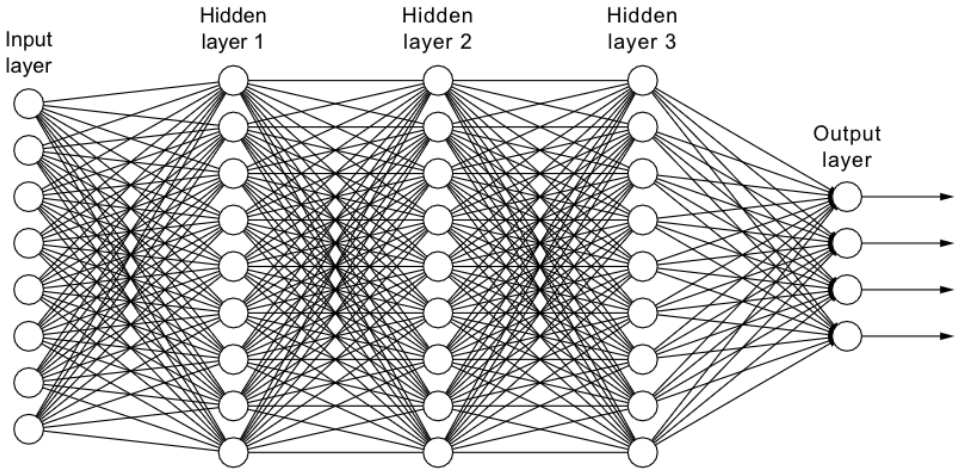

Figure 3.1: Example of a typical fully-connected neural network, with five FC-layers D in total. One input and one output FC-layer and three hidden FC-layers (image source: [57]).

An FC-NN consists of a series of D FC-layers, also referred to as the depth D of the network. A FC-layer is a function from $\mathbb{R}_N \Rightarrow \mathbb{R}_M$. Each output dimension of M depends on the input dimension N. The main computation within the FC-layers comprises a dot product with an added bias B_l. Usually, this is called the neuron, as shown in Figure (3.2) of the NN. for layer l we compute:

$$f_l(I) = B_l + (W_l^{\top} \cdot I_l) \tag{3.6}$$

were $I_l \in \mathbb{R}_N$ is the input vector, $W_l \in \mathbb{R}_{N \times M}$ the weight matrix, and $B_l \in \mathbb{R}_M$ the bias vector. Furthermore, a non-linear transformation in between layers is created by the use of activations functions such as: *sigmoid*, *tanh*, *ReLU*, *LeakyReLU* and *Swish* which are one of the most popular activation function. Using such an activation function enables a NN to learn approximations of complex functions. Depending on the requirements, activation function are applied to every output of a layers, or no activation function is used to keep the output linear.

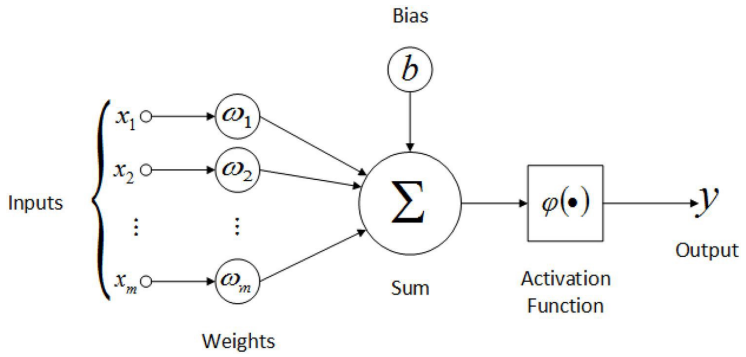

Figure 3.2: Example of a artificial neuron as typically used in neural networks, with multiple inputs M and one output (image source: [58]).

The compute complexity per layer l of such an FC-layer is often denoted as the number of multiply-accumulate (MAC) operations in a layer since the main computation is the dot product. We calculate the computational complexity without the activation function (since this negligible) and be calculated as follows:

$$\text{MACs per layer} = (I_l^N \times O_l^M) + O_l^M \tag{3.7}$$

were I_l^N is the input size of layer l, and O_l^M is the output size of layer l. The memory size of a layer l corresponds to the number of parameters which can be calculated as follows:

$$\text{Parameters per layer} = (I_l^N \times O_l^M) + O_l^M \tag{3.8}$$

The total amount of MACs and parameters in a model is respectively the sum of the MACs and sum of the parameters in each layer.

3.2.2 Convolutional Neural Network

CNNs gained a tremendous amount of interest since it outperforms humans in visual object recognition [2]. This enables many useful applications in a wide range of fields, but not limited to, from image and video recognition/classification/analysis/denoising/up-sampling, natural language processing, up to more specialized fields such as, e.g. channel estimation [12]. CNNs usually consists of two stages, as shown in Figure (3.3): a set of feature extraction layers, by applying a sequence of non-linear transformations to the input data. This is followed by a set of classification/decision layers which translates the resulting features into decisions. The feature extraction part usually consists of multiple Conv-layer(s) to extract the low-level features up to high-level features respectively from input data. Pooling layer(s) sub-sample the convolutional feature maps and extracts dominant features. The classification part usually consists of one or more FC-layers such as discussed in section (3.2.1), where the features are combined into decisions.

One downside of FC-NNs (introduced in section (3.2.1)) is that the FC-neurons in FC-layer l are connected to every FC-neuron in the next FC-layer l. Since every connection has a unique weight, the memory size of the model grows fast with increasing layer width W and increasing no. of layers D. This does not only increase the NN model size and computation but also training becomes more challenging and time-consuming. On the other hand, neurons in CNNs do only have local connectivity with a subset of neurons from the previous layer. Additionally, CNN-neurons are stored in feature maps, which share the same weights.

The primary operation in the Conv-layer, is the (discrete) convolution, the $*$ usually denotes the convolution.

$$o(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{3.9}$$

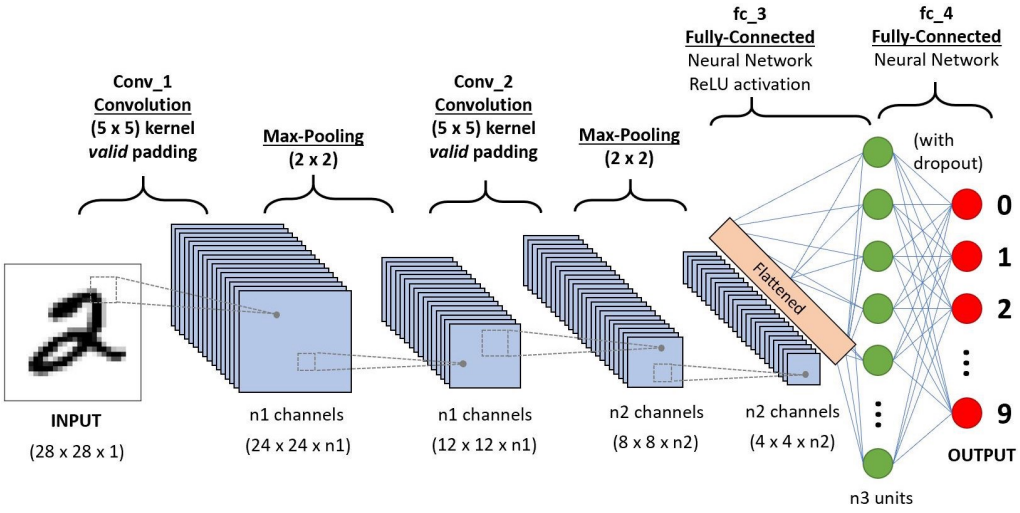

Figure 3.3: Example of a convolutional neural network as typically used in image classification. This model contains four feature extraction layers; two Conv-layers, two max-pooling layers. Furthermore in the classification layer two FC-layers are used (image source: [59]).

were the function x is the input, the function w is the kernel, and output o which is usually referred as the feature map.

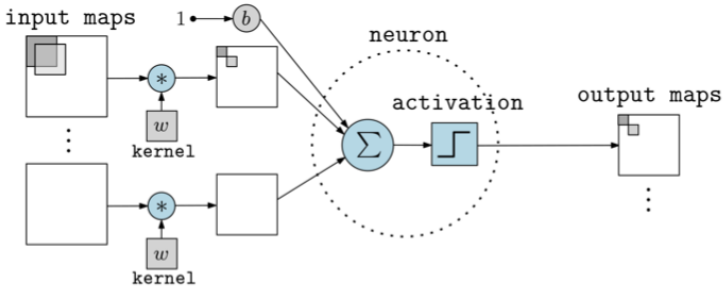

Figure 3.4: Example of a Conv-layer as typically used in convolutional neural networks (image source: [60]).

Usually in ML applications such as image and video recognition/classification but also channel estimation [12] the input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm. Finally, we often use convolutions over more than one axis at a time. For example, if we use a two-dimensional image I as our input, we probably also want to use a two-dimensional kernel K. The input-output relation of a Conv-layer l and a $KxKy$ Conv-filter K, and element $O_{x,y}$ then the output decisions can be computed as follows:

$$O_{x,y} = (I * K)(i,j) = \sum_m \sum_n I(i-m), j-n)K(m,n) \tag{3.10}$$

The compute complexity for a Conv-layer l can be calculated by counting the number of multiply-accumulate (MAC) operations. We calculate the compute complexity without the activation function (since this negligible), and be calculated as follows:

$$\text{MACs per layer} = (K_l^x \times K_l^y \times NF_l^m) \times (O_l^x \times O_l^y \times O_l^n) \tag{3.11}$$

with filter size K_l^x, K_l^y, no. of filters NF_l^m, output feature map size O_l^x, O_l^y and no. of output feature maps O_l^n in layer l. The total MACs of a CNNs, is the sum of the all the CONV MACs together with all the FC MACs in each layer. The memory size of a Conv-layer l corresponds to the number of parameters which can be calculated as follows:

$$\text{CONV parameters per layer} = ((K_l^x \times K_l^y + 1) \times NF_l^m) \times O_l^n \tag{3.12}$$

were the added 1 accounts for the bias parameters. When comparing Equation (3.11 and 3.12), it could be seen that computation in a CNN is usually $O_l^y \times O_l^n$ larger than the number of weights. Then the total amount of parameters in such a basic CNN is the sum of the total CONV parameters together with all the FC parameters.

3.2.3 Complex-Valued Neural Network

Fully-Connected Complex-Valued Neural Network (FC-CVNNs) are complex-valued counterparts to (real-valued) FC-NNs, where parameters are represented as complex numbers. These FC-CVNNs have been applied to various domains such as signal processing, communication systems, computer vision, in which complex numbers are often used through the Fourier transformation. Although these NNs are applied in various domains and research is done in parallel with real-valued NNs, results are marginal and they suffer from slow adoption. The main reasons are that there are only a few known methods for regularization and hyper-parameter optimization specifically developed for complex neural networks. Furthermore, popular machine learning frameworks such as CNTK, Tensorflow, Theano, PyTorch are lacking support for training complex NNs.

Similar to FC-NNs, CVNNs consist of a series of D complex-valued-layers also referred as the depth D of the network. A complex-valued-layer is a function from $\mathbb{C}_N \Rightarrow \mathbb{R}_M$ or $\mathbb{C}_N \Rightarrow \mathbb{C}_M$. Each output dimension of M depends on the input dimension N. The main computation within the CV-layers comprises a dot product with an added bias B_l i.e. for layer l we compute:

$$f_l(I) = B_l + (W_l^\top \cdot I_l) \tag{3.13}$$

were $I_l \in \mathbb{C}_N$ is the input vector, $W_l \in \mathbb{C}_{N \times M}$ the weight matrix, and $B_l \in \mathbb{C}_M$ the bias vector. In comparison to the real-valued activations, different functions are used which is a function $\mathbb{C}_N \Rightarrow \mathbb{R}_M$ or $\mathbb{C}_N \Rightarrow \mathbb{C}_M$. To deal with the complex-valued representation, various different activation functions have been proposed in the literature. Most popular activation functions are the Complex ReLU/tanh ($\mathbb{C}$ReLU, $\mathbb{C}$tanh). With this complex activation function applies separate ReLU/tanh function on both the real and imaginary part of the neuron as:

$$\mathbb{C}Act(z) = Act(\Re(z)) + iAct(\Im(z)) \tag{3.14}$$

Were Act is one of real-valued activation functions e.g. ReLU, LeakyReLU, tanh, Sigmoid.

In comparison to real-valued NNs, CV-NNs have an increased computational complexity since complex parameters introduce more operations. Instead of a single real-valued multiplication, up to four real multiplications and two real additions are required. The compute complexity per layer l of such an FC-CV-layer in MACs activation function (since this is negligible) can be calculated as follows:

$$\text{MACs per layer} = 4 \times (I_l^N \times O_l^M) + 2 \times O_l^M \tag{3.15}$$

with input size I_l^N, and output size O_l^M of layer l. The memory size of a layer l corresponds to the number of complex parameters which can be calculated as follows:

$$\text{Parameters per layer} = 2 \times (I_l^N \times O_l^M) + 2 \times O_l^M \tag{3.16}$$

The total amount of MACs and parameters in a model is respectively the sum of the MACs and sum of the parameters in each layer.

3.2.4 Binary Neural Networks

The FC-NNs, CNNs, and CVNNs can contain a massive amount of MACs and 32-bit floating-point parameters, that requires a massive amount of floating-point operations. As discussed in Section (2.3.3), quantization is usually an effective approach to reduce memory size and computation complexity on existing NNs, without performance degradation. In some cases, the floating-point/fixed-point model weights, biases and/or activation are quantized and constrained to one bit only $\{-1, +1\}$, without a significant performance decrease. In this way, it only takes 1-bit to store a (kernel) weight value, while the computation, floating-point or integer dot-product/convolution can be simplified [61]. In the extreme case, both weights and activations can be quantized to the binary representation. In this way the main computation within a FC-layer or Conv-layer (the dot product (3.6)), can be implemented by efficient bitwise *XNOR* for the multiplication, and *bit-count* for the addition [61]. These types of networks with binary operations are usually called Binary Neural Networks (BNN) and Binary Convolutional Neural Networks (BCNNs). While for regular quantized NNs architecture does not need to be changed, BNNs To prevent a large accuracy drop, typically in BNNs, the first and last layer is kept full precision (32-bit floating-point or fixed-point). Batch-normalization is a technique that stabilizes and accelerates the training process and is, therefore, an essential technique to achieve higher accuracy with BNNs [62]. But this comes at a downside of storing extra parameters and an extra non-binary addition. Furthermore, usually wider and deeper networks are used in-comparison to their full precision counterparts.

The main computation in BNNs and BCNNs is the dot product. When only the weights are quantized to one bit, this dot product keeps unchanged in comparison to FC-NNs and CNNs, as shown in Eqn (3.6). Although smaller hardware multipliers can be used [61]. When the activations are also quantized the main computation for layer l can be computed as follows:

$$f(I_l) = \text{bitcount}(\text{XNOR}(W_l^\top, I_l)) + B_l \tag{3.17}$$

were $I_l \in \mathbb{B}_N$ is the input vector, $W_l \in \mathbb{B}_{N \times M}$ the weight matrix, and $B_l \in \mathbb{R}_M$ the bias vector. The benefit of binarizing the activations as well, is that activations such as *sigmoid* and *tanh* can be replaced by a *Sign* function [63], which is a compare, as shown in Eqn (3.18).

$$\mathbf{A}ct(\cdot) = Sign(x) = \begin{cases} +1 & if \mathbf{x} \geq \mathbf{c} \\ 1 & otherwise \end{cases} \tag{3.18}$$

Chapter 4

Channel Denoising Neural Networks

The related work Chapter (2) discussed the many different types of NN based OFDM receivers, which replace only parts or the whole OFDM Rx. These related works have shown that replacing smaller parts does not necessarily lead to a performance decrease of the receiver, but can decrease the memory size and computational complexity. The previous Chapter (3) has shown that channel estimation in the OFDM Rx is one of the most computationally complex parts. Traditional channel estimation algorithms such as MMSE perform better than algorithms such as Least Squares but are highly complex due to the estimation of the statistics. In this Chapter, we, therefore, propose our hybrid NN based OFDM receiver who replaces only this specific part. Chapter (2) has shown that NN architectures and configuration sizes are not only important for performance but can massively reduce the complexity e.g. with BNNs. Although the most popular architectures in communications are the FC-NN and CNN [9, 11, 10, 12], it is unknown which and if these are among the best-performing ones. Furthermore, since it is unknown if other architectures such as CVNN and BNN are suitable for our application, and what the impact is on memory and computation. In this section, we introduce therefore a generic pilot channel denoising architectures, based on four NN architectures (FC-NN, CNN, CVNN, and BNN). Each of these introduced NN architectures will be evaluated in Chapter (6). Table (4.1) and (4.2) provides an overview of all the tuneable parameters of these architectures and their naming.

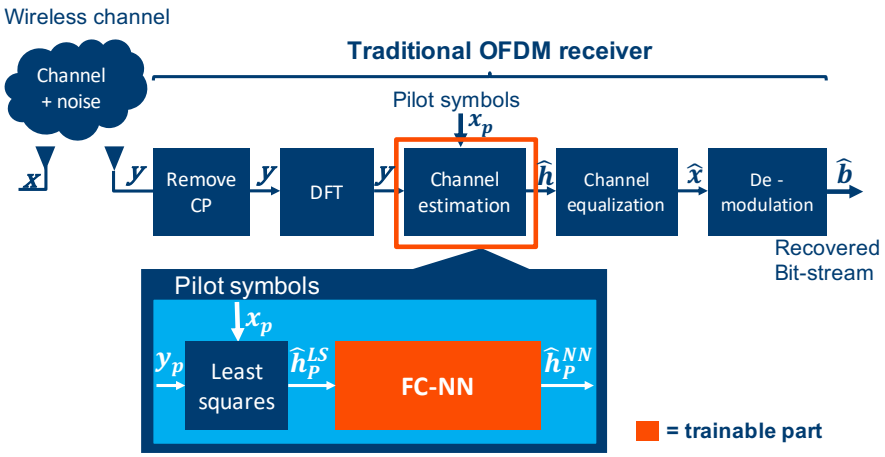

Figure 4.1: Example of an OFDM receiver with our NN-assisted channel estimation, where the red part denotes the trainable NN.

