

Henri Kasurinen

# QUANTIZATION-AWARE PRUNING FOR A CNN-BASED RADIO RECEIVER MODEL

Master's Thesis

Faculty of Information Technology and Communication Sciences

Examiners: Prof. Pekka Jääskeläinen, Dr. Mikko Honkala Nokia Bell-Labs

June 2021

## ABSTRACT

Henri Kasurinen: Quantization-aware pruning for a CNN-based radio receiver model  
Master's Thesis  
Tampere University  
Master of Science  
June 2021

---

Machine Learning (ML) has become a vital part of our world as Convolutional Neural Networks (CNN) enabled super-human performance in a multitude of tasks. The downside is that the computational complexity and memory foot print of CNNs has increased along with their performance to billions of operations, making it impossible to deploy CNNs on resource-restricted hardware. Solutions for this problem can be found in CNN compression methods like CNN pruning and quantization. CNN quantization replaces parts of the original network with approximations, which trade-off accuracy for smaller computation size. CNN pruning on the other hand, removes parts of the network to reduce the computational complexity.

This thesis studies the compression of a CNN-based radio receiver called Deep Learning Receiver (DeepRx), which faces the fore-mentioned issues of a high computational complexity and a resource-restricted working environment. We reviewed the literature on CNN compression and decided to focus on the combination of pruning and quantization. Based on this research we developed a quantization-aware pruning algorithm called Quantization-Aware Multi-Stage Pruning (QAMP). Our method achieved a 97,25% pruning ratio with the DeepRx receiver with only a small drop in accuracy. It has to be said, that the DeepRx model was initially over-parametrized, which leads to the huge percent-wise drop in size. Our results, however, are still impressive and encourage us for further development.

Keywords: Machine Learning, Convolutional Neural Network, CNN Pruning, CNN quantization, CNN optimization

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

# TIIVISTELMÄ

Henri Kasurinen: Konvoluutioneuroverkkojen kvantisointitietoinen harventaminen  
Diplomityö  
Tampereen yliopisto  
Diplomi-insinöörin tutkinto  
Kesäkuu 2021

---

Koneoppimisesta on tullut elintärkeä osa maailmaamme, kun konvoluutioneuroverkot mahdollistivat yli-inhimillisen suorituskyvyn monissa tehtävissä. Huonona puolena on se, että konvoluutioneuroverkkojen laskennallinen monimutkaisuus on kasvanut niiden suorituskyvyn myötä miljardeihin operaatioihin, mikä tekee niiden käyttöönotosta mahdotonta resurssirajoitteisilla laitteistoilla. Ratkaisu tähän ongelmaan löytyy konvoluutioneuroverkkojen optimointimenetelmistä, kuten "harventamisesta" ja kvantisoinnista. Kvantisoinnissa alkuperäisen verkon osat korvataan approksiimaatioilla, jolloin tarkkuus vaihdetaan pienempään. Konvoluutioneuroverkkojen harventamisessa puolestaan poistetaan osia verkosta laskennallisen monimutkaisuuden vähentämiseksi.

Tässä diplomityössä tutkitaan konvoluutioneuroverkko-pohjaisen radiovastaanottimen, Deep Learning Receiverin, eli DeepRx:än, optimisointia, jonka käytännön toteutuksen esteenä ovat edellä mainitut ongelmat: suuri laskennallinen monimutkaisuus ja rajatut laskennalliset resurssit. Tarkastelimme konvoluutioverkkojen optimointia koskevaa kirjallisuutta ja päätimme keskittyä harventamisen ja kvantisoinnin yhdistelmään. Tämän tutkimuksen perusteella kehitimme tätä tarkoitusta varten kvantitointitietoisen harvennus-algoritmin nimeltä Quantization-Aware Multi-Stage Pruning (QAMP). Onnistuimme pienentämään DeepRx-radiovastaanottimen neuroverkon kokoa 97,25 % ja vielä kvantisoimaan se siten, että se käyttää ainoastaan 8-bittisiä arvoja. On mainittava, että neuroverkko oli alkujaan "yliparametrioitu", joka osaltaan vaikutti neuroverkon koon suureen prosentuaaliseen muutokseen. Tuloksemme ovat silti vaikuttavia ja kehitystyötä jatketaan niiden perusteella.

Avainsanat: Koneoppiminen, konvoluutioneuroverkot, Optimointi

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

## PREFACE

Tämä diplomityö ei olisi koskaan valmistunut eikä sitä olisi edes aloitettu kirjoittamaan, ellei minua olisi pienestä pitäen kannustettu opiskelemaan ja kehittämään itseäni. Työn valmistumista on myös vauhdittanut se, että kotona vieraillessa on aina muistettu kysyä "Mites se dippa?". Haluaisin siis kiittää äitiäni Helena Kortelaista ja isääni Urpo Kasurista kaikesta tuesta ja ohjauksesta, jota olen elämäni aikana saanut. Suuri kiitos kuuluu myös veljelleni ja ikiaikaiselle kilpakumppanille Markus Kasuriselle, joka ehti saada oman diplomityönsä ennen minua valmiiksi. Tässä kilvassa hopea ei kuitenkaan ole häpeä. Kiitos myös isosiskolleni Nina Kasuriselle, joka on omalta osaltaan viitoittanut tietä nuoremmille sisaruksilleen korkeakoulutusta kohti. Kiitos myös tyttöystävälleni Laura Karintaukselle, joka on muistuttanut minua siitä että välillä on hyvä tehdä jotain muutakin kuin diplomityötä. Samasta syystä haluaisin kiittää koko BFI:tä, joka on dippa-vertaistuen ohella tarjonnut myös (toivottavasti) elinikäisen ystävyyden.

Haluaisin kiittää suuresti diplomityön valvojia Mikko Honkalaa Nokia Bell Labsilta, sekä Pekka Jääskeläistä Tampereen yliopistolta sujuvasta yhteistyöstä ja aina tarjolla olleista neuvoista. Haluaisin myös kiittää Sähkökiltaa erinomaisista opiskelutiloista, sekä Tampereen Teekkarien PerinneSeuraa unohtumattomista opiskeluaajoista. Ilman teitä opiskeluni olisi ehkä sujuneet nopeammin, mutta teidän ansiosta niistä jäi äärettömän paljon enemmän käteen. Viimeisenä haluaisin kiittää Nokia Oyj:tä mahdollisuudesta kirjoittaa diplomityötä pääsääntöisenä työnäni, jolloin siihen oli mahdollista panostaa kunnolla!

Tampereella, 17. kesäkuuta 2021

Henri Kasurinen

## CONTENTS

1	Introduction . . . . .	1
2	Deep Learning in radio receivers . . . . .	2
2.1	Convolutional Neural Networks . . . . .	2
2.2	Depth-wise separable convolution . . . . .	4
2.3	Nokia’s DeepRx . . . . .	6
2.4	Target hardware . . . . .	12
3	Optimizing DNNs for resource efficiency . . . . .	14
3.1	Pruning . . . . .	14
3.2	Knowledge distillation, Genetic Algorithms and Auto-ML . . . . .	19
3.3	Quantization . . . . .	20
3.4	Quantization-aware training frameworks . . . . .	22
4	Related work . . . . .	24
4.1	Combined quantization and pruning . . . . .	25
4.2	Contributions of this work . . . . .	26
5	Quantization-aware multi-stage pruning . . . . .	27
5.1	Block-wise loop . . . . .	28
5.2	Main loop . . . . .	29
5.3	Quantization-awareness . . . . .	31
6	Experimental Results . . . . .	32
6.1	Experiment setup . . . . .	32
6.2	Results . . . . .	33
6.3	Ablation study . . . . .	35
7	Conclusions . . . . .	39

## LIST OF SYMBOLS AND ABBREVIATIONS

5G	The fifth generation of a technology standard for cellular networks
ANN	Artificial Neural Network
APoZ	Average Percentage of Zero
BER	Bit Error Rate
CNN	Convolutional Neural Network
CP	Cyclical Prefix
DeepRx	CNN-based radio receiver
DMRE	Demodulation Reference Signal
DNN	Deep Neural Network
DWC	Depth-Wise Separable Convolution
FFT	Fast Fourier Transform
FLOPS	Floating Point Operation
IFFT	Inverse Fast Fourier Transform
KD	Knowledge Distillation
LDPC	Low-Density Parity Check code
LLR	Logarithmic Likelihood Ratio
LMMSE	Linear Minimum Mean Square Error
ML	Machine Learning
OFDM	Orthogonal frequency-division multiplexing
PRB	Physical Resource Blocks
SNR	Signal to Noise Ratio
SoC	System On a Chip
TAU	Tampere University
TTI	Transmission Time Interval
TUNI	Tampere Universities
UE	User Equipment
URL	Uniform Resource Locator

# 1 INTRODUCTION

Artificial Neural Networks (ANN) revolutionized the way machine learning models work and exploded their performance. For example, image processing tasks that used to be impossible, are now a part of everyday technology, like camera phones. Since the invention of ANNs, their computational complexity has been a restricting factor in their deployment on devices, where computational resources are scarce. Hence, the idea of compressing ANNs was pioneered already in 1990 by Le Cun et al. [1] The idea of running ANNs with lower precision arithmetics soon followed to further accelerate computation [2]. Innovations like Convolutional Neural Networks (CNN) [3] further sped up ANNs but as Machine Learning (ML) evolved, the ANNs got bigger and bigger. Nowadays the best performing image processing ANNs require up to billions of operations to produce their output. With a powerful Graphical Processing Unit (GPU) this is no problem and the output is calculated in a fraction of a second, but what if such processing power and parallelism is not available? This is the difficulty that Nokia Mobile Networks is facing with an ANN-based radio receiver called Deep Learning Receiver (DeepRx).