4.1 Generic Denoising NN Architecture

The previous related work solutions we discussed in Section (2.1) consider denoising across frequency, time or both [9, 11, 10, 12], and often across multiple antennas. This will heavily increase the complexity thus for simplicity and to enable a faster design-space exploration, this work focus only on denoising in the frequency dimension, but the methodology we follow can also be applied verbatim to more complex scenarios. Furthermore, frequency and time filtering can typically be performed separably [52]. To reduce further the complexity, in this work we only denoise the pilot channel symbols P. These pilot channel symbols are extracted according to the LS solution which is noisy. The NN denoises this pilot LS solution and hereafter the data channels can be estimated with classical methods(these steps are shown in Figure (4.1)). Pilot only denoising is typically done in e.g. MMSE solution (Section (3.1.4)) and can reduce the number symbols to be denoised up to 6 times (data symbols versus pilot symbols ratio can be up to 1-6 depending on the scenario) and thus reduce the computation. We expect with pilot only denoising the input of the NNs can be smaller (up to 6 times) and thus reduces the NNs complexity, in comparisons when input will also include data with e.g. [12] where both data and pilot symbols are denoised.

We start with a generic denoising autoencoder which is typical for denoising NNs, as introduced in Section (3.2). This generic denoising autoencoder has a residual structure, such a structure can enable estimating the noise characteristics often also required. The prediction of the regression with our model denoted by $f(I)$, can then be computed as:

$$y = f(I_1) - I_1 \tag{4.1}$$

Since there are no clear-cut NN design guidelines in the related literature and to minimize the computational complexity and memory size, we define and explore different NN architectures, in the following subsections based on the residual structure as proposed above. It is not exactly known how these network topography should look like inside the residual structure. According to the Universal Approximation Theorem, one hidden layer should be enough, to represent any function. But in real-world applications, NN with more hidden layers, perform better. The reason for this, is probably that the currently available training methods are unable to find the values of the parameters that corresponds to that function, and/or that the training finds the wrong function as a result of overfitting [64]. Furthermore, a NN with multiple hidden layers is a composition of several simpler functions, which are easier to learn, and could potentially achieve higher performance. Therefore, in the following subsections, we define four generic denoising NNs based on the previously introduce residual structure. This generic definition allows us to explore different topologies with a design-space exploration to find the best performing models and other optimized model topologies for every target.

4.1.1 Fully-Connected Neural Network

The first proposed generic NN topology is based on the FC-NN architecture with the addition of the residual structure (shown in Figure (4.2)). Each such a model configuration is defined by the NN input size P (no. of input pilot symbols), no. of FC-layers D, the FC-layer width W, and the type of activation function Act. These parameters can be adjusted to construct a model configuration for any target.

4.1.2 Fully-Connected Complex-Valued Neural Network

The second proposed generic NN topology is based on the FC-NN (Section (4.1.1)), where the real-valued layers are replaced by complex-valued FC-layers. Each such a model configuration is defined by the NN input size P, no. of complex-valued FC-layers D, the complex-valued FC-layer width W, and the type of complex activation function Act. These parameters can be tuned to construct a model configuration for any target.

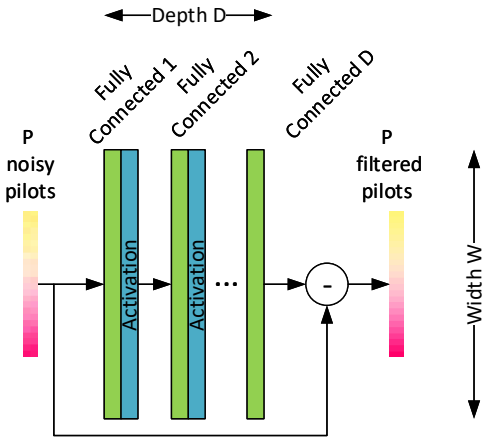

Figure 4.2: Real-valued, Fully-Connected Residual NN architecture with a variable P input pilot symbols, no. of FC-layers D and FC-layer width W.

4.1.3 Convolutional Neural Network

The third proposed generic NN topology is based on the CNN architecture with the addition of the residual structure (shown in Figure (4.3)). Each such a CNN model configuration is based on the FC-NN (Section (4.1.1)) but split into two separate parts: Conv-part and FC-part, which is typical for CNNs. The Conv-part consists of Conv-layers and is defined by P input pilot symbols, no. of Conv-layers D_{Conv}, the Conv-layer width W_{Conv}, the no. of filters NF, the filter kernel size K, the filter stride S, the padding size Pd and the type of activation function ACT_{Conv}. The FC-part is defined by the no. of FC-layers D_{fc}, the FC-layer width W_{fc}, and the type of activation function ACT_{fc}. These parts and parameters can be independently adjusted to construct a model configuration for any memory and computation target.

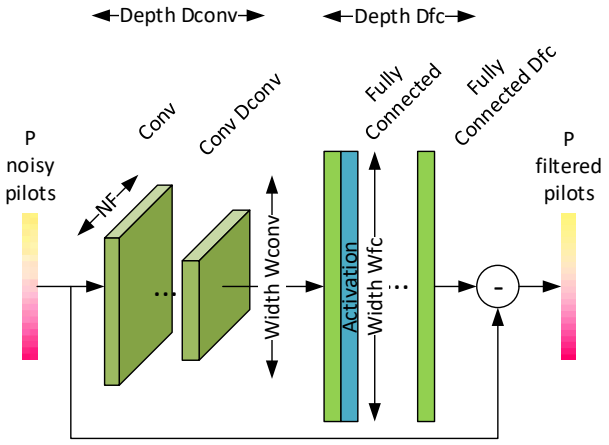

Figure 4.3: Real-valued, 1D-Convolutional Residual NN architecture with P input pilot symbols and with a variable no. of Conv-layers D_{conv}, Conv-layer width W_{conv}, no. of filters NF, the filter kernel size K, the filter stride S, the padding size Pd and the type of activation function ACT_{Conv}, no. of FC-layers D_{fc}, FC-layer width W_{fc} and FC-layer activation ACT_{fc}. Used for channel estimation de-noising.

4.1.4 Binary Neural Network

The fourth proposed generic NN topology is derived from the FC-NN (Section (4.1.1)), but the hidden layers are replaced by binary layers (shown in Figure (4.4)). To prevent a large accuracy

drop due to these low-precision layers, the first and last layer is kept in full precision (FP), as usual for BNNs. Each such a model configuration is defined by the NN input size P, the FP-layer input/output width W_{fp}, and the type of activation function Act. The binary part is defined by the no. of binary layers D_b and binary layer width W_b. These parts and parameters can be independently adjusted to construct a model configuration for any target.

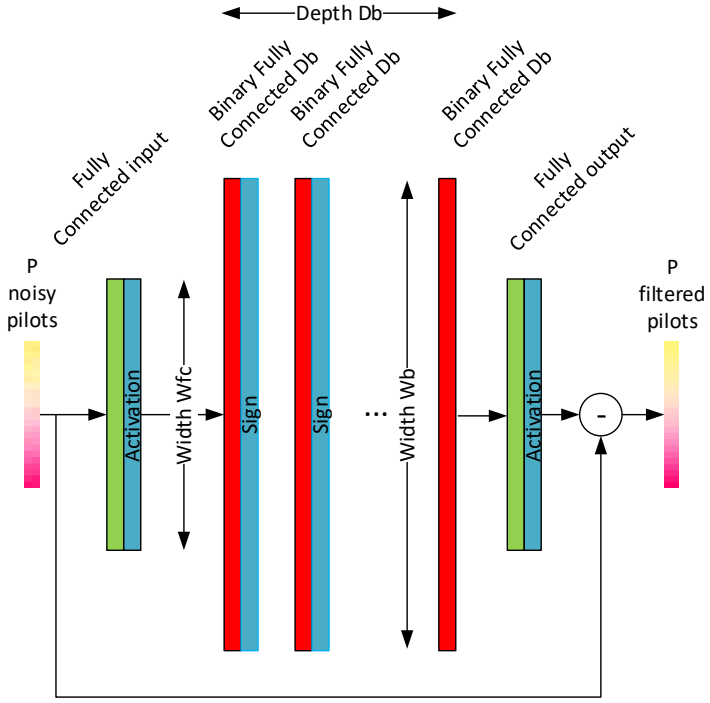

Figure 4.4: Generic binary Fully-Connected Residual NN architecture with a variable P input pilot symbols and FP-layer width W_{fp}, type of activation function ACT, no. of binary layers D_b and binary layer width W_b.

Neural Network Architecture	**Tunable Parameters**
Fully Connected Neural Network	P, D, W, Act
Complex-valued Fully Connected Neural Network	P, D, W, Act
Convolutional Neural Network	$P_{Conv}, D_{Conv}, W_{Conv}, NF, K, S, Pd,$ $Act_{Conv}, D_{fc}, W_{fc}, Act_{fc}$
Binary Neural Network	P, W_{fp}, Act, D_b, W_b

Table 4.1: Tunable parameters for each generic denoising neural network architecture as divined in the above sections.

Paramter	Meaning
P	Input/output layer width ($P/2$)= no.of input pilots
D, D_{fc}	No. of fc-layers
W, W_{fc}	Layer width offc-layer
Act, Act_{fc}	Activation function in fc-layer
D_{Conv}	No. of Conv-layers
W_{Conv}	Layer width of Conv-layer
NF	Number of filters
K	Kernel size (x,y,z)
S	Filter stride
Pd	Filter padding
Act_{Conv}	Activation function in Conv-layer
Db	No. of Binary-layers
Wb	Layer width of Binary-layer

Table 4.2: Meaning of the tunable parameters form the generic denoising neural network architectures from the above sections.

Chapter 5

Optimizing Channel Denoising Neural Networks

Chapter 4 presents the four proposed generic NN architectures that can be used for pilot channel denoising in an OFDM wireless communication system. Untrained neural network models are unable to denoise the pilot symbols, we introduce our dataset which is used to train the models together with the training method. Hereafter we introduce our framework for systematic design-space exploration. This framework supports all the channel denoising NNs architectures, which were introduced in Chapter (4).

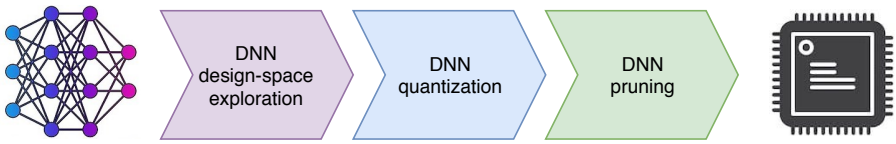

Figure 5.1: DNN optimization flow.

This methodology consists of three main steps:

1. Finding optimized NN model configurations for the NN architectures introduced in Chapter (4).
2. Finding optimized fixed-point quantized NN approximations of the optimized NN model configurations from step 1.
3. Finding optimized pruned NN of the quantized NN from step 2.

The goal is to reduce the NN memory size and computational complexity and to provide a Pareto-optimal performance-complexity trade-off for a range of targets. To perform robustly in different SNR scenarios, we introduce an optimization method for wide SNR ranges, which can be be used on top of this exploration. Followed by the introduction of an extra memory and computation complexity reduction step. The results of this design-space exploration are provided in Chapter (6).

5.1 Dataset

For the training dataset, a random multipath channel dataset is generated. The number of distinct paths is chosen uniformly at random in $\{1, \ldots, 30\}$, the (normalized) path delays are chosen uniformly at random in $[0, 1]$, and the path gains are chosen uniformly at random from 0 dB to -50 dB. The (non-normalized) delay spread for our randomly generated channels is chosen uniformly at random from 10 ns to 1000 ns similarly to the 6 GHz channels provided by the

3GPP [65]. The validation set contains another set of similarly generated random channels that is distinct from the training set and is also expanded with the aforementioned 3GPP channels to demonstrate the robustness and generalization power of the optimized denoising NNs which are obtained with the proposed framework. The pilot symbols have unit power such that the SNR is defined as:

$$\text{SNR} = \frac{1}{\sigma^2}. \tag{5.1}$$

5.2 Training

The goal of the NN model is to denoise the noisy channel frequency response at the pilot positions. Figure 4.1 has shown that the input to the score function (our NN model) is the noisy pilot channel frequency response vector $\hat{\mathbf{h}}_{\mathcal{P}}^{\text{LS}}$ and the target output value is the noiseless channel frequency response vector $\mathbf{h}_{\mathcal{P}}$ (both created within the above discussed dataset). The score function (our NN) is trained to minimize the difference between the output (estimated channel frequency response vector) and the noiseless channel frequency response vector. The difference can be portrayed in several ways. In our experiment a weighted $L2$ loss function is used, also called weighted mean squared error (MSE) which is defined as:

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N} ||(f(\hat{\mathbf{h}}_{\mathcal{P}}^{\text{LS}}; \mathbf{\Psi}) - \mathbf{h}_{\mathcal{P}}) \times \text{SNR}(n)||_2^2, \tag{5.2}$$

where f denotes the function that the trained NN implements, $\mathbf{\Psi}$ are all the trainable parameters of the NN, N is the number of training samples, $\text{SNR}(n) = \frac{1}{\sigma(n)}$, and $\sigma(n)$ is the noise variance used to generate the n-th training sample. The scaling factor $\text{SNR}(n)$ is used to achieve a robust performance over a wide SNR range with a single NN. For training of the NN model with the (w)MSE loss function with backpropagation is used. In a preliminary exploration it was found that the RMSprop [66] optimization function is was the best performing one in terms of training loss and training speed. Table (5.1) summarizes all the training parameters.

Parameter	**Value**
SNR	0-30*
Loss function	(w)MSE
Epoch	100, 100, 100, 100, 100, 100
Batch size	256, 512, 1024, 2048, 4096, 8192**
Initial learning rate	0.01
Optimizer	RMSprop
ρ	0.85
Regularization	None ***

Table 5.1: If not mentioned differently the above training parameters were used. * SNR is depending on the scenario. ** For BNNs other batch size is used. *** For shared layer regularization is used.

5.3 Optimizing NN Model Configurations

In Chapter (4) we have discussed that it is unknown what the best performing NN model architecture and configurations size is for our NN assisted channel denoising approach. This is mainly due to the lack of comparable related work. In this Section, we will, therefore, introduce the first step of the automatic design-space exploration methodology. The goal is to find Pareto-optimal model configurations for all the NN architectures introduced in Chapter (4) (FC-NN, CVNN, CNN, and BNN). This should provide the performance-complexity trade-offs between the above-mentioned architectures and between each model configuration size. This should result in a set of models for each budget and scenario.

5.3.1 Fully-Connected (Complex-Valued) Neural Network

To find the best trade-offs between memory size, computation, and denoise performance we explore different model configurations for both the FC-NN and FC-CVNN architectures which were introduced in Section (4.1.1) and (4.1.2). This exploration is done through a structured grid-search method and will result in a set of different model configurations for each architecture. We iteratively change one of the models hidden FC-layer width (W) with step size (W_{step}), no. of FC-layers (D) with step size D_{step} for each type of activation function (Act). This procedure is illustrated in Algorithm (1). While using the same FC-layer width W for each hidden layer is pessimistic, we prevent the design-space by growing exponentially. We expect to compensate this with the pruning method, which will be introduced in Section (5.5). Each created FC-NN or FC-CVNN model configuration in this design-space exploration is initialized and trained from scratch and its memory size in KB, computation in MACs are used for complexity requirements while the denoise performance is measured in SNR.

5.3.2 Convolutional Neural Network

In Section (4.1.3) the channel denoising with a CNN is introduced, with the advantage that it needs to store fewer parameters in comparison to the FC-NN and FC-CVNN. Just like for the previously discussed architectures (Section (5.3.1)), no knowledge is available on the optimal model configuration. Therefore a comparable design-space exploration is executed to find Pareto-optimal model configurations for every target. CNNs consist mainly of two parts: the Conv-part and the FC-part. The FC-part consist of the same parameters as the FC-NN (Section (5.3.1)): FC activation Act_{fc}, FC-layer width W_{fc}, and no. of FC-layers D_{fc}. While the Conv-part consist the parameters: Conv activation Act_{Conv}, Conv-layer width W_{Conv}, no. of Conv-layers D_{Conv}, no. of filters NF with filter kernel size K, with filter stride S, and filter padding Pd. Due to the many parameters in the Conv-layers part, the design-space grows arbitrary large in comparison with the FC-NN model. Even when fixing parameters K, S, Pd, and using uniform layer widths for both the Conv- and FC- layers, the design-space becomes unfeasible for the time being. Therefore only

Parameters	Description
P	Input/output layer width ($P/2$)= no.of input pilots
D	No. of FC-layers
D_{step}	Step size for adding layers
W	Uniform hidden layer width
W_{step}	Step size for increasing layer width
Act	Activation function

Table 5.2: FC-NN and FC-CVNN configuration size design-space exploration parameters.

one model size will be explored with fixed parameters.

Algorithm 1: Design-space exploration of FC-NN and CVNN model configurations for the parameters: Act, D, W. The models that are created and trained every new iteration are denoted by *model*. SNR is the performance of that particular NN configuration. $create\_new\_model(\cdot)$ initialise a new FC-NN model configurations, $train(\cdot)$ denotes the function that the trained NN implements with training data: $\hat{\mathbf{h}}_{\mathcal{P}}^{\mathrm{LS}}, \mathbf{h}_{\mathcal{P}}$, loss function: $MSE(\cdot)$ and hyperparameters: Ω. When finishing the exploration a set of trained model configurations and its performance metrics is returned.

$Act \leftarrow \{activations(\text{user selected types})\}$;
$D_{step} \leftarrow \{depth\_step\_size(\text{user selected types})\}$;
$W_{step} \leftarrow \{width\_step\_size(\text{user selected types})\}$;
$models \leftarrow \{\emptyset\}$;
$SNRs \leftarrow \{\emptyset\}$;
for *each act* $\in Act$ **do**
  while $D \leftarrow 1$ *to* x **do**
    $D \leftarrow D + D_{step}$;
    while $W \leftarrow 0$ *to* x **do**
      $W \leftarrow W + W_{step}$;
      $model \leftarrow create\_new\_model(Act, D, W)$;
      $model, SNR \leftarrow train(model, \hat{\mathbf{h}}_{\mathcal{P}}^{\mathrm{LS}}, \mathbf{h}_{\mathcal{P}}, MSE(\cdot), \Omega)$;
      models.$insert(model)$;
      SNRs.$insert(SNR)$;
      if $SNR < max(SNRs) - T_{snr}$ **then**
        break;
      end
    end
  end
end
return *models*, *SNRs*

5.3.3 Binary Neural Network

In Section (4.1.4) the BNN architecture is introduced, and like the previous NN model architectures, the optimal BNN model configuration is unknown. Therefore a comparable design-space exploration as in Section (5.3) is used with some minor modifications is proposed.

BNNs usually consist of a full-precision first and last layer, and binary hidden layers. We expected that due to the limited precision of the binary layers, these layers should be very wide and deep, compared to the FC-layers in FC-NNs, to hold a competitive performance. Therefore the full-precision layer widths W_{fp} and binary layer widths W_b are independently adjusted. The layer width between the layers is uniform to reduce the design-space as done in Section (5.3) for the FC-NNs. For the BNNs, we iteratively change the model FP-layer width (W_{fp}) with step size W_{FPstep} for each type of activation function (Act). Furthermore, we iteratively change the binary-layer no. of layers (D_b) with step size D_{FPstep} and binary-layer width (W_b) with step size W_{Bstep}. Each created BNN model configuration in the design-space exploration is trained from scratch and its memory size in KB, computation in MACs and denoise performance in SNR are used for evaluation. This method for BNNs is illustrated in Algorithm (2).

Parameters	Description
P	Input/output layer width ($P/2$)= no.of input pilots
W_{FP}	Uniform hidden layer width
W_{FPstep}	Step size for increasing layer width
D_b	No. of FC-layers
D_{b_step}	Step size for adding layers
W_b	Uniform hidden layer width
W_{b_step}	Step size for increasing layer width
Act	Activation function

Table 5.3: BNN configuration size design-space exploration parameters.

Algorithm 2: Design-space exploration of BNN model configurations for the parameters: Act_{fc}, W_{fp}, D_B and W_B. The models that are created and trained every new iteration are denoted by *model*. SNR is the performance of that particular BNN configuration. $create\_new\_model(\cdot)$ initialise a new BNN model configurations, $train(\cdot)$ denotes the function that the trained NN implements with training data: $\hat{\mathbf{h}}_{\mathcal{P}}^{\mathrm{LS}}, \mathbf{h}_{\mathcal{P}}$, loss function: $MSE(\cdot)$ and hyperparameters: Ω. When finishing the exploration a set of trained model configurations and its performance metrics is returned.

$Act \leftarrow \{activations(\text{user selected})\}$;
$D_{b\_step} \leftarrow \{activations(\text{user selected})\}$;
$W_{b\_step} \leftarrow \{activations(\text{user selected})\}$;
$W_{FPstep} \leftarrow \{activations(\text{user selected})\}$;
$models \leftarrow \{\emptyset\}$;
$SNRs \leftarrow \{\emptyset\}$;
for *each* $act \in Act$ **do**
  while W_{fp} **do**
    $W \leftarrow W + W_{FPstep}$;
    while D_b **do**
      $D \leftarrow D + D_B$;
      while W_B **do**
        $W_b \leftarrow W_b + W_{Bstep}$;
        $model \leftarrow create\_new\_model(Act, W_{fc}, D_b, W_b)$;
        $model, SNR \leftarrow train(model, \hat{\mathbf{h}}_{\mathcal{P}}^{\mathrm{LS}}, \mathbf{h}_{\mathcal{P}}, MSE(\cdot), \Omega)$;
        models.$insert(model)$;
        SNRs.$insert(SNR)$;
        if $SNR < max(SNRs) - T_{snr}$ **then**
          break;
        end
      end
    end
  end
end
return *models*, $SNRs$

5.4 Optimizing by Fixed-point Quantization

Quantization is a commonly used approach to reduce the memory footprint and computational complexity and power consumption of an NN and to support fixed-point hardware (accelerators). This has lead to impressive memory size reduction for many NN architectures, as discussed in Section (2.3.3). The goal of the NN quantization procedure in this work is to find good quantized fixed-point approximations for the pre-trained floating-point models found in the first step of our optimization approach (Section (5.3)) for a set pre-defined bit-widths. Furthermore, the goal is to maximally reduce the memory size and computational requirements, with minimal performance loss.

In this work, we re-train the set of Pareto-optimal FC-NN, CVNN and CNN configurations obtained from the first step described in Section (5.3) with the addition of quantization for the selected pre-defined bit-widths. We re-training floating-point models since this results in higher performance when compared to quantized models trained from scratch [67]. By choosing the Pareto-optimal NNs, we reduce our design-space, while still holding a set of optimized models for a broad range of targets.

We quantized both the model weights, biases and activations in each layer, of the FC-NN, CVNN, and CNN. A deterministic, asymmetric uniform affine quantization approach is used, to quantize a set of floating-point variables $\mathcal{X}$ to a set of fixed-point variables $\mathcal{X}_Q$ with a bit-width Q. Each element $x \in \mathcal{X}$ is quantized as follows:

$$x_{int} = round(\frac{x}{\Delta}) + z \tag{5.3}$$

$$x_Q = \text{clamp}(x_{\text{int}}, 0, 2^Q - 1) \tag{5.4}$$

we define our scale factor as:

$$\Delta = \frac{\max(\mathcal{X}) - \min(\mathcal{X})}{2^Q - 1}. \tag{5.5}$$

were:

$$clamp(a, b, x) = \begin{cases} a & \text{if } x \leq a \\ x & \text{if } a \leq x \leq b \\ b & \text{if } x \geq b \end{cases} \tag{5.6}$$

de-quantization operation is:

$$x_{float} = (x_Q - z) \times \Delta \tag{5.7}$$

Since the derivative of the quantization function is zero almost everywhere, quantization approximations are required or backpropagation. Therefore the straight-through estimator as proposed by Courbariaux et al. [68] is used.

Motivation for Deterministic, Asymmetric Uniform Quantization

Recent works have shown that deterministic quantization outperforms stochastic quantization [67], while simplifying the quantization approach. For these reasons we, use a deterministic quantization approach. It has been shown that FC-NN and CNN do not need a large bit-width to maintain good performance, but that the limited dynamic range of fixed-point numbers may be problematic [69, 70]. Therefore to maximize the performance, we always take the min, max range of the floating-point values as given by Equation (5.5). According to our dynamic range analysis, we found that we could get the same precision with fewer bits using an asymmetric quantization scheme. This is probably due to the asymmetric distribution of the NN model weights and bias parameters as illustrated in Figure (5.2). While asymmetric quantization brings computation overhead, compared to symmetric quantization, Benoit et al. [71] has shown that this overhead can be reduced to only one extra fixed-point multiplication which is negligible.

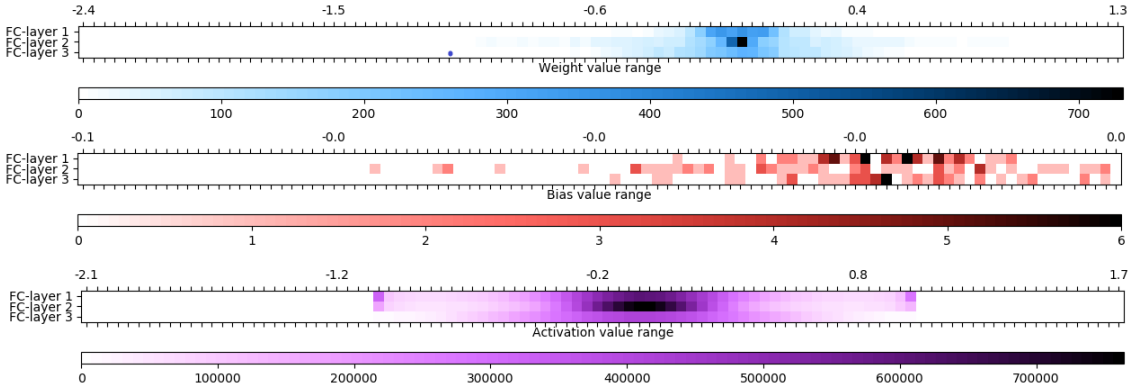

Figure 5.2: Layer-wise parameter weight (top), bias (middle) and activation data (bottom) distribution for a Pareto-optimal FC-NN model with configuration: $\{W = 64, D = 3, Act = \tanh\}$.

Parameter-wise, Layer-wise, Quantization

As illustrated in the range analysis in Figure (5.2), the NN model parameters (weights, biases, and activations) in each layer and between each parameter usually have a significantly different range and distribution. Thus, uniform quantization with a fixed-point data format for all the layers and parameter types may incur (great) performance loss. For the same bit-width Q, smaller parameter ranges can be represented with higher precision than larger parameter ranges as shown in Figure (5.3). By allowing distinct ranges Δ and bit-widths Q for each layer and parameter type, we can reduce the quantization error, at a small cost of storing three scaling factors per layer. We note that this overhead is taken into account in the model memory sizes in Section 6.

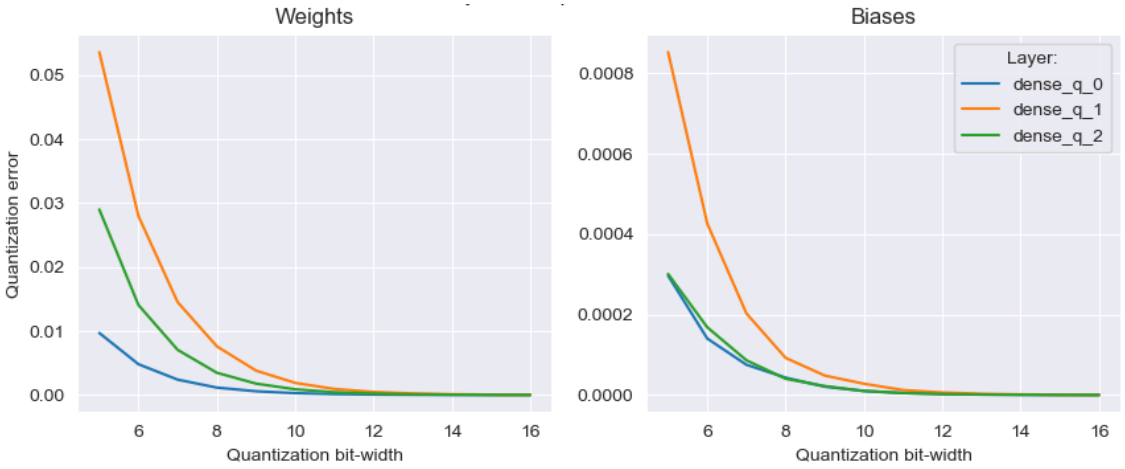

Figure 5.3: Quantization approximation error per bit-width Q, per layer D for the weights (left) and for the bias (right) for a Pareto-optimal FC-NN model with configuration: $\{W = 64, D = 3, Act = \tanh\}$.

Channel-wise Quantization

One problem of layer-wise quantization is that one range is used for the whole weight matrix, whose parameter distribution might be huge. The same problems arise as in layer-wise quantization but on a more fine-grained scale, as can be seen in Figure (5.4). This can be avoided by applying the quantization separately to each weight channel (each matrix row), which results in a lower error at the same bit-width, in comparison with layer-wise quantization, as could be seen in Figure (5.5).

Lower quantization can be archived for the same performance. The trade-off is better quantization accuracy, but at the cost to store two variables per channel: scaling factor and bit size.

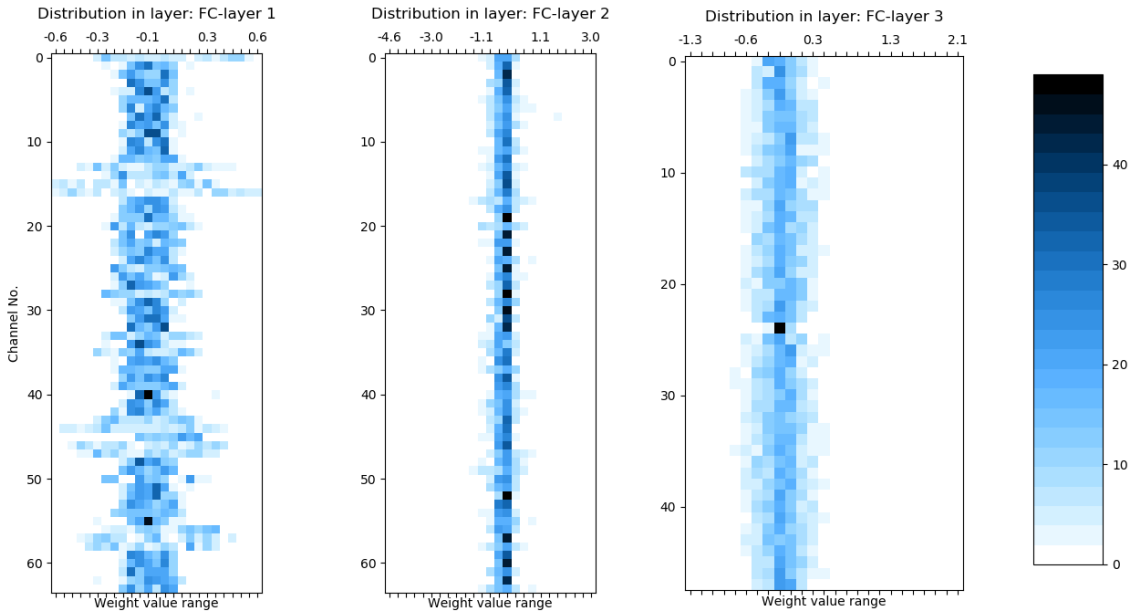

Figure 5.4: Channel-wise weight parameter distribution for a Pareto-optimal FC-NN model with configuration: $\{W = 64, D = 3, Act = \tanh\}$.

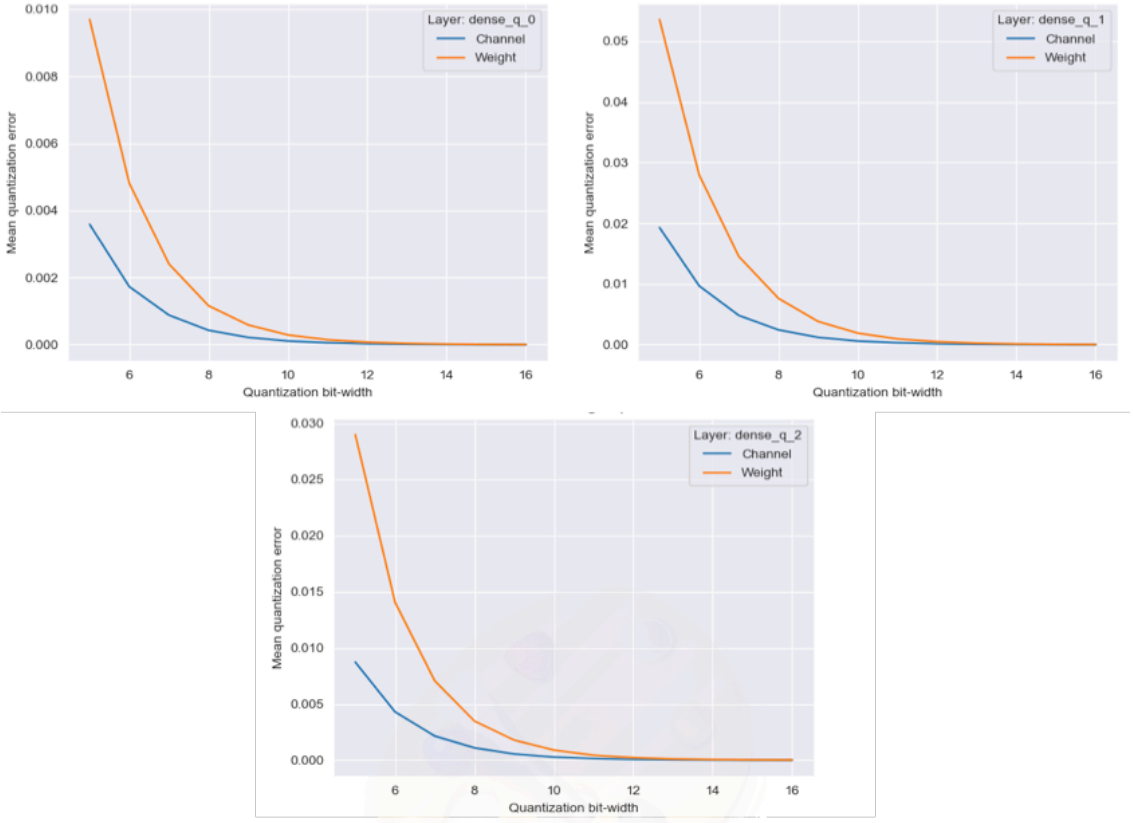

Figure 5.5: Mean error per bit-width Q for channel quantization on a Pareto-optimal FC-NN model with configuration: $\{W = 64, D = 3, Act = \tanh\}$.