Power efficiency and cost are among the most important properties of this kind of radio equipment and hence, the constraint for power consumption is very strict. In fact, the current DeepRx won't meet the requirements even with highly efficient and specialized hardware. Hence, the DeepRx has to be compressed in every way possible without degrading its performance. That is also the purpose of this thesis: To find out a powerful way to reduce the computational complexity of the DeepRx without affecting its accuracy and doing it so that the requirements of the target hardware are taken into account. We focus solely on the compression of the DeepRx model. Critical consideration of the target hardware or the model architecture are out of the scope of this thesis.

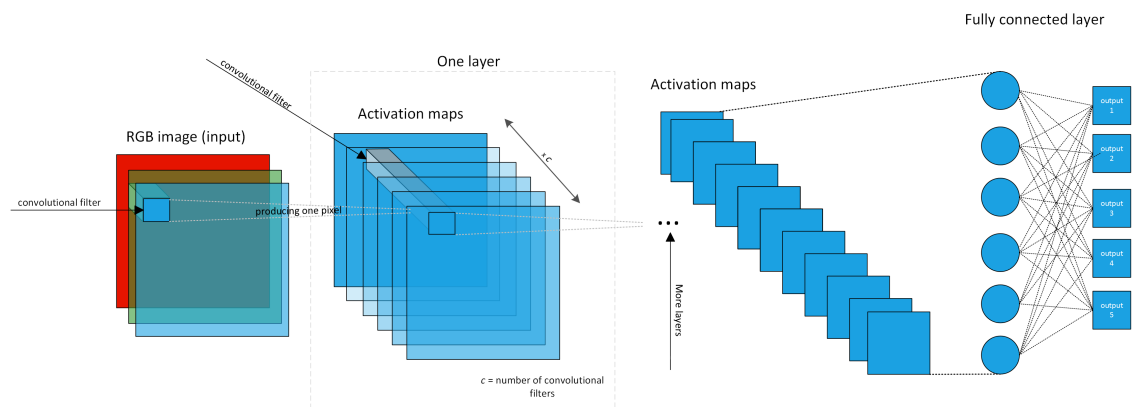
This thesis starts with an in-depth introduction to CNNs and their application in mobile networks, with a special focus on the DeepRx receiver. Different ways to optimize and compress ANNs are discussed and the target hardware and its limitations are introduced in Chapter 3. Chapter 4 reviews existing research that promises to combine quantization and pruning in some way and then our method, QAMP, is introduced and explained in more depth in Chapter 5. Finally, the results of QAMP are reviewed along with a short ablation study and then the conclusions of the thesis are presented.

## 2 DEEP LEARNING IN RADIO RECEIVERS

This chapter gives an introduction to the working principles of Convolutional Neural Networks and reviews different CNN architectures. The chapter is continued by reviewing the basics of modern radio receivers and some use-cases where CNNs have been utilized in them. Finally, Nokia's DeepRx radio receiver is introduced in terms of ANN architecture and hardware implementation.

### 2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have played a groundbreaking role in the rapid progress of ML. CNNs however, are not that new. In fact, the first ANN, which worked like today's CNNs was published by Alex Weibel et al. in 1989. [3] Their "time delay neural network" (TDNN) utilized weight sharing [4] and backpropagation [5], which are both key enablers of CNNs of today. With these methods the TDNN achieved many of the same properties that make the CNNs of today so powerful. To understand the strengths and weaknesses of CNNs, the working principles of them need to be understood. This is depicted in Figure 2.1



**Figure 2.1.** Illustration of a typical convolutional neural network.

Figure 2.1 shows a simplification of the architecture and working principle of a generic CNN. The left-most stacked rectangles represent the input, and the multiple rectangles between the input and output are called activation maps. This example uses RGB images but the input could basically be any data. Each stack of activation maps and the

connection between them are called layers as depicted by the dotted lined box in Figure 2.1. There are multiple activation maps in each layer of the network and each of them represents one channel of the network. These channels are like the red, green and blue channels of the input image: they are different representations of the same image. The convolutional filters are essentially just small matrices, typically 3x3 and they are convolved over the input. In this case convolving means that the filter is "slid" over the input, starting from the top-left corner. The values, eg. pixels-values under the filter are multiplied with the values of the filter matrix on each channel and these values are summed together to produce one value, eg. pixel of a activation map. This multiplication is not only done in the spatial domain, but also over all the channels at once, hence combining spatial correlation and inter-channel correlation.

Yosinski et al. argued that this combination of spatial information and the inter-channel information is what makes CNNs work so great. Spatial information means basically what kind of shapes there is in the image, and inter-channel information means how this spatial information relates to other representations of the image. In other words, different layers find different patterns in the input and these different patterns are stacked together to find abstract concepts, which humans wouldn't even think of searching for. [6]

Good accuracy is not always the only thing that matters. In many cases the number of operations i.e. the computational complexity, is a key restraint which needs to be taken into account. This is often the case when building ML solutions, which are going to be deployed on devices with limited computing power, storage space or where low power consumption is essential. This kind of devices are for example mobile phones, autonomous devices, sensors and radio equipment. In this kind of applications ANNs can't be deployed into the device itself. In some cases it's possible to use a cloud service, where the ANN is deployed on a remote server, and the computation is done there instead of the device. This creates latency into the usage of the ANN, rendering it useless in many applications and it can also create privacy issues. [7]

In order to estimate the possibilities of ML in resource limited devices, it is important to know how large the CNNs actually are. The number of operations that has to be calculated in one convolutional layer (like the one shown in Figure 2.1) of an arbitrary CNN is shown in Equation 2.1.

$$O_c = w_f * h_f * c * w_a * h_a * d_i, \quad (2.1)$$

where  $w_f$  and  $h_f$  are the width and height of the convolutional filter kernel,  $c$  is the channel count,  $d_i$  is the depth of the input i.e. channel count and  $w_a$  and  $h_a$  are the width and height of the activation map resulting from the convolution. The size of the resulting activation map depends on the size of the input, the used kernel size, zero padding,

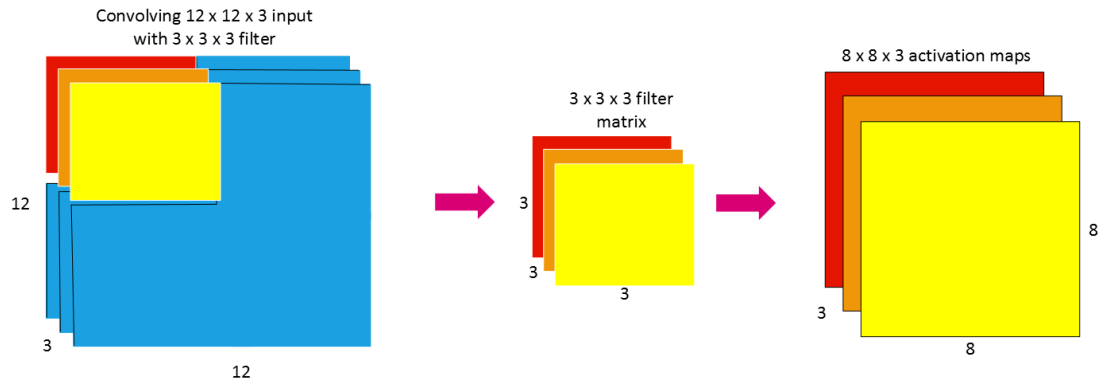
dilation and the stride. Zero padding means that the input of a convolutional layer is "padded" with zeros around the edges, changing the spatial size of the input without changing the information it contains. This makes it possible to change the spatial size of the output of the convolution. Stride depicts the increment that the convolutional filter is moved as the convolutional filter slides across the input. A stride of 1 means that the filter is moved only by one pixel, 2 means that it's moved by 2 pixels and so on. Striding increases the receptive field of the convolution but it also decreases the size of the output. Equation 2.1 assumes a stride of 1. The receptive field represents the maximum spatial area of the input in which a pattern can be detected. Dilation denotes that the convolutional filter is enlarged by inserting zeros between its values. Like Yu et al. stated in [8] striding can be used to increase the receptive field of the convolution without reducing the resolution. A bigger receptive field makes it possible for the model to detect patterns that are scattered on a wider spatial area of the input.

To put Equation 2.1 into some numbers, a normal convolutional kernel size  $w_f * h_f$  is 3x3, there are often hundreds of channels  $c$  in one layer, the input depth  $d_i$  of one layer depends on the channel count of the previous layer, making it likely for it to also be hundreds and the spatial size of the resulting activation map  $w_a * h_a$  can also be in the hundreds. All of this is just for one layer and there can be hundreds of layers in one complete CNN. Hence, we get a computational complexity that is measured in the billions and this amount of operations needs to be calculated on every inference with the network. Radio receivers work on very high frequencies, and hence latency is an important factor in receiver algorithms. In fact there is plans for using frequencies of up to 100 GHz for the 5th generation of radio receivers (5G).[9] Calculating billions of operations with a huge frequency, is not possible within current hardware requirements. This creates a high demand for efficient and fast ML models and this is where different model compression techniques come into play. [10]

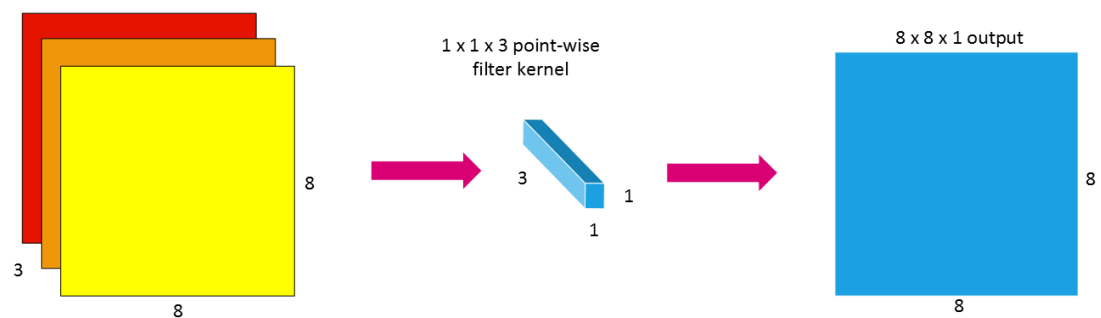
## 2.2 Depth-wise separable convolution

The architecture shown in Figure 2.1 is not the only way to create convolutional layers. Depth-wise Separable Convolutions (DWC) provide the same power as "normal" 2D convolutions, but have a significantly smaller computational complexity. As stated in the previous chapter, convolutional neural networks combine spatial information to inter-channel information by using multiple three-dimensional filters. [6] The idea of DWC is to split these two dimensions into two separate operations: depth-wise convolution and a point-wise convolution.

The idea of DWC was first introduced in 2014 by Laurent Sifre in his PhD. Thesis [11] and has since been successfully used in a variety of applications. One popular ANN architecture using depth-wise separable convolutions is the MobileNet architecture introduced in



(a) Depth-wise convolution.



(b) Point-wise convolution.

**Figure 2.2.** Depth-wise separable convolution consists of two operations: a) depth-wise convolution and b) point-wise convolution.

2017 by Andrew G. Howard et al. in [12]. As will be explained later in this chapter, the main benefit of DWC is that the computational complexity of it is only a fraction compared to conventional 2D convolution. Despite of the lower computational complexity and fewer trainable variables, DWC-based CNNs can reach similar accuracy as "normal" ones and in some cases even outperform them. [12]

The principle of DWC is further elaborated with the help of the illustration of depth-wise convolution in Figure 2.2a and the illustration of point-wise convolution illustrated in Figure 2.2b. As can be seen from Figure 2.2a, the depth-wise convolution uses two-dimensional filters to transform the input in the spatial domain without changing the channel count. It is important to note that there is one dedicated filter kernel for each channel. For demonstrative reasons in Figure 2.2a there is only 3 channels and accordingly 3 filters, but in reality CNNs can have hundreds or even thousands of channels. No inter-channel knowledge is transferred at this point. This means that the channel count stays unchanged, but the depth-wise convolution can change the spatial size of the output like traditional CNNs can. This can also be seen in Figure 2.2a, where the  $12 \times 12 \times 3$  input is transformed to  $8 \times 8 \times 3$  with three  $3 \times 3$  filters.

Like depicted in Figure 2.2b the point-wise convolution uses  $1 \times 1 \times c$  kernels to transform the input in the depth-wise dimension, where  $c$  is the channel count of the input. Usually there is a multitude of these filters in each layer of a network, resulting in as many activation maps as there is filters. This can also be seen in Figure 2.2b, where there is only one filter, which results in one activation map, which has the same spatial size as the input ( $8 \times 8$ ). For demonstrative reasons Figure 2.2b has only one filter, in real life solutions there would be tens or even hundreds of these filters. The combination of one depth-wise convolution and one point-wise convolution creates one DWC layer, as the output of the depth-wise convolution is fed into the point-wise convolution. This creates the same effect as in 2D convolution, where spatial and inter-channel knowledge is combined.

The reason why this type of convolution has a lower computational complexity becomes evident in Equation 2.2 and Equation 2.3.

$$O_{dwc} = w_f * h_f * w_a * h_a * d_i + d_i * w_a * h_a * c \quad (2.2)$$

$$\frac{O_c}{O_{dwc}} = \frac{w_f * h_f * w_a * h_a * d_i + d_i * w_a * h_a * c}{w_f * h_f * w_a * h_a * d_i * c} = \frac{1}{c} + \frac{1}{w_f * h_f} \quad (2.3)$$

Equation 2.2 shows how many operations have to be calculated in one inference with a DWC layer and Equation 2.3 shows the ratio of computational complexities between DWC and conventional 2D convolution. The number of computations in a conventional 2D convolution was calculated in Equation 2.1. As also calculated by Zhang et al. in [13], the ratio is about  $1/9$  when using  $3 \times 3$  filters and a stride of 1. Requiring only  $1/9$  of the compute power while still providing a similar accuracy makes DWC an excellent starting point for creating even more efficient models. In the context of this thesis DWC layers are especially important because they yielded even better accuracy than conventional 2D convolutional layers in the ANN based radio receiver. [14]

### 2.3 Nokia's DeepRx

To understand what is required from radio receiver, it's necessary to understand how the radio receivers used in telecommunication work. The digital data transmission type used in most telecommunication is Orthogonal Frequency-Division Multiplexing (OFDM), which uses closely spaced orthogonal subcarrier signals to transmit data over multiple channels in parallel. OFDM was invented by Bell Labs researcher Robert W. Chang in 1966 [15] and has since become a vital part of mobile networks.