5.5 Optimizing by Neuron Pruning

In an attempt to further reduce both memory size and computational complexity on top of quantization, we also apply a neuron pruning approach based on the average percentage of zeros (APoZ) proposed in [16]. In this work we re-train only the set of Pareto-optimal quantized NNs configurations obtained from the quantization step described in Section (5.4) with the addition of APoZ neuron pruning, to keep the design space size reasonable. The goal of neuron pruning is not only to heavily reduce the model memory size and computational complexity, but also to compensate for the pessimist uniform NN layer widths introduced in Section (5.3), and to find multiple models for each target. While other more fine-grained pruning methods exist such as weight pruning, we avoid these to keep the design-space reasonable.

The APoZ is defined as the percentage (or, equivalently, fraction) of zero-valued neuron activations of a particular neuron over a set of N input data. Let $O_c^{(l)}(n)$ denote the output of neuron c in layer l for input sample n. Then, the $\text{APoZ}_c^{(l)}$ of neuron c in layer l is defined as:

$$\text{APoZ}_c^{(l)} = \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}\left(O_c^{(l)}(n) = 0\right), \tag{5.8}$$

where $\mathbb{I}(\cdot)$ is an indicator function defined as:

$$\mathbb{I}(\cdot) = \begin{cases} 1 & \text{if } true \\ 0 & \text{if } false \end{cases} \tag{5.9}$$

All neurons in the NN for which $\text{APoZ}_c^{(l)}$ is smaller than some threshold t are pruned. The threshold t is gradually increased in steps of 1% and pruning is followed by retraining to compensate for any performance degradation. This results in a new set of Pareto-optimal quantized and/or pruned model configurations, where each model is an optimized version of the generic de-noise model. Based on these results, an optimized model configuration for a requirement can be chosen. The trade-off is the model-size, versus computation versus performance. We present the results of the optimization strategies; compact model configuration size, quantization and pruning in Chapter (6).

5.6 Optimizing NNs Across a Wide SNR Range

The parameters of NN models used for (pilot) channel denoising and channel estimation generally depend on the SNR [24, 12], and supporting a large range of SNRs is an important requirement in wireless communication systems as discussed in Section 2.2. The goal is, therefore, to optimize these NN models further, to perform robustly across a predefined wide SNR range, while at the same minimizing memory and computational requirements of this (wide SNR) NN solution.

Training a (single) NN model on a dataset that includes a large predefined SNR range results in a robust performance over the entire range. However, in this case, the NN needs to learn a rather complex function that requires a relatively large (memory size) and high-complexity (computation) NN, which is hard to train and to archive optimal performance. A NN model covering a smaller range only needs to learn a simpler function, could, therefore, be smaller, and thus easier to train. Therefore the model can potentially achieve higher denoising performance for the training SNR, while at the same time being smaller and lower-complexity since only a single small NN will be active at any given time. The downside of this approach is that the NN will only perform robustly close to the training SNR and multiple (different) NN models and their parameters have to be stored if the desired SNR range is large. In a pose to perform not only robust around this single SNR, a third approach is introduced, where the set of single SNR trained NN models is retrained to perform robustly on a wider SNR range for the same model size. It is therefore expected that the performance of such a Kth model will perform more robust on the range, and thus improve the overall performance on the total predefined range for the same model size. The following three

approaches are used in ou design-space exploration to demonstrate these two strategies through two approaches:

- ***Wide SNR range***

  A single NN is trained and evaluated on a dataset that includes randomly sampled SNRs in a predefined desired range $[\mathrm{SNR}_{\min}, \mathrm{SNR}_{\max}]$. During deployment only one model should be stored.

- ***Split SNR ranges***

  A set of K models is trained, each on a single SNR. Specifically, model $k \in \{0, \ldots, K-1\}$, is used in the range $[\mathrm{SNR}_{\min}+k\mathrm{SNR}_{\mathrm{step}}, \mathrm{SNR}_{\min}+(k+1)\mathrm{SNR}_{\mathrm{step}}]$,
  where $\mathrm{SNR}_{\mathrm{step}} \triangleq \frac{1}{K}(\mathrm{SNR}_{\max}-\mathrm{SNR}_{\min})$. The training SNR is chosen as the midpoint of each range, while the validation is done in the total predefined range $[\mathrm{SNR}_{\min}, \mathrm{SNR}_{\max}]$. Finally, when these models are deployed, all NN parameters are stored and depending on the measured SNR of the signal, only the appropriate NN is selected and inferenced for denoising.

- ***Optimized Split SNR ranges***

  A set of K pre-trained models, each trained on a single different SNR according to the *Split SNR ranges* approach above. We start with a single SNR $\mathrm{SNR}^K_{\min} = \mathrm{SNR}^K_{\max}$ for each K model, and increasing the training SNR range of each K model $\left[\mathrm{SNR}^K_{\min}, \mathrm{SNR}^K_{\max}\right]$ with a step size $SNR_{Tstep} + SNR_{Tstep}$: $\mathrm{SNR}^K_{\min} - SNR_{Tstep} = \mathrm{SNR}^K_{\max}$ and retrain the model. This is iteratively repeated until no performance increase is obtained on the total predefined working range of all the K models together $[\mathrm{SNR}_{\min}, \mathrm{SNR}_{\max}]$. Although this will probably decrease the denoise performance on the single SNR, it is expected that the average SNR will be increased on the whole desired range of interest $[\mathrm{SNR}_{\min}, \mathrm{SNR}_{\max}]$.

5.6.1 Layer and Channel Weight Sharing

In a pose to compensate for the increased memory size of the *Split SNR ranges* method introduced in Section (5.6), we make use of the shared capabilities between the set of K models. By sharing one or multiple layers and/or channel weights between K distinct models, the memory size requirements can be reduced. We hypothesize that these K distinct models probably learn/encode the same initial features in the (first) layer(s). To enable one or multiple shared layers between K distinct models, the training algorithms should be changed, and contains the following three steps:

1. The first set of K models is trained at the same time, with L shared layers between these models. The training SNR is chosen as the midpoint of each range according to the criteria given in Section (5.6), and the training set contains an equal amount of training SNRs for each Kth model. Since these K distinct models are trained with L shared layers, at the same time we see it as one model, depending on the no. of L shared layers, therefore it contains L common inputs, and outputs, for each SNR midpoint. To deal with these changes the training loss function is changed.

2. Due to switching between the training SNRs for each midpoint (output), we freeze the parameters of the shared layer (its weights and bias cannot be trained). Furthermore, we freeze each partials model With this method we hypothesize that by fine-tuning each partial model (output) can compensate for the shared fixed (layer), and perform maximally.

3. The rest of the Kth model(s) that don't share a layer with the other models, is trained according to the standard routine in Section (5.2) and Section (5.6.

The total loss functions for the shared layer networks is a weighted mean squared error (MSE) between the $k \in \{0, \ldots, K-1\}$ NN output and the channel frequency response for the P pilot carriers which is defined as:

$$\mathrm{MSE}_L = \frac{1}{N} \sum_{n=1}^{N} ||(f(\hat{\mathbf{h}}_{\mathcal{P}}^{\mathrm{LS}}; \mathbf{\Psi}) - \mathbf{h}_{\mathcal{P}}) \times \mathrm{SNR}_L(n)||_2^2, \tag{5.10}$$

where f denotes the function that the trained NN implements, $\mathbf{\Psi}$ are all the trainable parameters of the NN, N is the number of training samples, $\mathrm{SNR}(n) = \frac{1}{\sigma(n)}$, and $\sigma(n)$ is the noise variance used to generate the n-th training sample. The scaling factor $\mathrm{SNR}(n)$ is used to achieve a robust performance over a wide SNR range with a single NN.

Chapter 6

Results and Evaluation

In this section, we present and interpret the results of our design-space exploration as introduced in Chapter (5). First, a brief overview of our evaluation approach is discussed. Followed by the results of the NN model configuration exploration, quantization and pruning for both the *Wide SNR range* FC-NN and *Split SNR ranges* FC-NN architectures. Hereafter we finish with the design-space exploration and comparison for the other NN architectures; BNN, CNN, and CVNN for both the *Wide SNR range* and *Split SNR ranges* approach.

6.1 Evaluation Approach

To evaluate the introduced systematic design-space exploration methodology and its effectiveness concerning memory and computation complexity reduction, and the complexity reduction impact of quantization and pruning we will systematically evaluate the following matters:

1. First we evaluate the NN model configuration design-space exploration for both the *Wide SNR range* and *Split SNR ranges* approach, for the FC-NN architecture. We show that the grid-search based design-space exploration is a good technique to find models for a given target.

2. In the second step we evaluate the NN model quantization design-space exploration executed on the generated Pareto-optimal floating-point NN model configurations found in the previous step. The denoise performance and model memory size is compared to their floating-point counterpart. We show that quantization together with the exploration is a good approach to reduce the memory size of these NN models and that these models are robust to quantization.

3. NN model pruning of the Pareto-optimal fixed-point quantized NN models is evaluated, by comparing the denoise performance, model size, and computational complexity to their floating-point and quantized counterpart. These results show that pruning is an effective approach to further reduce the memory size and computation complexity on-top of quantization.

4. After the third step, the above-mentioned evaluation steps (1-3), are executed for the *Split SNR ranges* approach. It is found that these combinations of *Split SNR* trained models to perform better than *Wide SNR range* while reducing computation complexity.

5. Furthermore, we evaluate the proposed NN layer and channel sharing for the FC-NN *Split SNR ranges* approach while comparing it with the non shared (layer/channel) equivalent. We show that this is an extra step in the design-space exploration without substantial improvement.

6. The above-mentioned evaluation steps (1-4) are applied to the other proposed NN architectures (BNN, CVNN, and CNN), and compared with the FC-NN architecture in each step. We show that FC-NN can obtain better solutions when compared to the other architectures.

We note that our goal is not to directly compare our results to the related work but rather to compare the various complexity-reduction methods in a well-defined and fully-controlled environment against our baseline solution and two widely used classical solutions. Moreover, the works of [9, 11, 10, 12] consider a different setting than our work and the methodology that we explore can also be applied verbatim to these works.

6.2 Evaluation Metrics

We use three performance metrics for the selection of the obtained NN configurations:

- The model size in KB. For sake of clarity, this includes weight and bias parameters and scaling parameters for quantization.
- We also report the computational complexity of each obtained configuration by counting the required number of MAC operations to perform one denoising operation. The MAC operations for the NNs activation functions is not included since it is negligible compared to the matrix MACs.
- The denoising gain is measured in dB concerning the LS solution, which is defined as:

$$G = 10 \log_{10} \frac{\|\hat{\mathbf{h}}_{\mathcal{P}}^{\mathrm{NN}} - \mathbf{h}_{\mathcal{P}}\|_2^2}{\|\hat{\mathbf{h}}_{\mathcal{P}}^{\mathrm{LS}} - \mathbf{h}_{\mathcal{P}}\|_2^2}. \tag{6.1}$$

  For the denoise performance gain evaluation two scenarios are considered:

  - For the design-space explorations and robustness evaluation a set of 10.000 samples are created and used according to the channel estimation scenario as introduced in Section (5.1). Since this set of similar generated data is distinct from the training-set and includes extra 3GPP channels we directly demonstrate the robustness and generalization power of the obtained NNs.
  - For the wide-range performance plots again the same channel estimation scenario as introduced in Section (5.1) is used. But to simplify and speed up evaluation, only one 3GPP channel TDL-A (instead of multiple random and 3GPP channel) is used, with 50 channel samples per SNR steps, and steps of 1dB are used. With this scenario, we can demonstrate the robustness and generalization power of the obtained NNs for the whole SNR range.

  We consider SNRs ranging from 0 dB to 30 dB, which are practically relevant for a modern OFDM-based system that uses adaptive modulation and coding. Each OFDM symbol contains 72 data symbols and 24 pilot symbols.

6.3 DSE Settings

A preliminary exploration was used to determine the ranges of W and D. The maximal range was set when no denoising performance increase was obtained. Moreover, for the step size multiples of higher values are explored to reduce the design space, as time-consuming training is required to obtain the denoising performance of each considered NN architecture. For sake of clarity, for the CNN, only one model configuration is explored for time reasons, as discussed in Section 4.1.3.

For the quantization bit-width, we consider $Q \in \{4, 5, 8, 10, 12, 16, 32\}$ in order to cover commonly supported bit-widths and some additional in-between values.

For the pruning design-space exploration, a threshold t of 1% is considered, and models are only re-trained if the denoise performance drops below of the previous pruned configuration. All the Pareto-optimal models are pruned until their performance dropped 1dB below of the original non-pruned quantized NN model.

Parameter	Range
Fully-Connected NN	
W	32 - 160
W_{step}	32
D	2 - 6
D_{step}	1
Act	tanh, ReLU
Complex-Valued Fully-Connected NN	
W	16 - 96
W_{step}	16
D	2 - 6
D_{step}	1
Act	$\mathbb{C}$tanh
Convolutional NN	
W_{Conv}	32
D_{Conv}	5
NF	64
K	$[L1^*, L3^* = 3 \times 1, L2^*, L4^* = 5 \times 1]$
S	1
Pd	no
Act_{Conv}	tanh
Act_{fc}	tanh
Binary NN	
W_{fp}	32 - 160
$W_{fp_s tep}$	32
W_b	128 - 2048
$W_{b_s tep}$	128
D_b	1-5
$D_{b_s tep}$	1
Act	tanh, ReLU

Table 6.1: Neural network model configuration size design-space exploration parameter ranges. $^*L\# =$ layer no.

6.4 DSE of Neural Network Configuration

Figure (6.1) and Figure (6.2) show the results of our design-space exploration only for the FC-NN configuration (i.e., not considering quantization or pruning yet) for the cases where a single NN (*wide SNR trained*) and $K = 3$ distinct NN models (*split SNR trained*) are used for the SNR range of interest, respectively. The settings for the FC-NN model configuration are defined in Section (6.3), while the training settings from Table (5.1) are used.

We observe that the model size versus performance trade-off curve is steep in both cases and levels off quickly after some model size. This shows that it is crucial to carefully select an appropriate NN model configuration to maximize the denoising gain for a given model size constraint or to minimize the model size and computation for a given denoising gain target.

When comparing Figure (6.1) and Figure (6.2) (single NN versus $K = 3$ distinct NNs), we also observe that, as expected, the $K = 3$ distinct NNs achieve higher denoise gains compared to the single NN, especially for lower SNRs than at high SNRs due to the limiting effect of the additive Gaussian noise at low SNRs. The average performance of the single NN is limited by the performance at low SNRs. On top of this, we observe that for the single NN performance gain drops faster after some model size, compared to the $K = 3$ distinct NNs.

For the next two Section (6.4) and (6.4) we consider for the single SNR approach (*wide SNR trained*), a Pareto-optimal model configuration of (*Act*=ReLU, $W = 128$, $D = 5$). For the $k = 3$ distinct models (*split SNR trained*) we consider a Pareto-optimal model configuration of (*Act*=tanh, $W = 64$, $D = 3$) for each model, and is summarized in Table (6.2). These model configurations are found from the design-space exploration in Figure (6.1) and Figure (6.2) and are models from the Pareto frontier just before the performance quickly drops.

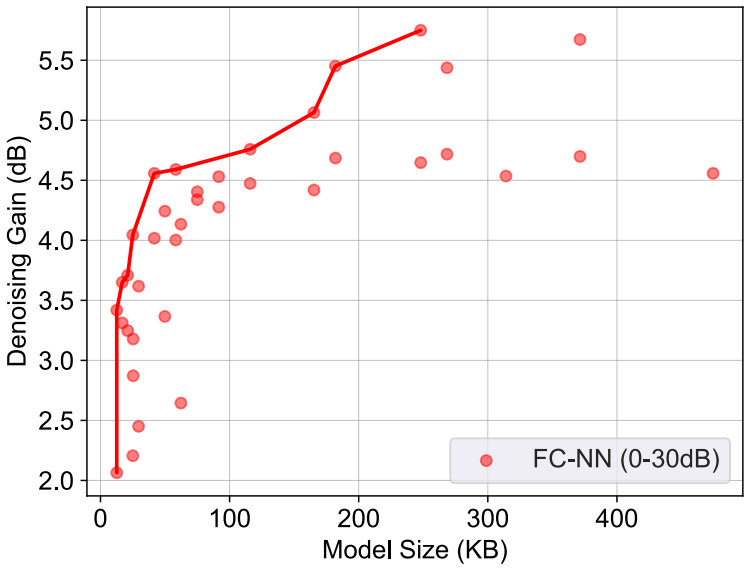

Figure 6.1: Design-space exploration where each point represents a single FC-NN (*wide SNR trained*) that is used for the entire SNR range 0 - 30 dB. The line represents the Pareto frontier.

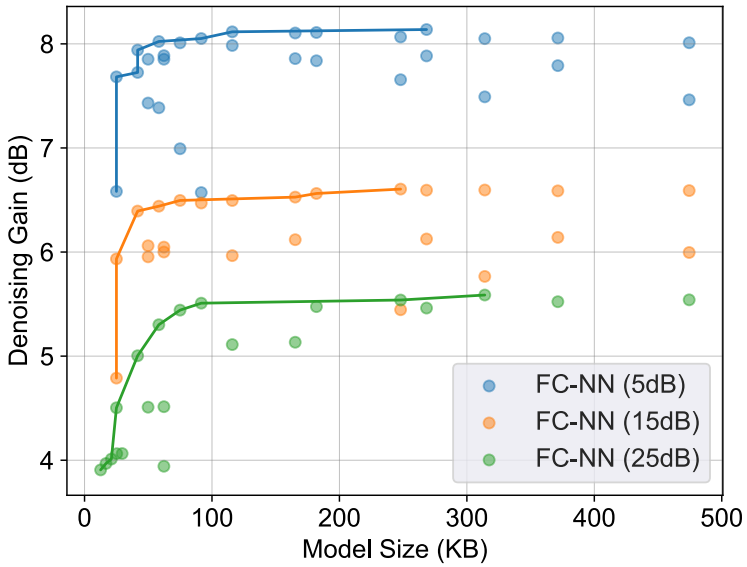

Figure 6.2: Design-space exploration where each point represents three distinct FC-NNs (*split SNR trained*) that are used for the SNR range 0 - 30 dB. The lines represent the Pareto frontiers for each FC-NN.

Evaluation Across Wide SNR

Figure (6.3) provides an overview of the wide range denoise performance overview of the Pareto-optimal selected single NN and $K = 3$ distinct NNs. As expected, the single SNR model performs robust on the whole SNR range of interest.

While for $K = 3$ distinct NNs, each Kth-model performs maximally on the specific SNR it was trained on ($kSNR_{step}$). The denoise performance difference on these points can go up to 3dB when compared to the single *wide SNR* trained model. For higher and lower SNRs than $kSNR_{step}$, the denoise performance has an increasing decline from that point. By making use of this capability e.g. multiple of these distinct NNs models can be used to perform robustly over the whole range for any performance target. By the used $K = 3$ distinct models, their denoise performance overlaps on the points just before the performance drops below the single SNR model. In Appendix (A.1) we show the results of this effect for different values of K. The setup with $K < 3$ does not have enough overlap such that the performance in the middle drops below the single SNR model, which results in a lower performance than $K > 2$. While for $K > 3$ the average performance rises, since these models overlap each other just before the performance quickly drops. The results for different values of K (with $K = 1$ corresponding to the single SNR range NN), for the selected Pareto-optimal model configuration, are summarized in the second column of Table 6.5, where we show the average denoising performance over the entire range of interest. Moreover, for $K > 1$, we report the worst-case number of MACs for the largest NN and the sum of the model sizes of all K NNs. We observe that, without quantization or pruning, using $K = 4$ results in a 1.08 dB better denoising gain with 83% fewer MACs and a 33% smaller model size compared to the $K = 1$ case.

Evaluation with Classical Solution

Further, we have compared the denoise performance of our proposed methods with that of two widely used baseline algorithms LS and MMSE.

The LS has the worst performance since no prior statistics of the channel has been utilized, and is set as our lower bound. On the contrary, the ideal MMSE, were channel statistics are given

instead of estimated from the received data, has the best performance and gives an upper bound of the achievable mean squared error. This solution is not practically feasible since the covariance matrix should be fully known without error, which is not a valid assumption in practical applications. The estimated MMSE is a practically feasible method and which includes the estimation of the correlation matrix, based on received data samples. It is shown in Figure (6.3) that our approach, which does not use any statistics, performs much better than the LS method and the MMSE up to 200 samples. In the case of a static channel and when there are more than 200 samples available to estimate the correlation matrix, the estimated MMSE outperforms the NN solution. As a result, the NN approach will be better performing at the start and for dynamically changing channels which occurs whenever there is movement in the real world.

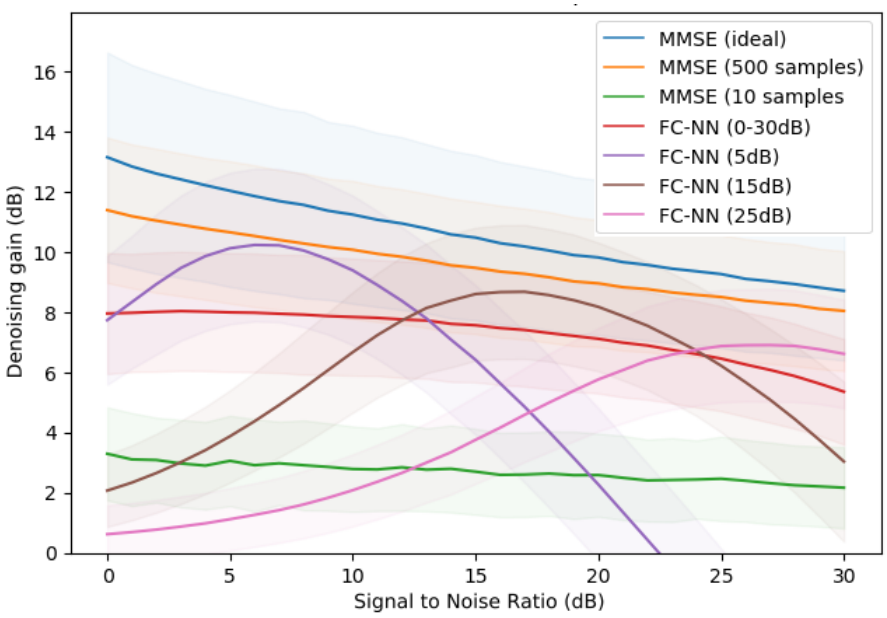

Figure 6.3: Denoise performance over a wide range of SNR for carefull selected FC-NN Pareto-optimal single SNR model, three distinct models, ideal MMSE, MMSE with 10 and 500 samples used for the channel statistics.

Paramter	**Value**
Wide SNR range **(K=1)**	
W	128
D	5
Act	ReLU
Wide SNR range **($K > 1$)**	
W	64
D	3
Act	tanh

Table 6.2: Selected Pareto-optimal model configurations which are used through the report for evaluation and comparison of the FC-NN model.

6.5 DSE of Fixed-point Quantization

In Figure (6.4) and Figure (6.5) we present the quantization design-space exploration only for the Pareto-optimal FC-NN configuration (i.e., not considering pruning yet) found in the first design-space exploration step. We consider only the cases where a single FC-NN (*wide SNR trained*) and $K = 3$ distinct FC-NNs (*Split SNR trained*) are used for the SNR range of interest, respectively. The settings for the FC-NN quantization are defined in Section (6.3), while the training settings from Table (5.1) are used. Since our approach uses parameter-wise and layer-wise quantization, it is not guaranteed that all the parameters in all the layers have the same bit-width. Reporting all the bit-width settings for each parameter and in each layer for every model will be difficult to read. Therefore for all the quantization results are the worst-case results, were the reported bit-width is that of the highest one in the model. It should be noted that while it is expected that the proposed channel-wise quantization, would further decrease the model memory size, these results are not presented. Since quantizing and exploring each bit-with for each channel massively increases the design-space. Due to time restriction we were not able to execute the all the steps to make a fair evaluation and comparison.

As expected and observed in the previous step, Figure (6.4) and Figure (6.5) show that careful selection is needed to pick the best model for a target. As an example: for some Pareto-optimal models, minor performance improvement of 0.1dB can be achieved, while memory size and/or computation can grow up to 5 times. Furthermore, it is observed that for all of the Pareto-optimal models $K = 3$ distinct NNs perform better, than a single model, while being smaller in both size and computational complexity. Again as observed in Section (6.4), the performance of smaller models drops faster with single SNR models than for the $k = 3$ distinct models.

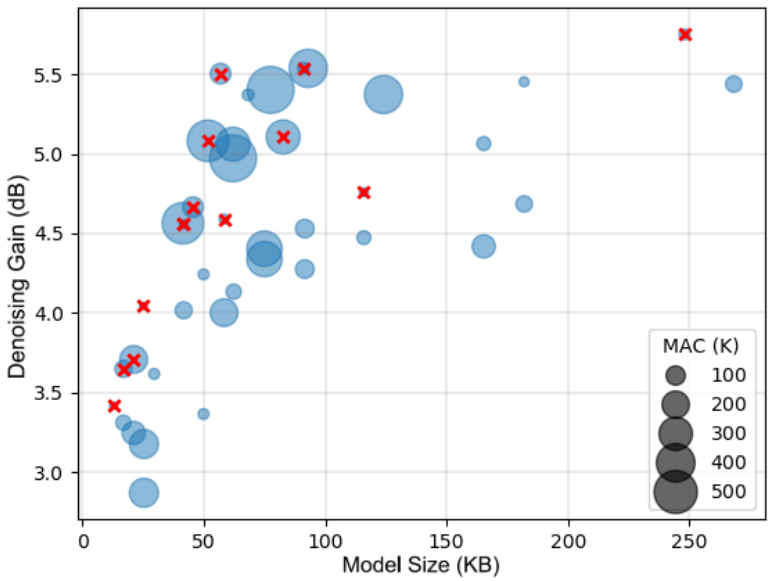

Figure 6.4: Design-space exploration where each bubble represents a single FC-NN (*wide SNR trained*) that is used for the entire SNR range from 0 dB to 30 dB. The crosses represents the Pareto frontier.

In Figure (6.6) we illustrate the denoising gain as a function of the quantization bit-width for the cases where a single NN and $K = 3$ distinct NNs are used for the SNR range of interest. For this evaluation, we use again the Pareto-optimal NN configurations for each case, as chosen in Section (6.4) with the settings from Table (6.2). We observe that all considered NNs are quite robust to quantization down to a bit-width of $Q = 10$ bits, while a significant performance degradation starts appearing for $Q = 8$ bits. Interestingly, all NNs have very similar robustness concerning quantization, although the single NN seems to suffer from a slightly larger loss when

going from $Q = 32$ to $Q = 10$. For $K = 3$ distinct NNs a bit-width of 16, 12, 10, results only decrease of 0,0, 0.02, and 0.01 dB in denoise gain, while reducing the memory size with respectively 50%, 62.5%, 68.75%. The same trend is denoted for the single model.

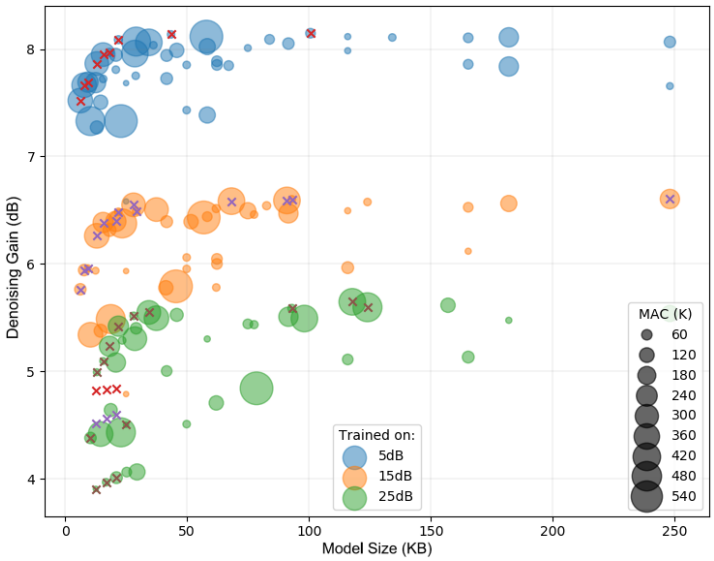

Figure 6.5: Design-space exploration where each bubble represents one of the three distinct FC-NNs (*split SNR trained*) that is used for the entire SNR range from 0 dB to 30 dB. The crosses represents the Pareto frontier for each FC-NN.

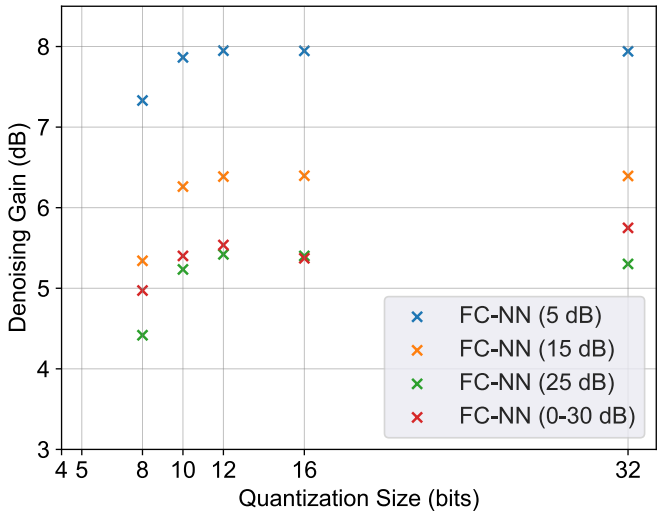

Figure 6.6: Effect of quantization with different bit-widths on smaller models trained on single SNR (*split SNR trained*) and a big model (*wide SNR trained*) range of 0-30dB.

Optimizing across wide SNR

Figure (6.7) provides the denoise performance on a predefined range of one selected single NN configuration (*wide SNR trained*) and $K = 3$ distinct NNs (*split SNR trained*) configurations Pareto-optimal 10-bit quantized models (NN configurations as previously selected in Section (6.4). When comparing the quantized models with the LS and MMSE we observe the same trend as discussed in Section (6.4). When we comparing the 10-bit quantized *sid SNR trained* models with their Non-quantized counterparts, no performance loss is noticed at low SNRs up to 17dB, while for the higher SNRs minor performance loss is noticed. This is probably due to the more complex functions for high SNRs, who require more precision. Comparing $K = 3$ distinct quantized NNs (*split SNR trained*) with their counterparts, no performance decrease is noticed on the $kSNR_{step}$ point, while a slowly increasing minor performance loss is noticed on other SNRs. This is not a problem since the noticeable mostly occurs when the models overlap each other. Thus we show that the single SNR model is overall robust to quantization, while $K = 3$ distinct NNs are more robust to quantization due to their overlapping.

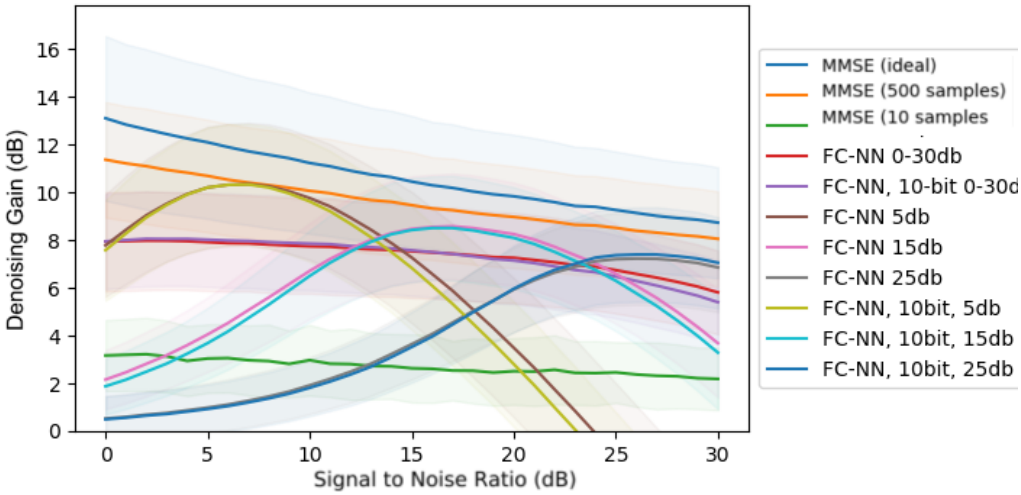

Figure 6.7: Denoise performance of picked FC-NN Pareto-optimal (quantized) single SNR model (*wide SNR trained*) and (quantized) $k = 3$ distinct models (*split SNR trained*) and ideal MMSE, MMSE with 10 samples for ACF and 500 ACF samples.

These results indicate that quantization is an efficient technique to massively decrease the FC-NN model size up to 10-bit while reducing the memory size up to 68.75%, with almost no noticeable performance increase. Furthermore, Table (6.5) indicates that using multiple K distinct 10-bit quantized models can further reduce the model size by almost 50% and reduce the computation by 1/3.

6.6 DSE of Neuron Pruning

In Figure (6.8) and Table (6.5) we present the results of our pruning strategy. These models are pruned according to our proposed method from Section 5.5, with the pruning settings as defined in Section (6.3), while the training settings from Table (5.1) are used. It is shown that a combination of three Pareto-optimal model configurations (quantized and pruned), trained according to the *Split SNR* approach, performs always better than a single *Wide SNR ranges* model for the same model-size and/or computation, and for the same performance, three models are always smaller. But compared to quantization only from the previous Section (6.5), the differences are smaller.

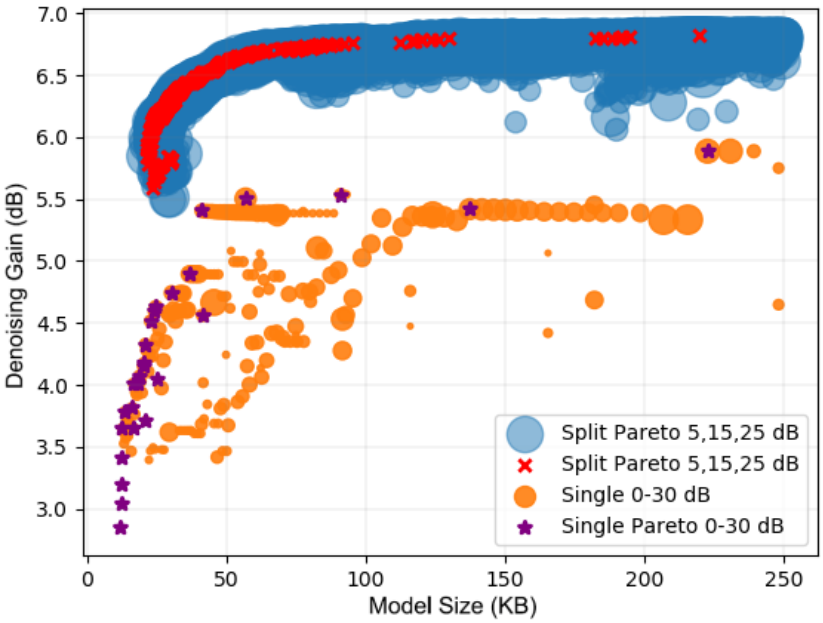

Figure 6.8: Design-space exploration of pruned single *Wide SNR ranges* FC-NN (orange bubble) versus combined *Split SNR ranges* $K = 3$ distinct FC-NN (blue bubble) that is used for the entire SNR range 0 - 30 dB. The purple and red stars represents the Pareto frontier respectively for the *Wide SNR ranges* model and *Split SNR ranges* models.