The OFDM transmission starts by encoding the bit-stream to be sent (the payload) with a Low-Density Parity Check code (LDPC), which is put through a rate matching process. This creates a code word which is mapped into so-called OFDM-symbols which are then

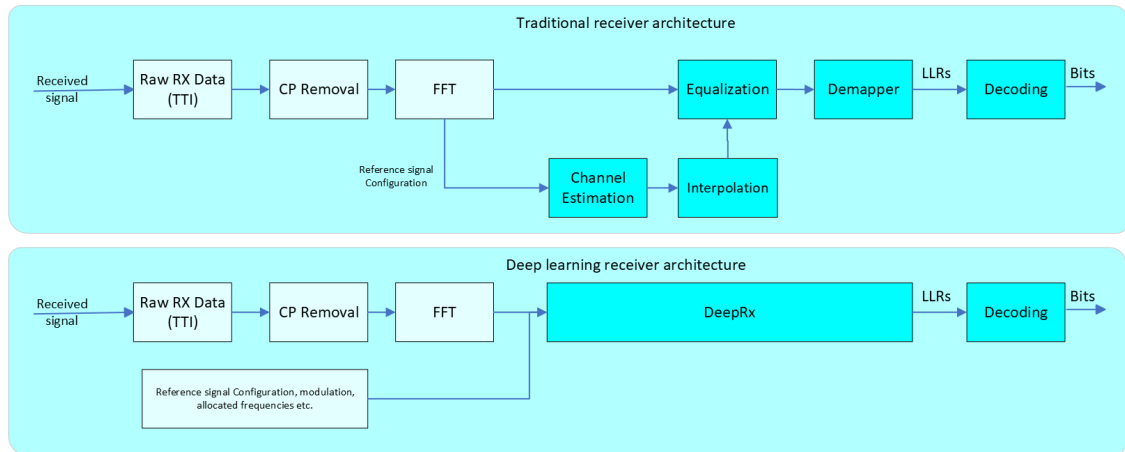
distributed over the Physical Resource Blocks (PRB). A PRB is the smallest unit of transmission capacity that can be assigned to one user equipment (UE) and in the context of this thesis they consist of 12 subcarriers and 14 OFDM symbols. In order to be able to demodulate the signal at the receiver end, Demodulation Reference Signals (DMRS) are added to specified subcarriers. The purpose of these is later explained in more detail. After that the PRBs are transformed to the time-domain by using Inverse Fourier Transform (IFFT), which creates a OFDM waveform. As the last step a Cyclic Prefix (CP) is inserted into the beginning of each of the separate OFDM symbols to separate them from each other. All these steps are done in one Transmission Time Interval (TTI), which is the time it takes for all the subcarriers to send their 14 OFDM symbols. [14, 15] The OFDM transmission can be thought of as a constant stream of matrices that represent a number.

When a signal is sent from the UE to a radio receiver, the signal gets distorted due to multiple reasons. Some of them are characteristic to OFDM transmission like IQ-imbalance and phase noise [16] and others are general disturbance coming from the environment. These kinds of distortions arise for example because of obstacles in the way of the signal, from background radiation, thermal noise and atmospheric radiation. [17] In today's telecommunication it is also common, that the UE is mobile. The UE could be contacting the radio receiver from a car or a train, which creates Doppler shift to the signal. This kind of distortion causes inter-carrier interference, which greatly effects the Bit-Error Rate (BER) of a radio receiver. [18] In order to correctly interpret the received signal, these kinds of distortions and noise in the signal have to be accounted for in the process of receiving the signal.

Basically, the receiving of a OFDM signal needs to contain the same steps as sending the signal, but in reverse order and with the capability to assess the distortion of the signal. The focus point of this thesis is on the receiving of the signal, since that is the task of the ANN that is introduced later in this thesis. There are many ways to implement a OFDM receiver, but in the work of Honkala et al. a traditional OFDM receiver is used as a benchmark. It is also used here to demonstrate what steps need to be done in order to receive a OFDM signal. The traditional OFDM receiver is depicted in Figure 2.3.

As can be seen from Figure 2.3 there is basically 7 main components in the traditional OFDM receivers signal processing path. The first two, Cyclic Prefix Removal and Fast Fourier Transform (FFT), merely demodulate and prepare the received signal for further processing. The Cyclic Prefix is a "buffer" which stops different OFDM symbols to interfere with each other. It doesn't contain any payload so it can be discarded from further analysis. FFT transforms the signal from the time domain to the frequency domain, which makes many signal processing steps possible. [14, 19]

The first step after demodulation is to estimate the channel. Estimating the channel basically means that it is investigated, how the signal is distorted during its journey from



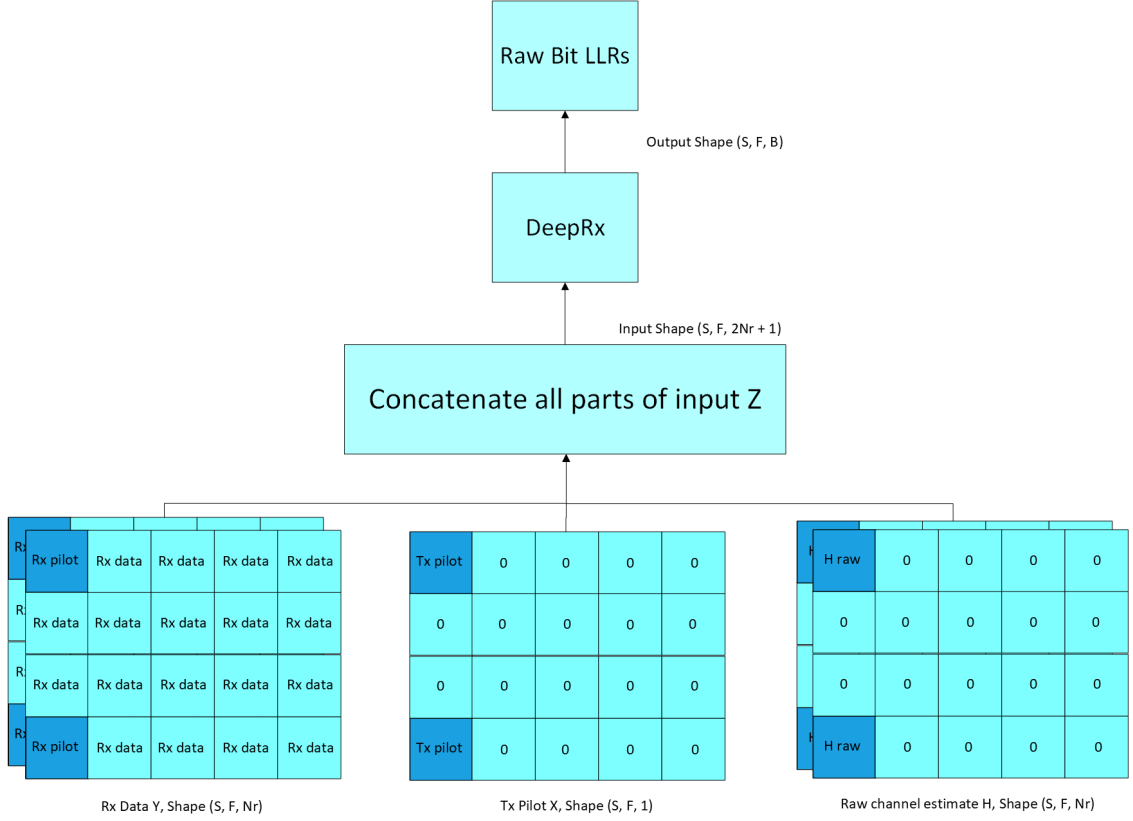
**Figure 2.3.** Traditional OFDM radio receiver architecture vs. the DeepRx [14].

the UE to the receiver. This stage utilizes the DMRS which is sent along with the actual payload. DMRS are signals, which are already known to the receiver so by comparing the received DMRS to the original one, the effect of the channel can be calculated. These signals are also called pilots. The Linear Minimum Mean Square Error (LMMSE) between the un-distorted and distorted DMRS is called the raw-channel estimate and it is interpolated to cover the whole time-frequency grid. This interpolation yields the actual channel estimate. The channel estimate is then used to equalize each data symbol, which are then sent to a demapper. The demapper calculates the log-likelihood ratios (LLRs) for each received bit, which in turn are then converted to bits in the decoder. [14, 19]

Many researchers have proposed to replace one of the presented blocks with an ANN, like for example D. Neumann et al. proposed to replace the channel estimation block [20] and Z. Chang et al. proposed to replace the equalization block with an ANN [21]. While it is intuitive to assign the ANN one well-defined task, Honkala et al. proposed a drastically different approach in [14] where they replaced almost the complete receiver with one CNN. Their work is the foundation on which this thesis is built on.

DeepRx is based on the idea that CNNs can combine information in ways that humans could never think of. As can be seen in Figure 2.3 the network replaces the whole traditional receiver pipeline. The rationale behind this is that by providing the network with as much data as possible and by not dividing the process into separate tasks, the network can find relevant correlations in the data better than traditional methods can. One key element in the DeepRx is the way the data is fed into it, which is illustrated in Figure 2.4. [14]

The input data consists of three parts: The received signal  $Y$ , the pilot reference symbols (DMRS) aligned with the pilots in the received signal  $T_x$  and finally the raw channel estimate  $H_r$ . [14]



**Figure 2.4.** Illustration of the input data of the network [14].

The first part, the Fourier transformed received signal can be denoted like Equation 2.4:

$$\mathbf{y}_{ij} = \mathbf{H}_{ij}\mathbf{x}_{ij} + \mathbf{n}_{ij}, \quad (2.4)$$

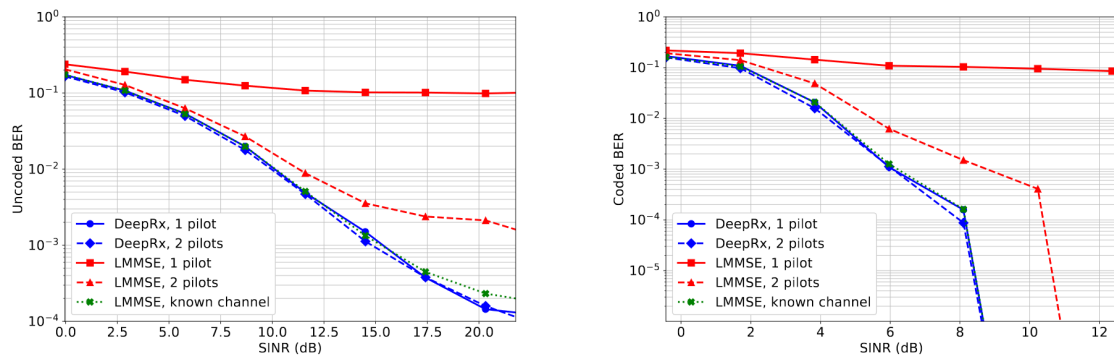
where  $i$  and  $j$  denote the indices of the OFDM symbol and subcarrier,  $\mathbf{y}_{ij}$  and  $\mathbf{x}_{ij}$  are the received and transmitted OFDM symbols,  $\mathbf{H}_{ij}$  is the channel in the  $i$ th OFDM symbol over the  $j$ th subcarrier and  $\mathbf{n}_{ij}$  is the noise and inference in the signal. The shape of the received data is  $S * F * N_r$ , where  $N_r$  is the number of RX antennas,  $S$  is the total number of OFDM symbols and  $F$  denotes the number of subcarriers. [14]

The second part is a matrix  $T_x$ , which spatial shape is the same as the first parts, but it is only two-dimensional. It contains pilot reference symbols positioned so that they correspond to the DMRS positions within the received signal. The pilot reference symbols are the same as the ones inserted into the received data before they were sent from the transmitter. This way the model gets the information how the signal is distorted during the transmission. [14] The received and the reference symbols can be used to obtain the third part of the input data, the raw channel estimate  $\mathbf{H}_r$  and it's calculated:

$$\mathbf{H}_r = \mathbf{y}_{ij}\mathbf{x}_{ij}^c \quad (2.5)$$

$\mathbf{y}_{ij}$  and  $\mathbf{x}_{ij}$  are the received and transmitted OFDM symbols where  $i$  and  $j$  denote the indices corresponding to pilot locations in the time-frequency grid and  $(\_)^c$  denotes the complex conjugate. Like in RGB-pictures, where the different channels correspond to different color channels in the picture, all the layers in the input data of the DeepRx are related but different to each other. They all bring some new information to the equation. By stacking the data into channels i.e. separate layers in the three-dimensional matrix, allows the convolution to combine and use all this data. [14]

In this thesis we focus only the model presented in [14], which is designed to use only one receiver antenna. A bigger version of the DeepRx model, which supports multiple antennas is proposed in [22] but due to restrictions in computational resources, the compression of that model is left as a future work item. The DeepRx model is a pre-activation ResNet like introduced by He et al. in [23] consisting of 11 residual connection blocks. Like shown in Figure 2.4 the output of the network is a vector of raw bit LLRs, one for each of the received bits of the symbols. Hence, the spatial size of the received input data is the same as the output of the network. This is why striding and maxpooling are not used in the network. Instead, the receptive field of the convolutional layers is increased with dilated convolutions. This allows the convolutions to detect patterns that are spread over a wider spatial area of the input. The number of filters is increased in the middle of the network and then decreased again towards the end. This architecture allows the network to capture long lasting dependencies while still maintaining detailed information of the symbols. [14] The radio performance of DeepRx can be seen in Figure 2.5.



**Figure 2.5.** Radio performance of DeepRX compared to traditional receiver architecture and a "all knowing" reference receiver [14].

The performance is measured by uncoded and coded Bit Error Rate (BER) and Signal to Noise Ratio (SNR). The uncoded or "raw" BER is measured based on the value of the LLR of each bit. In this case the bit is interpreted as being 1 if LLR < 0 and 0 otherwise. The coded BER on the other hand is obtained by feeding the LLRs into a 5G-compliant decoder, which outputs either 1 or 0.

Figure 2.5 contains 5 curves: DeepRx with 1 and 2 pilots (DMRS), a "traditional" OFDM receiver utilizing Linear Minimum Mean Square Error (LMMSE) for the channel estimation with 1 and 2 pilots (DMRS) and a theoretical, optimal receiver, which has perfect channel knowledge. The last receiver is not possible to achieve in practice and is used here as a benchmark. SNR measures the quality of the signal and is calculated as the ratio of the power of the signal to the power of its noise. The lower the SNR value is, the worse the signal. As can be seen from Figure 2.5 the DeepRx over-performs the traditional receiver with a significant margin and is on par with the "magical" bench mark receiver. [14] It is also important to note that the DeepRx performs equally good with one pilot and two pilots. By using only one pilot, the space taken up by the second pilot signal is freed up to be used for payload, which increases the transmission bandwidth of the receiver.

The results are very encouraging and show that this kind of receiver would increase the accuracy of radio receivers significantly. While the DeepRx works well in theory, the practical implementation is not that simple. Price and power consumption are important factors in the competitiveness of telecommunication equipment. The whole receiver is implemented on one System on Chip (SoC), which is created in a tedious process out of silicon. Producing these SoCs is very expensive and the price of them can be minimized by minimizing the used silicon area. Power consumption on the other hand directly affects the running cost of the equipment. Hence, silicon area and power consumption need to be minimized in order to create a competitive product.

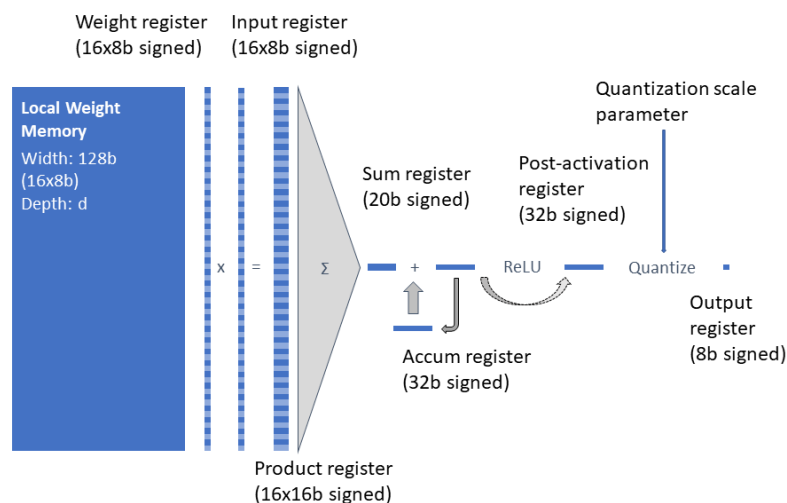
The "small" DeepRx introduced previously in this chapter contains about 650 000 parameters and one inference with it requires  $5,7 * 10^9$  operations, 5,7 GOPs. [14] As a reminder, the model supports only one receiver antenna, where the actual receiver needs to support multiple antennas. The larger version for multiple antennas introduced in [22] is over 4 times larger and needs a "pre-DeepRx" CNN in order to work. The power limit for the hardware for running the DeepRx will be around 30W, which translates with the specialized hardware introduced in Chapter 3 to a operations limit of around 4,5 GOPs [24]. Even the "smaller" DeepRx won't fit into this limit without compression.

The power limit may change in the future and more specific details of it are not disclosed. One factor however, is evident: effective model compression is dearly needed. Since the model used in the scope of this thesis is not the model that will be deployed in the equipment, the numerical goal for compression in terms of power consumption or operations count is not defined at this point. Instead, the purpose of this thesis is to investigate how much the computational complexity of the model can be reduced without greatly affecting the accuracy of the model after quantization.