To show the robustness of pruning on these models we present Figure (6.9). It is shown that these models are robust to pruning (on top of 10bit quantization) and that each of the $K = 3$ distinct models (*split SNR*) can be pruned up to 18%, while the single *wide SNR* model can be pruned far more up to 58% without performance degradation. In total this has resulted that for the *split SNR* approach, the total model size can be reduced by 17.8% (39KB to 32KB), while the computation can be reduced by 16% (from 10416 MACs to 8542 MACs). For the *wide SNR* model approach, the memory can be reduced by 46.5% (77.5KB to 41.44KB) and computation by 57% (62000 MACs to 33159 MACs). Thus the non-split model outperforms the single model approach in both memory and computation requirements while achieving a higher denoise performance.

Optimizing across wide SNR

In Figure (6.10) we show performance on the SNR range of interest for a specific selected Pareto-optimal NN configuration for $K = 3$ distinct models (*split SNR trained*) and the single model (*wide SNR trained*). The same Pareto-optimal NN configuration is used as in the previous section, from Table (6.2). But these models are pruned on top of quantization. It can be clearly seen, that pruning for our chosen threshold only has minor noticeable performance loss over the whole range. We, therefore, can conclude that pruning is an effective way to further reduce both memory size and compute complexity on-top of an quantized Pareto-optimal model configuration

for our channel denoising application.

In Table (6.5) we show the final results of our design-space exploration configuration size, quantization and pruning when after every step a Pareto-optimal point is chosen. It is shown that in total this has resulted that for the *split SNR* approach, the total model size can be reduced by 74.375% (41.44KB), while the computation can be reduced by 16% (from 10416 MACs to 8542 MACs). For the *wide SNR* model approach, the memory can be reduced by 86.87% (32.6KB) and computation by 57% (62000 MACs to 33159 MACs). This shows the the effectiveness of these approaches, and when compared

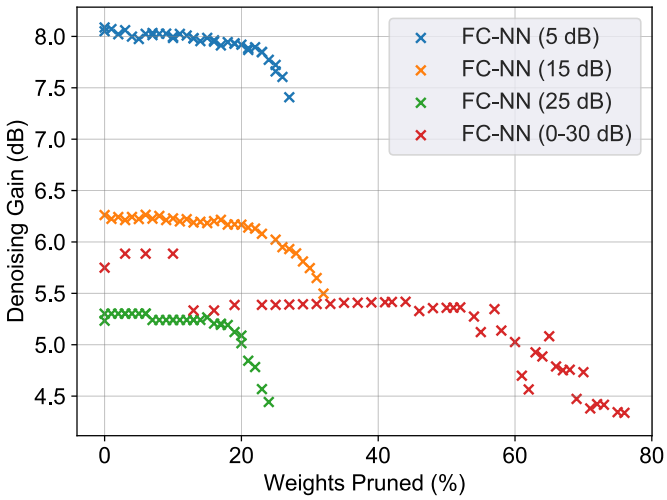

Figure 6.9: Performance of Pareto-optimal NNs after neuron pruning for different thresholds t resulting in different percentages of pruned neurons.

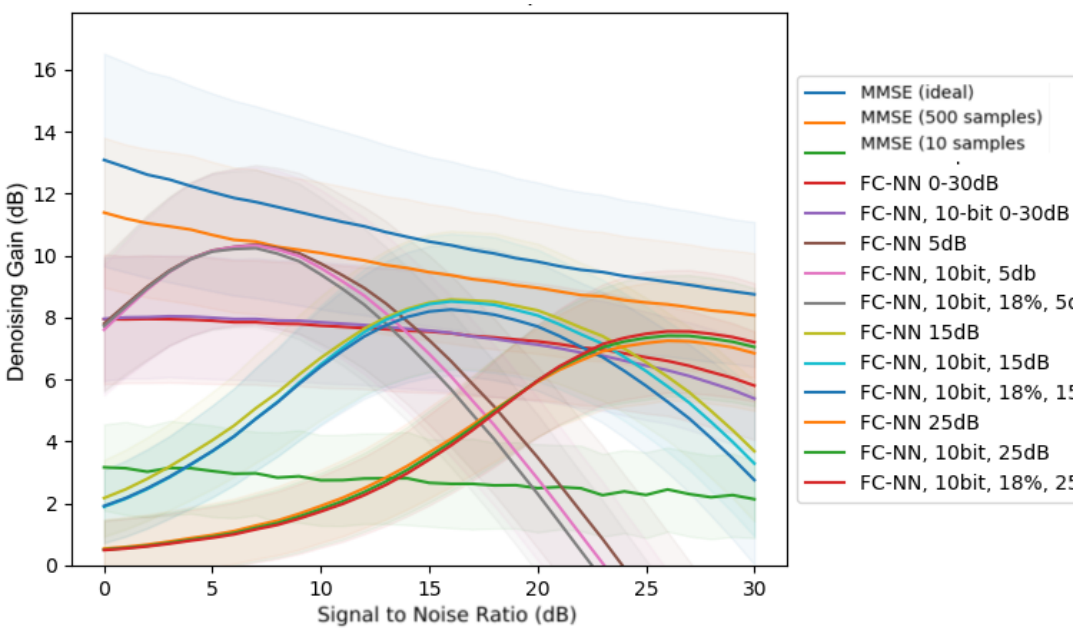

Figure 6.10: Performance of Pareto-optimal NNs after neuron pruning for different thresholds t resulting in different percentages of pruned neurons.

6.7 Optimizing by sharing Layers and Channels K Models

To see if we could improve the memory-efficiency of the *split SNR ranges* approach with K distinct NNs, we tried to fuse one or multiple of the first layers, as we hypothesize that the initial encoding/features of the denoising autoencoder might be similar. The approach is proposed in Section (5.6.1). As a proof of concept, we have started with K=3 distinct FC-NN models trained with Pareto-optimal select model configuration as denoted in Table (6.2).

Our results in Figure (6.11) indicate that sharing the first layer between two or more (Pareto-optimal) distinct NNs, to reduce the combined memory size for $K = 3$ distinct NNs is possible. Sharing the first layer between two distinct NNs trained, on low and middle SNR (5dB, 15dB) (while the high SNR model with K=3 (25dB) is not shared), results, in a modest accuracy degradation of 0.07dB, while being 9.6% smaller in total combined memory size. This results in a new Pareto-optimal model configuration, which lies between the started/selected Pareto-optimal configuration from Table (6.2) and one smaller Pareto-optimal configuration, who do not share a layer. These settings are shown in Table (6.3) below. This introduces another extra dimension in the design-space. Sharing the first layer between all the $K = 3$ distinct NNs results in another memory decrease, but also with a significant performance loss, such that the model doesn't become a Pareto-optimal model. Choosing a combination of two smaller Pareto-optimal models, and one default Pareto-optimal model with no shared layers, would result in a smaller and better performing model, as shown in Figure (6.11).

Therefore we hypothesize that the learned features of the first layers might be more different between (for different SNR that are further of each other) than initially anticipated. It could be therefore interesting to investigate what the impact is of sharing multiple layers for the *Split SNR* approach for $K > 3$. It is expected that models that are close to each other, in terms of SNR, e.g. SNR difference is $< 10dB$, and a comparable performance could be achieved as with our two-layer approach. Since this introduces another dimension in the design-space, with more complex training (Section (5.6.1), this approach is not further investigated due to time restrictions.

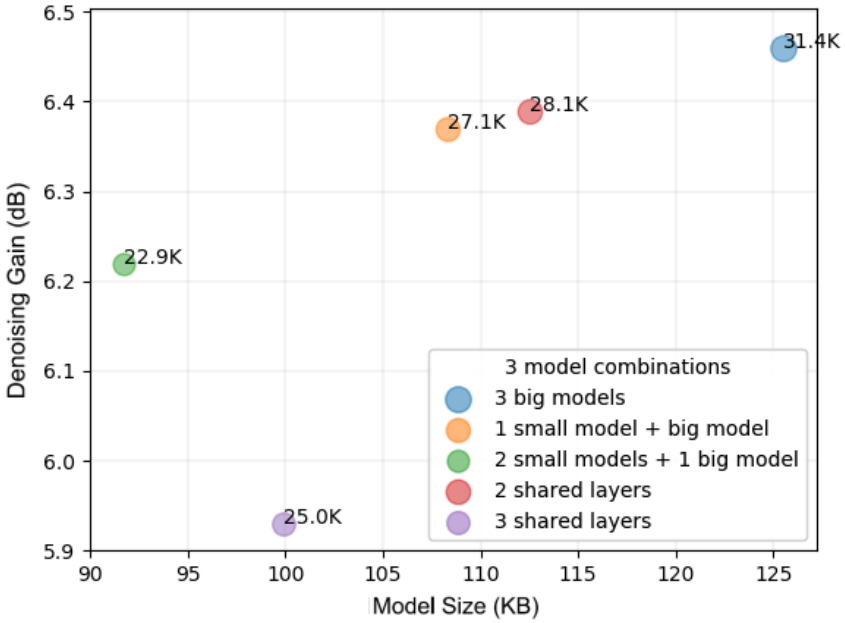

Figure 6.11: Average performance of Pareto-optimal FC-NNs where each point represents a combination of K distinct FC-NNs that share one or more common first layer(s). The performance is measured on the entire SNR range 0 - 30 dB.

Moreover, to see if we could reduce the model size on a more fine-grained basis, of the $K > 1$ models we tried to re-use some of the NN channel weights across different SNR ranges, as we

Model number	Trained SNR	Blue	Red	Yellow	Green	Purple
Model 1	5dB	(64, 3)	(64, 3)	(64, 2)	(64, 2)	(64, 3)
Model 2	15dB	(64, 3)	(64, 3)	(64, 3)	(64, 2)	(64, 3)
Model 3	25dB	(64, 3)	(64, 3)	(64, 3)	(64, 3)	(64, 3)
Shared layers		No	M1 - M2	No	No	M1 - M2 - M3

Table 6.3: Shared model configuration for Figure (6.11)

hypothesized that some of the weights across models might be similar. This did not improve the performance and, even made it worse.

6.8 DSE of Complex-Valued Neural Networks

To evaluate the performance and the complexity of our FC-CVNN architecture as introduced in Section (4.1.2), we compare it with our FC-NN, for both the *wide SNR range* and *split SNR range* approach as introduces in Section (5.6). The results for the FC-NN are take from Section (6.4) and (6.5). The FC-CVNN model configuration are defined in Section (6.3), while the training settings from Table (5.1) are used.

Figure (6.12) and Figure (6.13) provide the design-space exploration of the FC-CVNN architecture for $K = 3$ distinct NNs for both the *wide SNR range* and *split SNR range* approach. The same trend is observed as for the FC-NN: the model size versus performance trade-off curve is steep in both cases and levels off quickly after some model size. When we compare the design-space exploration of both architectures, it is noticed that for smaller models ($25KB <$) and for low SNRs (5dB and 15dB) the FC-CVNN performs better for the split SNR approach. The same trend is also noticed for the single SNR: the FC-CVNN performs better for model sizes up to 40KB. For all other model sizes (*split SNR* ($\geq 25KB$) and *wide SNR* ($\geq 40KB$)), the models perform always worse compared their real-valued counterpart, although differences are minimal.

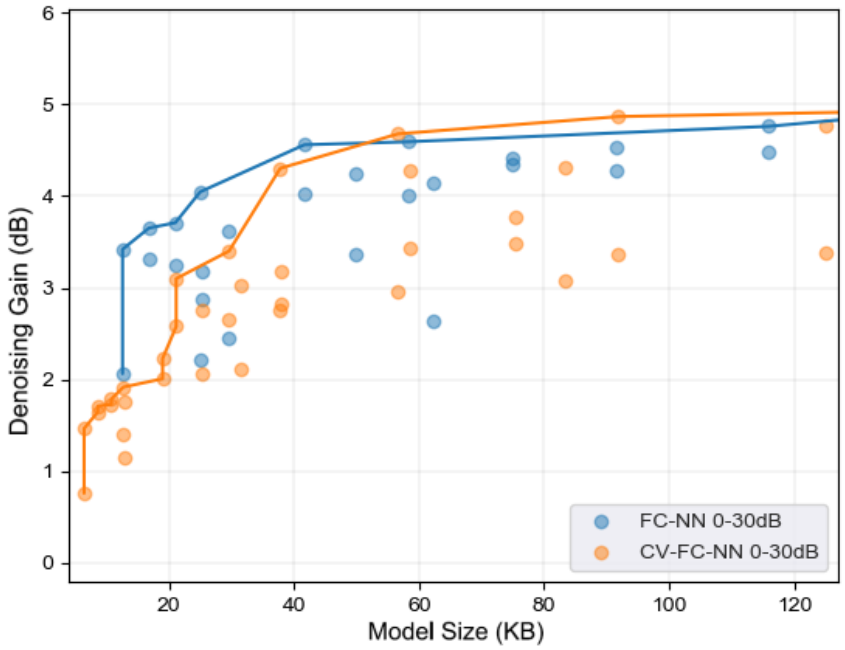

Figure 6.12: Design-space exploration where each point represents a single FC-NN or FC-CVNN that is used for the entire SNR range from 0 dB to 30 dB. The line represents the Pareto frontier for each NN.

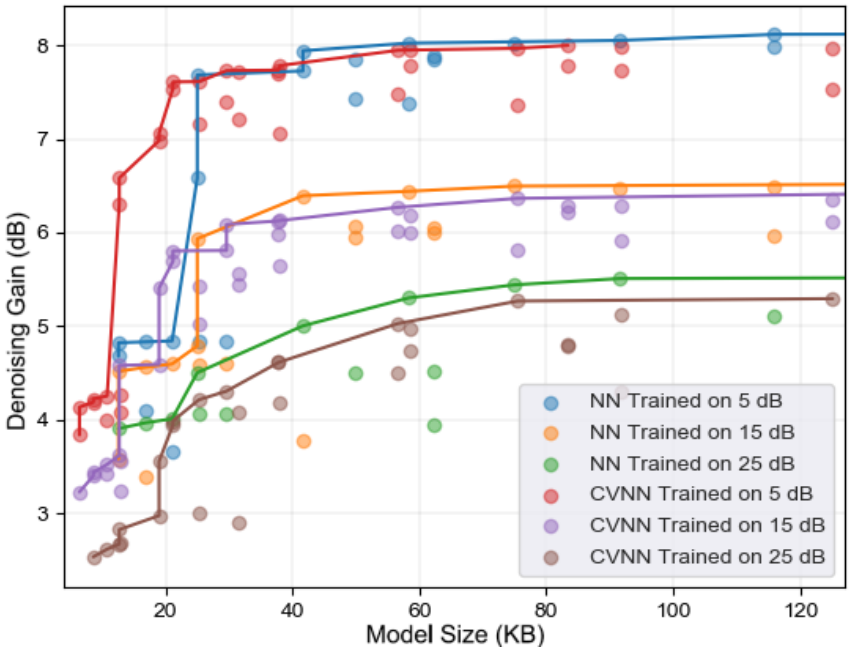

Figure 6.13: Design-space exploration where each point represents a on of the three distinct split SNR FC-NN or FC-CVNN that is used for the entire SNR range from 0 dB to 30 dB. The line represents the Pareto frontier for each NN.

Optimizing across wide SNR

In Figure (6.17) we provide the performances for the SNR range of interest for the FC-CVNN single SNR and split SNR approach for the Pareto-optimal real-valued counterparts. A comparable in Pareto-optimal model configuration (in memory size and computation) for single SNR is: $Act = \mathbb{C}tanh$, $D = 5$, $W = 64$, and for $K = 3$ distinct split SNR: $Act = \mathbb{C}\text{tanh}$, $D = 3$, $W = 48$. As expected from the previous design-space exploration results, they have a similar generalization power compared to the FC-NN. While on overall have a minor performance loss compared to the FC-NN Pareto-optimal counterparts (with the same memory size and computational complexity).

DSE of Fixed-point FC-CVNN Quantization

In Figure (6.14) and Figure (6.15) we present the results of our quantization design-space exploration executed on the Pareto-optimal FC-CVNN. When we compare the FC-CVNNs with the real-valued counterparts, the same trend is denoted as for the non-quantized models. For small models, the FC-CVNN performs better, but bigger models suffer from a minor performance loss. The best performing models are the FC-NNs. Furthermore, in Figure (6.17) we provide the performance for the SNR range of interest for the FC-CVNN 10-bit quantized single SNR and split SNR approach, which shows that that effect applies on the wide range compared to the quantized FC-NN, but the performance is lower.

In Figure (6.6) we illustrate the denoising gain as a function of the quantization bit-width for the cases where a single FC-CVNN and $K = 3$ distinct FC-CVNNs are used for the SNR range of interest. For this evaluation, we use again selected Pareto-optimal FC-CVNN configurations for each case, chosen in Section (6.4). We observe that all considered FC-CVNNs are quite robust to quantization down to a bit-width of $Q = 10$ bits, while a significant performance degradation starts appearing for $Q = 8$ bits. This similar is noticed for the FC-NN, and we can conclude that FC-CVNNs in our application areas robust to quantization as FC-NN but the maximal feasible in our application is lower.