## 2.4 Target hardware

The efficiency of an ANN application depends greatly on the compatibility of the hardware and the architecture of the ANN. The target hardware for the DeepRx is a programmable Machine Learning accelerator that can execute a limited set of tensor operations. The target is to achieve a 100% utilization rate, which means that all of the hardware's computational resources are fully utilized. This however, is only possible when the architecture of the model and the hardware are perfectly aligned. Hence, the pruning has to be done so that the architecture of the model still fits the hardware. For example, the hardware benefits greatly from knowing the shape of input tensors and from the uniformity and structure of the model and its parameters. [24]

Figure 2.6 illustrates the core concept of how inference with the model is done. One characteristic of the application of DeepRx is that inference needs to happen very fast, up to 36000 times in a second. Hence, it is not possible to use external memory to store the weights. Instead, the weights and biases have to be stored on-chip using SRAM (Static Random Access Memory), which is depicted in Figure 2.6 as the blue box on the left. The exact dimensions of the memory are not disclosed, but in this canonical example, the width of it is 128 bits, meaning that it contains 16 8-bit weights in each row. On inference all the weights of one layer are read at once from the local memory and then multiplied with the input of the layer. After the multiplication the result is written into a product register and then summed up to produce the output of the layer. [24]



**Figure 2.6.** 8-bit vector dot-product pipeline with local weight memory [24].

As will be later explained, pruning can be done in two ways: in a structured or unstructured way. The unstructured way means basically replacing weight values by zeros, which

creates a so-called sparse structure. With this hardware however, this can type of pruning can not be used. The reason why sparsity can't be utilized with this hardware is due to the structure of the dot-product pipeline. To benefit from sparsity, the weight matrix should be searched for zeros in some way in order to discard them before processing. With the target hardware all the weights of one layer are loaded from the local memory at once and multiplied with the input regardless of their value. Hence, a sparse weight tensor would actually lower the utilization ratio, and hence the efficiency. A weight with the value of zero doesn't contribute to the output, but still takes up memory space, requires power to be read and requires power when the output is calculated. All this energy could be used for a non-zero weight value, which would also contribute to the output and hence, increase the accuracy. [24]

Consequently, the best possible way to prune the DeepRx is to avoid zero-valued weights and to use channel pruning. However, the channel pruning must happen on the terms of the hardware. For the canonical example of Figure 2.6, this means that the pruning must progress in 16-channel steps in order to hit the right memory size and achieve 100% utilization. For example if a layer has 64 channels and it's being pruned, the pruned channel count should be 48, 32 or 16. If the algorithm would prune 15 channels, the weight tensor wouldn't fit the size of the memory, resulting in a lowered utilization rate and efficiency.

### 3 OPTIMIZING DNNs FOR RESOURCE EFFICIENCY

Most modern CNNs can be described as Deep Neural Networks (DNNs), because they consist of multiple relative narrow layers, creating a "deep" pipe-like structure. While their accuracy in various tasks is great, their weakness is their huge size and energy consumption. [7, 10] CNNs are however generally very resilient once they are trained and can be compressed in many ways. K. Paupamah et al. stated in their work that ANN compression can be divided into 3 parts: pruning, quantization and using efficient network structures. The latter mentioned includes the utilization of depth-wise separable convolutions like introduced earlier. Paupamah et al. concluded that using such efficient structures gives a better starting point for compression. [25] After creating an efficient ANN architecture, an intuitive solution for further compression is Neural Network Pruning.

#### 3.1 Pruning

The idea of pruning is to reduce the size of an ANN by systemically removing parts of the model. Depending on the way pruning is done, it reduces the amount of memory required to store the model and it can also reduce the amount of memory accesses, since there are fewer parameters to fetch from memory. [7, 10] One interesting finding about pruning that Gale et al. made in [26] is that the ratio of model accuracy to model size that is achievable with pruning, is not possible to achieve by just training a smaller network from scratch. We also came to the same conclusion in our own work. In other words, pruning finds a suitable architecture for a specific initialization of the network.

Algorithm 1 shows the basic working principle of most pruning algorithms. The network is divided into elements which can be removed. These elements can be single parameters or structural components of the network like filters in CNNs. On line 4 of algorithm 1 each of these elements are given an "importance score". This score can be the absolute value of the parameter or the sum of absolute values in a convolutional filter. Also on line 4, every value that has a low score is removed from the network. Since removing parameters of a neural network reduces its accuracy, the network is re-trained or "fine-tuned" on line 5. This is repeated until the wished amount of values is removed. Some so-called "one-shot" pruning algorithms have been proposed, which complete the iteration of algorithm 1 only once, but their performance is generally worse than iterative algorithms. [10]

---

**Algorithm 1:** General pruning algorithm

---

**Result:** Elements of the model are removed based on a score

```

1 V := Trained network;
2 N := Number of pruning iterations;
3 while  $i$  in range(1, N) do
4   | v := Prune(Score(v));
5   | train(v)
6 end

```

---

Different to algorithm 1 some pruning strategies, like the one introduced by Gale et al. in [26], prune parts of the ANN already during training. Wang et al. even proposed to prune the network before it is trained, just after it is initialized and then train the pruned network from scratch. [27]

Pruning strategies can be divided into two distinct groups based on what elements of the network are being removed. So-called unstructured pruning strategies remove single parameters and structured strategies remove complete structural components of the network, like for example convolutional filters. Unstructured pruning creates a so-called sparse network structure, because individual connections are removed. Hence, unstructured pruning often relies on special hardware or software to utilize the theoretical efficiency gain they create. Hence, unstructured pruning can in the end be more complex to implement than structured strategies. [10] To be more precise, a sparse architecture means that a part of the parameters of the network are zeroed-out, their value is changed to zero. The distinction between increasing sparsity and removing structures is very important, since in literature the achieved compression with unstructured pruning is often stated to be larger than with structured pruning. This however, is to some extent misleading. Unstructured methods rarely actually *remove* anything from the network, they just zero-out specific weights. The zeroed-out weights don't contribute to the output of the network and their memory footprint can in some cases be compressed, but they are still part of the network and create unnecessary operations. For example, in CNNs zeroed-out weights still produce activation maps that need to be calculated and then stored in the memory to calculate the output of the next layer. Structured pruning strategies on the other hand, actually remove grouped structures of the network like filters or channels.

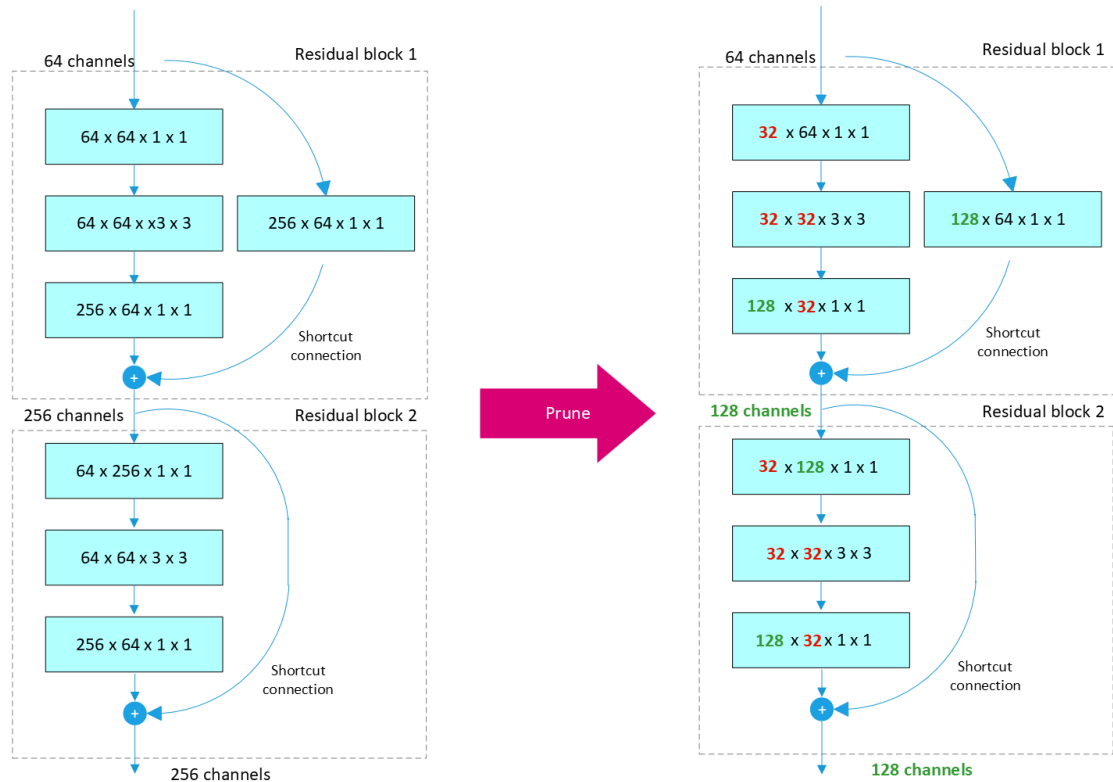
One key difference between different pruning strategies is also the scoring function. In algorithm 1 the scoring function is depicted on line 4 and it decides what parts of the network need to be pruned away. The proposed scores vary from the simple absolute value of the parameters [7] to the contribution to the network activation [28]. Also the definition of the threshold for pruning differs between strategies. Some, like Lee et al. in [29], consider the scores globally across the whole network so that the elements with the lowest scores globally are removed, while others remove a certain percentage of the lowest scored elements locally on each layer like Han et al. in [7]. One interesting

finding of Gale et al. in [26] is that the simplest scoring processes often work best. For example magnitude pruning, where only the absolute value of the weights affects pruning decisions, was found to perform better than its more complex rivals.