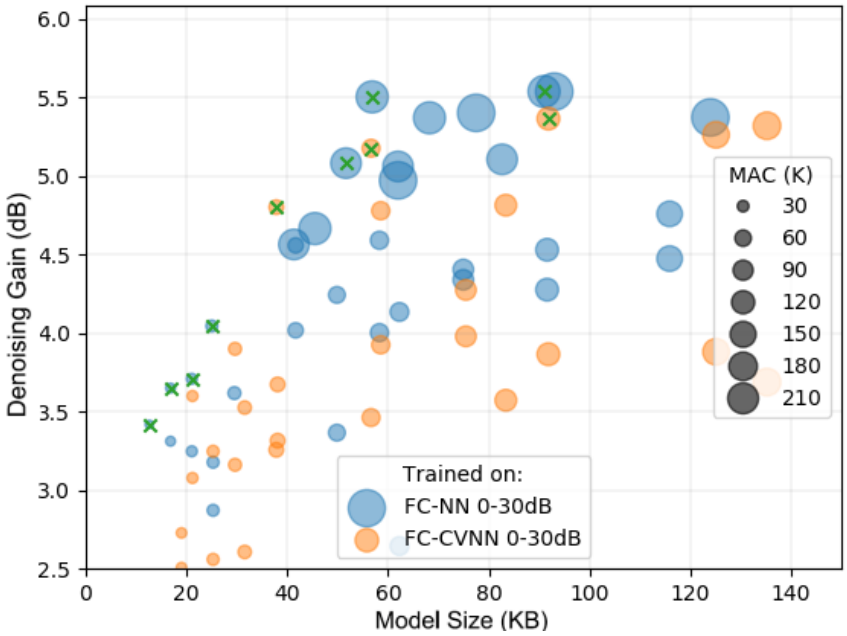

Figure 6.14: Design-space exploration where each bubble represents a single FC-CVNN (*wide SNR trained*) or FC-NN that is used for the entire SNR range 0 - 30 dB. The crosses represents the Pareto frontier for each NN.

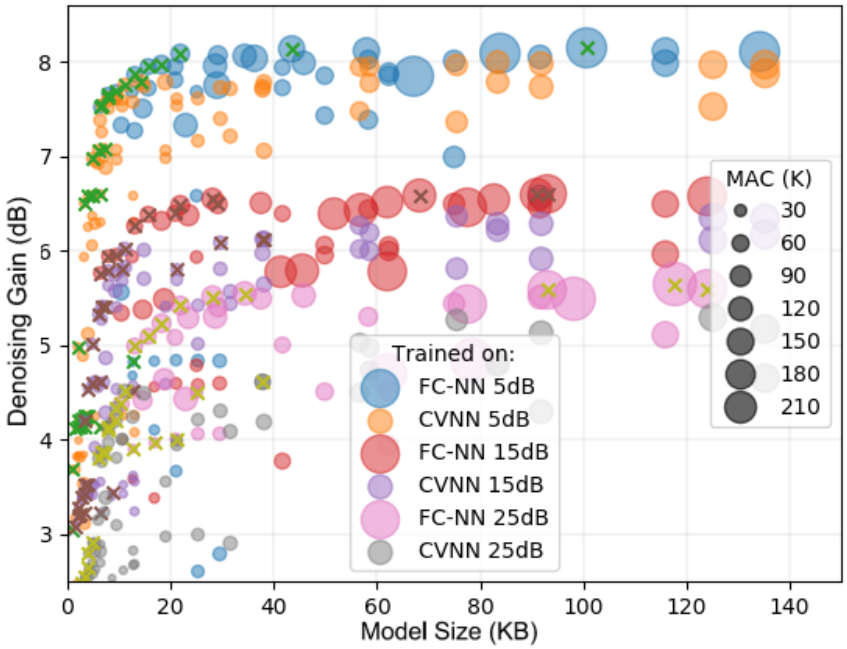

Figure 6.15: Design-space exploration where each bubble represents a one of the three distinct FC-CVNN (*split SNR trained*) or FC-NN that is used for the entire SNR range 0 - 30 dB. The crosses represents the Pareto frontier for each NN.

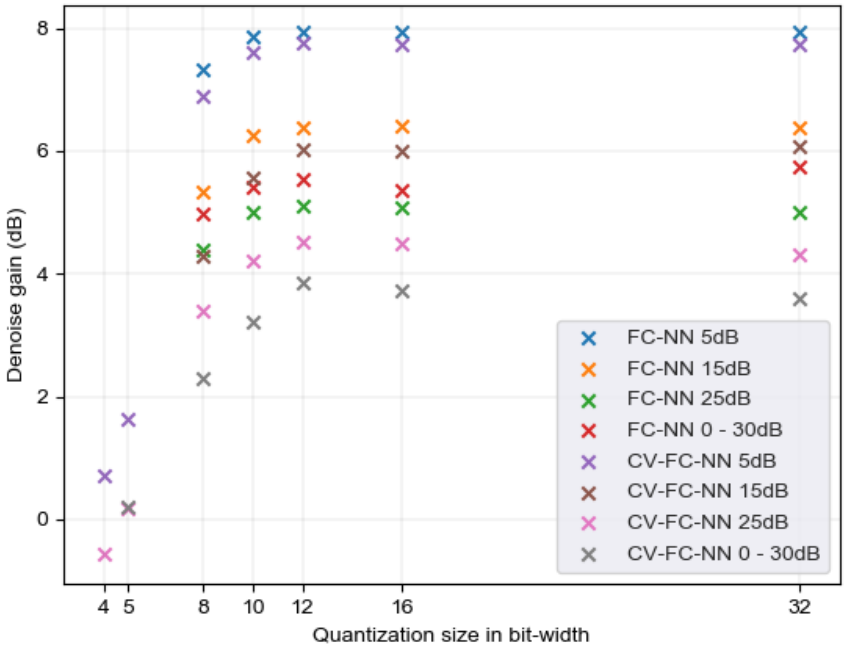

Figure 6.16: Effect of quantization with different bit-widths on smaller FC-CVNN models trained on single SNR and a big FC-CVNN model trained on a wide SNR range of 0 - 30dB.

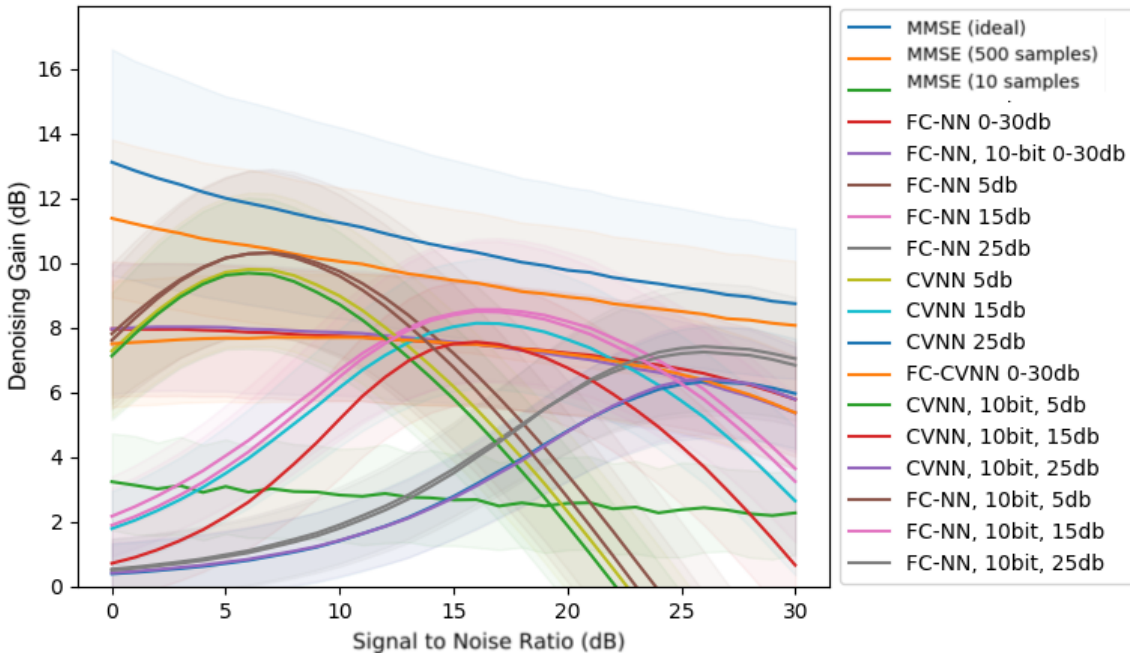

Figure 6.17: Denoise performance of picked FC-CVNN Pareto-optimal (10bit quantized) single SNR model and (quantized) $k = 3$ distinct models and ideal MMSE, MMSE with 10 samples for ACF and 500 ACF samples.

While it is found that careful selection of the FC-CVNN design-space exploration of models configurations can help to find Pareto-optimized models for each performance target and that these models and have comparable robustness to quantization as the FC-NN. The maximal achievable performance is slightly lower than the maximal feasible performance with comparable FC-NN models. Therefore, the FC-CVNN could be an alternative to the FC-NN on complex hardware (e.g. when the FC-NN has too much overhead or/and for memory efficiency reasons as discussed in Section (2.4)), but at a cost of minor performance loss. Since this architecture performs on average lower than the FC-NN and for time reason, neuron pruning is not explored for this NN architecture.

Paramter	**Value**
Wide SNR range **(K=1)**	
W	64
D	5
Act	$\mathbb{C}$tanh
Wide SNR range **($K > 1$)**	
W	48
D	3
Act	$\mathbb{C}$tanh

Table 6.4: Selected Pareto-optimal model configurations which are used through the report for evaluation and comparison of the FC-CVNN model.

6.9 DSE of Convoluional Neural Networks

As discussed previously, due to the massive design-space of the CNN architecture and limited time, only one CNN configuration is explored. Therefore no design-space exploration of the configuration is explored, and only one CNN model configurations is used for the quantization and pruning step. The settings for the CNN are defined in Section (6.3), while the training settings from Table (5.1) are used.

Figure (6.18) and (6.19) presents the results of the CNN for the *wide SNR range* and *split SNR ranges* approach in comparison with the FC-NN architecture. When the full-precision (32bit) CNN is compared with the full-precision (32bit) FC-NN architecture, there are three things noticed. The $K = 3$ distinct CNN (*split SNR*) together perform up to 1dB better compared to the single model (*wide SNR*), for the same computation complexity but for three times higher memory size. Furthermore, the maximal achievable denoise performance for the CNN is up to 1dB lower than the *wide SNR* FC-NN, while for the *split SNR* FC-NN the difference is smaller with 0.5dB.

Evaluation with Quantization

When we look at the quantized models in Figure (6.18) and (6.19), it is noticed that both the *wide SNR* and *split SNR* are robust to quanization to least 16-bit, which is comparable to the FC-NN and FC-CVNN. Similar to FC-NN and FC-CVNN, we show that quantization is an effective way to reduce the memory size of CNN in our application. But compared to FC-NN are still a better a better choice when it comes to memory size and computational complexity. For the single CNN SNR model, the non quantized

Evaluation Across Wide SNR

The above discussed CNN model architecture does not introduce any improvements, and has noticeable performance loss when compared to the FC-NN, while having significant higher computation complexity and/or memory size. It is expected that applying our pruning on the current quantized CNN configuration will not lead to an performance with competing computational complexity and model size for our application with FC-NN. Therefore for time reasons, neuron pruning is not explored. We expect that design-space exploration of the CNN model configuration can have a bigger impact on the performance, computational complexity and model size. Therefore if time and computation power for training is available, designs space of the model configuration would be interesting future work

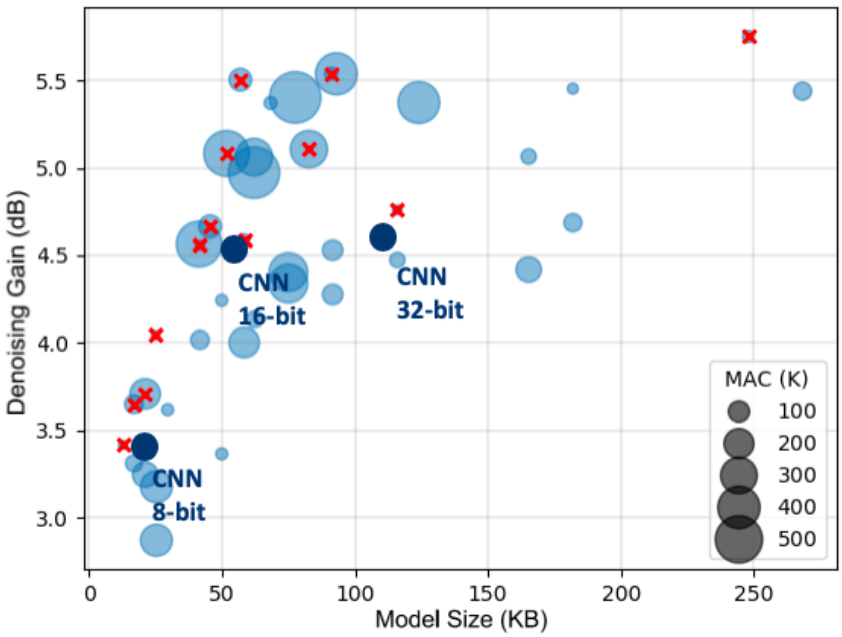

Figure 6.18: (Left) Design space exploration of different model sizes. Each dot represents a model configuration, lines denotes the Pareto optimal model size in terms of de-noise gain and model memory size.

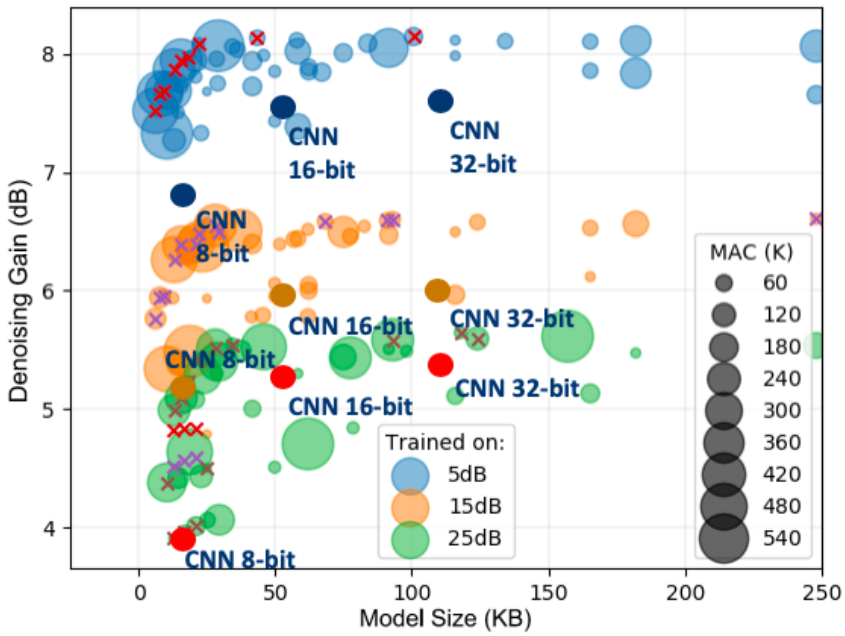

Figure 6.19: (Left) Design space exploration of different model sizes. Each dot represents a model configuration, lines denotes the Pareto optimal model size in terms of de-noise gain and model memory size.

6.10 Binary Neural Network

The design-space exploration of the BNN size The settings for the FC-NN model configuration are defined in Section (6.3), while the training settings from Table (5.1) are used. Due to the design space exploration, we were not able to achieve higher accuracy than the smallest Pareto-optimal FC-NN model configuration. It is expected that the used NN is too small, and the BNN loses too much information. Therefore we don't report the performance.

6.11 Overview

In Table (6.5 a comparison is provided of the average denoising performance, MACs, and model size over the SNR range from 0 dB to 30 dB for various numbers K of distinct, for only the FC-NNs architecture. For reference, the average denoising gain for the ideal MMSE estimator with perfect knowledge of the channel covariance matrix in this SNR range is 9.1 dB. Furthermore, for the evaluation of the realistic MMSE estimator, 100 samples are used to estimate the channel covariance. For the computational complexity is assumed that the MMSE is always updated for every new pilot symbols.

		Models			
		1	2	3	4
Q = 32	Perf.	5.75 dB	6.01 dB	6.60 dB	**6.83 dB**
	MACs	62000	10416	10416	**10416**
	Model size	248 KB	83 KB	125 KB	**166.7 KB**
MMSE	Perf.	***5.22 dB***			
	MACs	***18094***			
	Memory size	***9.216 KB***			
Q = 10	Perf.	5.40 dB	5.80 dB	6.58 dB	**6.82 dB**
	MACs	62000	10416	10416	**10416**
	Model size	77.50 KB	26 KB	39 KB	**52.10 KB**
Q = 10 & pruning	Perf.	5.12 dB	5.79 dB	**6.47 dB**	6.74 dB
	MACs	33159	8547	**8547**	8547
	Model size	41.44 KB	21.40 KB	**32.00 KB**	42.70 KB

Table 6.5: Performance, computational complexity and memory size over view for the FC-NN architecture.

Chapter 7

Conclusions

High quality of service in wireless communication systems is important to serve high demanding applications and user experience. While DL in communication has gained a tremendous amount of interest in recent years as an alternative to the classical algorithms to improve the quality of service, they suffer from high memory and computation requirements. This limits practical implementations of DL algorithms on memory, computation, and power-limited embedded devices. This work presents a framework for systematic design-space optimization of channel denoising NNs for OFDM wireless communication systems. This framework was used for design-space exploration of diverse channel denoise NN architectures, and configuration in wide SNR scenarios. On top of this, this framework is used for design-space exploration of quantization, pruning and layer/channel sharing to further reduce the memory and computation complexity of these NNs.

We showed that carefully exploring and selecting the NN configuration is necessary to design solutions with Pareto-optimal performance-complexity trade-offs. By generating such a set of NN configurations, an NN configuration can be selected for a specific memory, computation and/or performance target.

Furthermore, we showed that the FC-NN is the best performing NN architecture among the other evaluated architectures in terms of memory size, computational complexity, and denoise performance, during all the steps in the design-space exploration for our channel denoising scenario. To be more specific, the FC-CVNN is the only architecture that is performing close (up to 0.3dB less for a comparable memory and computation target) to the FC-NN, and could, therefore, be an alternative to the FC-NN when hardware with complex arithmetic is used. While other NN architectures such as CNNs introduce a substantial performance loss up to 1dB while requiring 23% more memory and 50% more computation. For BNN architectures the performance penalty is too big, such that these are not suitable for our application.

Moreover, we confirmed that fixed-point quantization and neuron pruning are effective complexity-reduction methods for this application. We found that, when quantization is applied on-top a Pareto-optimal NN configuration, practically feasible bit-width of $Q = 10$ bits is sufficient, without a noticeable performance decrease, for the examined channel denoising scenario. On overall quantization is robust up to bit widths of $Q = 10$, for all the architectures. This makes it possible to deploy these NN on fixed-point (embedded) hardware e.g. DSPs, GPUs. After selecting and quantizing a model often almost 50% of the neurons can be pruned, which can reduce 46% of the memory and 48% of the computation with negligible performance degradation, in comparison to the 32-bit floating-point Pareto optimal counterpart. In some cases, this can lead to a memory size reduction of up to 83.2%, while the computational complexity can be reduced up to 86.2%. This Pareto-optimal quantized and pruned model outperforms the classical MMSE solution in fast fading channel situations where less than 200 samples are available for estimation of the channel. On top of this we show that due to these quantization and pruning steps, the computation becomes close to the classical MMSE solution with a difference of only 2000 MACs. Furthermore, our approach massively decreases the computational complexity and memory size massively in comparison with related works, especially in wide SNR scenarios.

Supporting a wide range of SNR is an important criterion in wireless communication systems. We showed that during all steps of our design-space exploration using multiple distinct NNs trained for different SNR ranges, (where each model is trained on only on SNR of a very small range) is more effective than using a single NN trained over the entire SNR range of interest. This not only increases the performance but also can reduce significantly the memory and computation requirements. For example, when using both quantization and pruning, $K = 3$ distinct NNs have a 1.35 dB better average denoising gain while also being 74% less computationally complex and 23% smaller in terms of model memory size. Finally, we show that sharing NN-layers or NN-channels between two or more of these models multiple distinct NNs is possible but leads to another dimension of the design-space without significant improvements.

The work in this thesis does not only solves and contributes to existing related work but also brings new questions and possibilities for future work, which usually were not explored due to time restrictions. One of the most interesting things could be exploring for the more advanced state of the art NN architectures, including a design-space exploration of the CNN configuration size. Also, other techniques for channel estimation and/or equalization could improve the quality of service of the OFDM Rx, while reducing the computational complexity and memory size. One of them which was already started with exploring is an estimation of the correlation matrix in the MMSE solution. While we have shown that our approaches significantly reduce memory and computation complexity it is unknown what the exact execution speed impact on embedded hardware. Also, real-world test in a practical set-up could lead to interesting insights and the effectiveness of DL-based RX in-comparison with classical solutions.

Bibliography

[1] A. OUELLETTE. (2019) These are the 17 top tech buzzwords you need to know. [Online]. Available: https://careerfoundry.com/en/blog/web-development/tech-buzzwords-to-learn/#13-smart-industry-40 1

[2] S. Durcevic. (2020) Top 10 it & technology buzzwords you wont be able to avoid in 2020. [Online]. Available: https://www.datapine.com/blog/technology-buzzwords/ 1

[3] S. Gupta. (2018) Deep learning performance breakthroug. [Online]. Available: https://newsroom.ibm.com/IBM-systems?item=30076 1

[4] G. Intelligence. (2019) Definitive data and analysis for the mobile industry, subscriber penetration. [Online]. Available: https://www.gsmaintelligence.com 1

[5] C. Hoymann, D. Larsson, H. Koorapaty, and J. Cheng, "A lean carrier for lte," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 74–80, February 2013. 2

[6] S. Parkvall, E. Dahlman, A. Furuskar, and M. Frenne, "Nr: The new 5g radio access technology," *IEEE Communications Standards Magazine*, vol. 1, no. 4, pp. 24–30, Dec 2017. 2

[7] ETSI. (2018) Why do we need 5g? [Online]. Available: https://www.etsi.org/technologies/5g?jjj=1577891413176 2

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012. 2

[9] H. Ye, G. Y. Li, and B. Juang, "Power of deep learning for channel estimation and signal detection in ofdm systems," *IEEE Wireless Communications Letters*, vol. 7, no. 1, pp. 114–117, Feb 2018. 2, 4, 5, 6, 7, 13, 18, 19, 35

[10] P. Jiang, T. Wang, B. Han, X. Gao, J. Zhang, C.-K. Wen, S. Jin, and G. Li, "Artificial intelligence-aided ofdm receiver: Design and experimental results," ", 12 2018. 2, 4, 5, 6, 7, 13, 18, 19, 35

[11] X. Gao, S. Jin, C.-K. Wen, and G. Li, "Comnet: Combination of deep learning and expert knowledge in ofdm receivers," *IEEE Communications Letters*, vol. PP, pp. 1–1, 10 2018. 2, 3, 4, 5, 6, 7, 13, 18, 19, 35

[12] M. Soltani, A. Mirzaei, V. Pourahmadi, and H. Sheikhzadeh, "Deep learning-based channel estimation," *CoRR*, vol. abs/1810.05893, 2018. [Online]. Available: http://arxiv.org/abs/1810.05893 2, 5, 7, 14, 15, 18, 19, 31, 35

[13] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," *CoRR*, vol. abs/1802.05668, 2018. [Online]. Available: http://arxiv.org/abs/1802.05668 3, 8

[14] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *CoRR*, vol. abs/1702.03044, 2017. [Online]. Available: http://arxiv.org/abs/1702.03044 3

[15] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," *CoRR*, vol. abs/1807.10029, 2018. [Online]. Available: http://arxiv.org/abs/1807.10029 3, 8

[16] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," 2016. 3, 8, 31

[17] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks," *CoRR*, vol. abs/1810.11809, 2018. [Online]. Available: http://arxiv.org/abs/1810.11809 3, 8

[18] G. Palm, "Warren mcculloch and walter pitts: A logical calculus of the ideas immanent in nervous activity," in *Brain Theory*, G. Palm and A. Aertsen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 229–230. 5

[19] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, p. 563575, Dec. 2017. 5

[20] D. Gunduz, P. de Kerret, N. D. Sidiropoulos, D. Gesbert, C. Murthy, and M. van der Schaar, "Machine learning in the air," Apr. 2019. [Online]. Available: https://arxiv.org/abs/1904.12385 5

[21] A. Balatsoukas-Stimming and C. Studer, "Deep unfolding for communications: A survey and some new directions," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2019. 5

[22] T. Wang, C. Wen, H. Wang, F. Gao, T. Jiang, and S. Jin, "Deep learning for wireless physical layer: Opportunities and challenges," *China Communications*, vol. 14, no. 11, pp. 92–111, Nov. 2017. 5

[23] D. Liu, P. Smaragdis, and M. Kim, "Experiments on deep learning for speech denoising," *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pp. 2685–2689, 01 2014. 7

[24] S.-W. Fu, Y. Tsao, and X. lu, "Snr-aware convolutional neural network modeling for speech enhancement," in *"*, 09 2016, pp. 3768–3772. 7, 31

[25] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *"*, 03 2015. 8

[26] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-scale dense convolutional networks for efficient prediction," *CoRR*, vol. abs/1703.09844, 2017. [Online]. Available: http://arxiv.org/abs/1703.09844 8

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842 8

[28] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360 8

[29] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861 8

[30] N. Ma, X. Zhang, H. Zheng, and J. Sun, "Shufflenet V2: practical guidelines for efficient CNN architecture design," *CoRR*, vol. abs/1807.11164, 2018. [Online]. Available: http://arxiv.org/abs/1807.11164 8

[31] X. Yu, T. Liu, X. Wang, and D. Tao, "On compressing deep models by low rank and sparse decomposition," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 67–76. 8

[32] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. S. Feris, "Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification," *CoRR*, vol. abs/1611.05377, 2016. [Online]. Available: http://arxiv.org/abs/1611.05377 8

[33] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. 8

[34] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," *CoRR*, vol. abs/1504.04788, 2015. [Online]. Available: http://arxiv.org/abs/1504.04788 8

[35] H. Hu, R. Peng, Y. Tai, and C. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *CoRR*, vol. abs/1607.03250, 2016. [Online]. Available: http://arxiv.org/abs/1607.03250 8

[36] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015. [Online]. Available: http://arxiv.org/abs/1510.00149 8

[37] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," *CoRR*, vol. abs/1711.11294, 2017. [Online]. Available: http://arxiv.org/abs/1711.11294 8

[38] A. Hirose, *Complex-Valued Neural Networks: Advances and Applications*. ", 05 2013. 9

[39] D. P. Reichert and T. Serre, "Neuronal synchrony in complex-valued deep networks," 2013. 9

[40] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2013. 9

[41] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," *CoRR*, vol. abs/1511.06464, 2015. [Online]. Available: http://arxiv.org/abs/1511.06464 9

[42] I. Danihelka, G. Wayne, B. Uria, N. Kalchbrenner, and A. Graves, "Associative long short-term memory," *CoRR*, vol. abs/1602.03032, 2016. [Online]. Available: http://arxiv.org/abs/1602.03032 9

[43] S. Wisdom, T. Powers, J. R. Hershey, J. L. Roux, and L. Atlas, "Full-capacity unitary recurrent neural networks," 2016. 9

[44] A. Hirose and S. Yoshida, "Generalization characteristics of complex-valued feedforward neural networks in relation to signal coherence," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 4, pp. 541–551, April 2012. 9

[45] A. Hirose, "Nature of complex number and complex-valued neural networks," *Frontiers of Electrical and Electronic Engineering in China*, vol. 6, no. 1, pp. 171–180, Mar 2011. [Online]. Available: https://doi.org/10.1007/s11460-011-0125-3 9

[46] C. Trabelsi, O. Bilaniuk, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal, "Deep complex networks," *CoRR*, vol. abs/1705.09792, 2017. [Online]. Available: http://arxiv.org/abs/1705.09792 9

[47] J. Thickstun, Z. Harchaoui, and S. Kakade, "Learning features of music from scratch," ", 11 2016. 9

[48] O. Moran, P. Caramazza, D. Faccio, and R. Murray-Smith, "Deep, complex, invertible networks for inversion of transmission effects in multimode optical fibres," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 3280–3291. 9

[49] J. Gao, B. Deng, Y.-l. Qin, H. Wang, and X. Li, "Complex-valued convolutional neural network enhanced radar imaging," *IEEE Geoscience and Remote Sensing Letters*, 12 2017. 9

[50] B. Stantchev and G. Fettweis, "Time-variant distortions in ofdm," *IEEE Communications Letters*, vol. 4, no. 10, pp. 312–314, Oct 2000. 10

[51] P. Robertson and S. Kaiser, "The effects of doppler spreads in ofdm(a) mobile radio systems," in *Gateway to 21st Century Communications Village. VTC 1999-Fall. IEEE VTS 50th Vehicular Technology Conference (Cat. No.99CH36324)*, vol. 1, Sep. 1999, pp. 329–333 vol.1. 10

[52] P. Hoeher, S. Kaiser, and P. Robertson, "Two-dimensional pilot-symbol-aided channel estimation by wiener filtering," in *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, April 1997, pp. 1845–1848 vol.3. 12, 19

[53] Y. Romano, M. Elad, and P. Milanfar, "The little engine that could: Regularization by denoising (RED)," Sep. 2017. [Online]. Available: https://arxiv.org/abs/1611.02862 12, 13

[54] J. Xie, L. Xu, and E. Chen, "Image denoising and inpainting with deep neural networks," in *International Conference on Neural Information Processing Systems (NIPS)*, 2012, pp. 341–349. 12

[55] D. Liu, P. Smaragdis, and M. Kim, "Experiments on deep learning for speech denoising," *Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pp. 2685–2689, Jan. 2014. 12

[56] Y. B. Pascal Vincent, Hugo Larochelle and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *International Conference on Machine Learning*, 2008. 12

[57] B. S. Oliver Drr and E. Murina. (2019) Neural network architectures. [Online]. Available: https://freecontent.manning.com/neural-network-architectures/ 13

[58] J. B. Ahire. (2018) The artificial neural networks handbook: Part 4. [Online]. Available: https://medium.com/@jayeshbahire/the-artificial-neural-networks-handbook-part-4-d2087d1f583e 14

[59] S. Saha. (2018) A comprehensive guide to convolutional neural networks the eli5 way. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 15

[60] B. de Bruin, "Fast and efficient mapping of convolutional neural networks on an imaging dsp," Oct. 2017. [Online]. Available: https://pure.tue.nl/ws/portalfiles/portal/89092569/0919605_BarrydeBruin_thesis_report_v4.pdf 15

[61] M. van Lier, L. Waeijen, and H. Corporaal, "Bitwise neural network acceleration: Opportunities and challenges," in *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, June 2019, pp. 1–5. 17

[62] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167 17

[63] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830 17

[64] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016. 19

[65] 3GPP, "Study on channel model for frequency spectrum above 6 GHz," *TR 38.900*, 2016. 24

[66] G. Hinton. (2012) Neural networks for machine learning, lecture 6a, overview of mini-batch gradient descent. [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf 24

[67] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *CoRR*, vol. abs/1806.08342, 2018. [Online]. Available: http://arxiv.org/abs/1806.08342 28

[68] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 3123–3131. 28

[69] A. Murthy, H. Das, and M. A. Islam, "Robustness of neural networks to parameter quantization," *CoRR*, vol. abs/1903.10672, 2019. [Online]. Available: http://arxiv.org/abs/1903.10672 28

[70] P. Gysel, "Ristretto: Hardware-oriented approximation of convolutional neural networks," *CoRR*, vol. abs/1605.06402, 2016. [Online]. Available: http://arxiv.org/abs/1605.06402 28

[71] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," *CoRR*, vol. abs/1712.05877, 2017. [Online]. Available: http://arxiv.org/abs/1712.05877 28

Appendices

Appendix A

FC-NN Model Configuration Design-Space Exploration

A.1 Optimized Pareto-optimal FC-NN K Split SNR Model Configurations

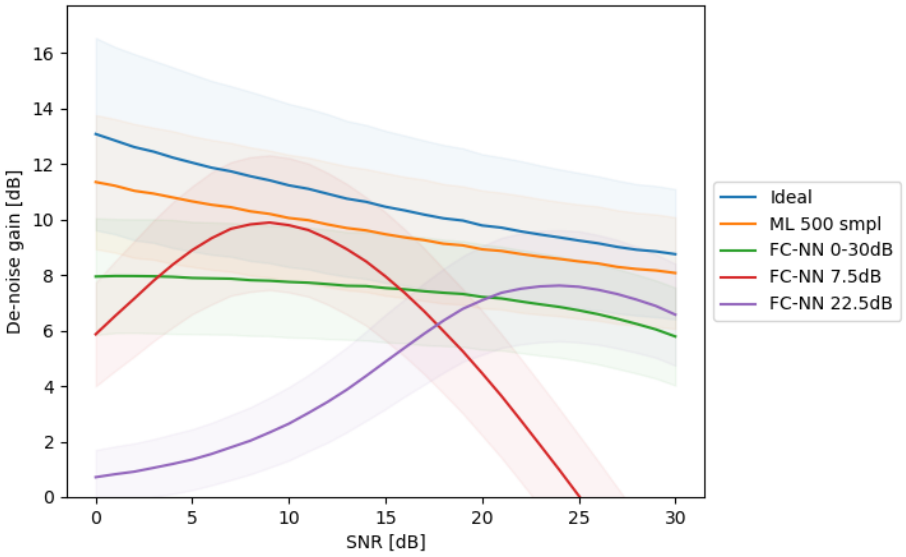

Figure A.1: Channel denoise performance for the entire SNR range 0 - 30 dB for $K = 2$ distinct Pareto-optimal FC-NNs *split SNR trained* compared with single SNR Pareto-optimal FC-NN, ideal MMSE and MMSE with 500 ACF samples.

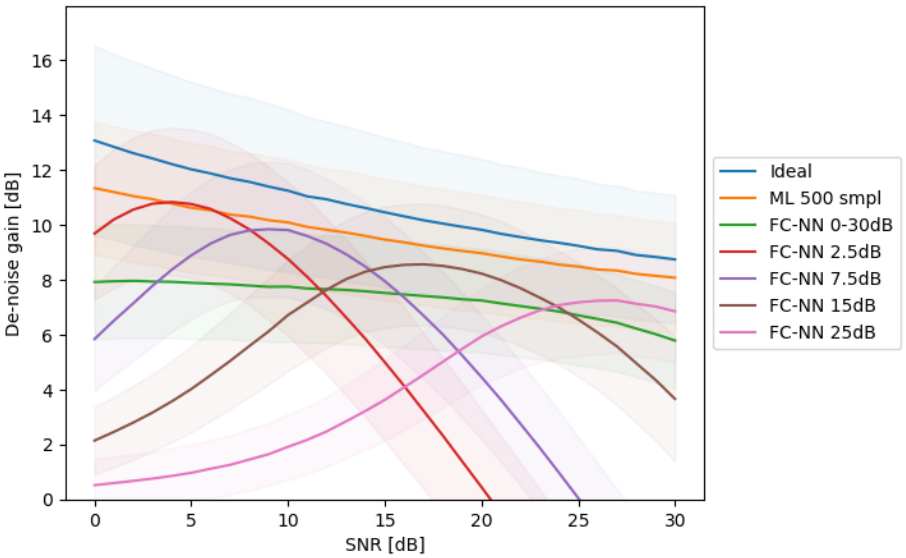

Figure A.2: Channel denoise performance for the entire SNR range 0 - 30 dB for $K = 4$ distinct Pareto-optimal FC-NNs *split SNR trained*compared with single SNR Pareto-optimal FC-NN, ideal MMSE and MMSE with 500 ACF samples.