Different networks and layers require different pruning strategies and there isn't a one-fits-all solution. For example, the most common type of score for structured pruning of CNNs is some sort of function of the filter matrices, like the L1-norm. [10] However if the network to be pruned uses DWC layers, then this kind of score can't be used. Like explained in chapter 2, DWC layers compose of multiple operations, which means that separate filters can't even be removed, instead the whole channel has to be removed at once. DWC layers require a different approach where the whole channel is taken into account in the scoring. This is further elaborated later in this chapter. The easiest structure to prune is a fully connected layer, which are by default over-parametrized. That is also one reason why it is hard to compare pruning strategies to one another, since they are tested and designed on different networks consisting of various layers. [10]

The amount of compression a pruning strategy is able to achieve can be measured in many ways depending on what the goal of pruning is. There is no industry standard on how to compare different pruning methods to one another and the effects of pruning depend on multiple aspect. For example, the goal could be just to reduce the storage space needed for the model, where a sparse structure can be useful. Pruning alone is not guaranteed to reduce latency or lower power consumption, since the structure and design of the hardware plays also a key role in the efficiency of the final application. [10, 26] The best outcome is achieved by taking the limitations of the hardware into account in the optimization and design process of the model and vise-versa. The actual unit how the compression is measured varies from how much storage space the model takes up measured in MB to how many operations are required for one inference with the model, measured in Floating Point Operations (FLOP). Popular measures are also the number of variables in the model and the drop in latency. [10] To wrap up, it's hard to compare pruning methods to one another based on literature, the only way to find out how a specific pruning strategy affects a specific model on a specific set of hardware, is to implement it and try it out. This finding is also supported by Blalock and Gale et al. [10, 26].

**Pruning networks with residual connections** creates new challenges as the connection combines different parts of the network, which have to match in size. Unstructured pruning methods don't need to take this kind of structures into account since they don't change the number of channels in the network. Structured pruning methods however do. The residual connection was created in order to reduce the "vanishing gradient problem", which makes it impossible to train very deep neural networks without a structure like the residual connection. The first such network was created by Srivastava et al. [30] and it utilized a "high way connection" to add identity mappings together. The problem the residual connection creates is illustrated in Figure 3.1.



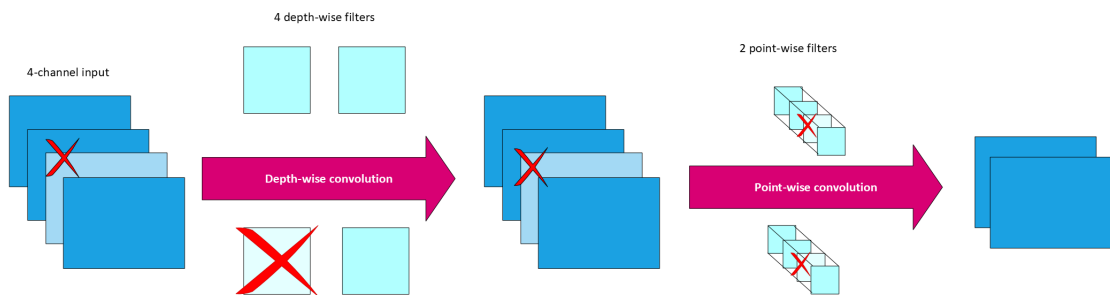
**Figure 3.1.** Illustration of why residual connections make pruning harder [31].

The structures depicted by dotted lined boxes shown in Figure 3.1 are called residual blocks, which are the building pieces of residual neural networks, or Resnets. Basically there is two types of these building blocks: those which change the number of channels and those who don't. In Figure 3.1 the residual block 1 changes the number of filters with a point-wise convolution and residual block 2 keeps the number of filters unchanged. In residual block 1, where the number of filters is changed, the number of filters in the output of the block has to equal the number of filters in the point-wise convolution of the shortcut connection. Equally, the output of residual block 2 has to have the same number of filters as the input of the block. This means that if a number of filters is pruned in any input or output of a block, the same number of filters needs to be pruned in multiple other parts of the network as well. [32] Hence, some filters can't be pruned and some have to be pruned despite their importance score being over the pruning threshold. The problem can be solved by either not pruning the last layer of the blocks or by pruning also the input of the next block and the point-wise convolution of the shortcut connection in accordance of the pruning of the output of the last block.

The pruned channel counts are shown in Figure 3.1 with two colors: green and red. The channel counts affected by pruning of the output of block 1 are shown in green and the others are shown in red. The channels inside of a block can be pruned (red numbers) without affecting other parts of the network, but if the output of block 1 is pruned, then

the input of block 2 and the convolution in the shortcut connection of block 1 have to be pruned as well (green numbers). [32] The latter approach is harder to implement and contains uncertainty and hence many researchers end up using the first approach, where the outputs of each block are kept un-pruned. For example Hu et al. decided in [33] to leave the last two layers of each block un-pruned.

**Pruning networks that utilize depth-wise separable convolutions** is very different to the pruning of conventional 2D convolutions. Most DWC networks not only contain the fore-mentioned residual connections, their core idea of working complicates pruning a great deal. Pruning of DWCs is illustrated in Figure 3.2.



**Figure 3.2.** Illustration of why depth-wise separable convolution makes pruning harder.

Like shown in Figure 3.2 the amount of filters in a DWC block affect not only the amount of filters in the output but also the amount of filters in the input. [34] If during pruning a decision is made that the 3rd convolutional filter in the point-wise convolution is redundant and it should be pruned away, then the same filter has to be removed not only from the point-wise convolution, but also from the depth-wise convolution and the input of the whole layer. This is why pruning of DWC layers has to rely on either increasing the sparsity by setting individual weights to zero or by using channel pruning methods which remove complete channels from the layers. This means removing the same filter from the depth-wise convolution and the point-wise convolution. Additionally, the same filters need to be removed from the input of the next layer, i.e. the depth-wise convolution of the next layer, because the number of channels in the input has changed due to the pruning. [34]

Like explained in chapter 2 the architecture of Depth-wise separable renders many "traditional" pruning strategies unusable. Channel-wise pruning however suits DWC layers in many cases well. Numerous methods are proposed and their research concentrates mainly on the topic of how to determine the importance of each channel. For example, Hu et al. proposed a channel pruning method, that calculates the importance of each channel based on a value called APoZ (Average Percentage of Zeros) in the activation maps produced by the channel. The idea is that a relatively un-important channel produces an activation map with a greater amount of zeros, than a channel that has an greater impact on the output of the model. [35] The sum of absolute values of the weights in a channel has also been a popular choice for an importance score. It has been introduced by Cheng

et al. in [34]. Basically, the idea is very similar to the APoZ method, but the difference is the place where the answer is searched for. If the sum of weights is small, then the effect of the channel on the output of the network is also inevitably small. The absolute sum of weight values has also been found to be a very effective pruning score, despite its simplicity. Gale et al. found in [26] that magnitude pruning actually over-performs more complicated pruning algorithms in many pruning cases. Luo et al. suggested an original approach by presenting a pruning strategy that examines the correlation between different activation maps. They argued that if a channel's activation map has a great correlation with the previous layer, then the importance of the channel is relatively high. [36] Zhang et al. proposed to assess the channel importance by calculating the entropy of the resulting activation maps. Their hypothesis was that the more entropy an activation map has, the more information it transfers. Hence, they pruned the channels which produced the activation maps with the lowest entropy. [37]

### 3.2 Knowledge distillation, Genetic Algorithms and Auto-ML

The point of pruning is to create a smaller model but pruning is not the only method that is proposed for the job. In pruning literature Knowledge distillation, Genetic Algorithms and Auto-ML solutions are often left out of comparison. Unlike pruning, they don't make an existing model smaller, but they attempt to create a small model from scratch.

**Knowledge distillation (KD)** is a way to train a small model also called student model with the help of a larger model, a teacher. The training starts out by training the teacher model to convergence and then use it to teach the student model from scratch. For example, Tung et al. achieved very promising results in compressing MobileNets in their work. [38] Some attempts to achieve Quantization-aware knowledge distillation also exists, but due to the regulating effect of KD they tend to get stuck in a local optimum during training. Kim et al. managed to get over that problem by starting out with a quantized student model and by dividing the learning process of the student into multiple parts. [39] As Oguntola et al. showed in their research [40], KD and pruning can be combined to achieve greater compression rates. Knowledge distillation is a very promising compression technique and is left as a future research item.

**Auto-ML solutions** mean in the context of this thesis the process of utilizing a ML model to determine the architecture of a completely different ML model. These approaches promise to find the smallest possible ANN architecture for a specific task. For example, Wang et al. proposed in 2020 that their auto-ml solution finds the most efficient quantizable ANN for the Imagenet dataset [41] with state-of-the-art accuracy. [42] The problem with auto-ml solutions is that according to our tests and the tests of Gale et al. in [26], auto-ml solutions can't produce the most efficient solution. If two ANNs would be trained to convergence, one small in the beginning and one pruned to be the same size as the

small one, the pruned one would be more accurate. Auto-ML solutions find amongst millions of possible ANNs the most efficient one, and then train it from scratch. This doesn't remove the fact that a small model is trained from scratch. Hence, they are not likely to achieve the same kind of efficiency achievable by pruning.

**Genetic algorithms** on the other hand, mimic the effect of natural selection and mutation by initializing ANNs at random and then picking the best performing ones and creating new initializations by "cross-breeding" them. Abotaleb et al. proposed a pruning method where the "genes" of each individual model contain amongst other parameters the number of filters in each layer of the network. By iteratively training networks with different amounts of filters, picking the best performing ones and then randomly combining them, they achieved good pruning results on GoogleNet and AlexNet architectures using the ImageNet dataset. [43] The weakness of this kind of pruning is that it requires powerful computing devices and a very long training time. Also, there is very few studies done on the subject. Due to the lack of information and our lack of computing resources, genetic algorithms are not considered further in this thesis.

### 3.3 Quantization

The calculation of gradients during training of ANNs requires high-precision and hence, most ML models are trained by using 32-bit floating point arithmetics. This means that each weight and "pixel" of an activation map is represented by 32 bits. All mathematical operations done during convolution, are done by calculating with 32-bit floating point values and also the network along with its weights are stored with 32-bit precision. The idea of quantization is to reduce this precision, which lowers the storage footprint of the model and makes the model more efficient. Quantization often achieves many of the goals that also pruning tries to achieve. A popular bit-depth is 8-bit but research for lower bit-depth quantization has been proposed too. One example of a quantization method that can do under 8-bit quantization is [44]. Even binary networks, where all the weights consist of zeros and ones, have been proposed like reviewed extensively in [45].

Most quantization methods utilize a so-called code book, which is a vector with  $K$  values  $C = [c[1], c[2], \dots, c[K]]$  containing all possible values in the desired bit-depth. Quantization maps the full-precision values  $W_f$  to the codebook so that the actual (32-bit) values don't have to be saved. Instead, only the codebook  $C$  and the index  $I_{wh}$  that each weight corresponds to are saved. At inference, the weight matrices are re-constructed by reading the quantized weight value  $wq_{hw}$  from the codebook from the index specified by the mapping:  $wq_{hw} = c[I_{hw}]$  [46]. Quantization always creates error in the form of deviation from the trained values. A quantized value  $wq_{hw}$  can be written in the form:

$$wq_{hw} = w + n_w \tag{3.1}$$

where  $n_w$  is the quantization noise. Each weight produces this noise as they are quantized, hence the noise accumulates throughout the network. Lin et al. found in their research that each layer contributes to the total noise linearly. This means that if the numbers of layers is doubled, then the noise is also doubled. They concluded that layers with more parameters should use relatively lower bit-depth as it leads to better compression under the overall quantization noise constraint. [47] Ge et al. formulated Equation 3.2 to calculate the actual value of quantization noise.

$$E = \sum_x \|x - C[I]\|^2, \quad (3.2)$$

where  $\| \cdot \|$  is the euclidean distance,  $x$  is a value (weight or activation map) to be quantized and  $C[I]$  is the value in the  $I$ th index in the codebook  $C$ . The equation generalises for any quantization method. [48] The major differences between quantization methods come from how many weights they quantize at once. Fixed-point Scalar quantization quantizes one value at a time. The other popular choice is product quantization, which quantizes several values at once, making each value of the codebook a vector. [46]

**Fixed-point Scalar quantization** maps all the weight values and values of the activation maps to single integers  $q$ , of which there is  $2^N$ , where  $N$  is the desired bit-depth. [44] These values are evenly spaced by a scale factor  $s$  and shifted by a bias or zero-point  $z$  like shown in Equation 5.1.

$$q = \frac{r}{s} + z, \quad (3.3)$$

where  $r$  is the real value to be quantized. The compression ratio of this quantization scheme is  $32/N$  assuming the original weights are 32-bit values. [46] The way  $s$  and  $z$  are calculated varies between quantization techniques in terms if they are calculated for the whole network at once, for separate layers or single elements like filters. Jacob et al. for example calculated in [44] dedicated parameters for each activation map and each filters matrix.

**Product quantization** maps values  $w_{hw}$  to codewords  $c$  from a codebook  $C$ , which is basically the same process as fixed-point scalar quantization. The difference is that product quantization first divides all the values to be quantized to groups and constructs a dedicated codebook for each group. To be more specific, first the weight matrices are grouped amongst its columns and then these columns are split into subvectors. Each quantized vector is then obtained by assigning each subvector to the nearest codeword of the codebook. The codebook is learned by using k-means clustering with  $K$  centroids, where  $K$  depends on the used bit-depth.

In the context of product quantization, learning means that the centroid of each cluster is updated after every forward pass with the sum of gradients of weights in the cluster. The actual value that is assigned to the weights can still be a floating-point value. The benefit comes from one value being shared by multiple weights, which makes it necessary to save just a fraction of the weights: only the centroids of the codebooks and the indexes corresponding to them and the right place in the weight matrix. [48] Traditionally all the weights and activations are quantized at once, but more recent studies like [49] suggest that the quantization should be done in an iterative way. This means that first the lower layers of the network are quantized while maintaining higher layers in full-precision. Then the network is trained so that the higher layers can adapt to the quantization error of the lower layers. Once the network is fine-tuned a new portion of the higher layers are quantized and the fine-tuning is continued. This is continued until the whole network is quantized. Thanks to the adaptability to quantization noise, iterative product quantization can in many cases achieve better compression than fixed-point scalar quantization. [46]

In the case of the Deep Learning Receiver, the hardware only supports 8-bit integer arithmetics. Hence, using product quantization as such can't be applied. Additionally, since the weights are in the end still in floating-point precision, the activation maps get floating-point values, making the run-time memory requirements high. But there is a way to combine fixed-point scalar quantization and product quantization. The idea is quite simple: first calculate the centroids for each subvector of the weights of a network, then quantize these centroids to 8-bit integers using the Equation 5.1. This centroid value is then shared by all the weights in one subvector, hence combining the high compression of product quantization and the efficiency of fixed-point quantization.

### 3.4 Quantization-aware training frameworks

Like pruning, quantization is most often done after training a model to convergence, like proposed for example by Lin et al. in [47]. Some researchers however, have proposed works which incorporate quantization to the training process. These include the work of Fan et al. [46] and the quantization-aware training of Tensorflow [50], which is based on the work of Jacob et al. [44]. It has to be noted that these methods do not produce a quantized model, but they prepare the model to be quantized after they are trained. If quantization is taught to the ANN already during training, the network is more "prepared" to be quantized. Some of these training frameworks also combine pruning to the training process. This kind of approach was proposed by Ding et al. in [51]. They proposed a training framework where quantization and pruning are taught to the network already during training. Ding et al. claim that this renders the popular fine-tuning step useless, since the network is already pruned and quantized during training. The coupling of pruning and quantization is done by alternately exposing the network to them both during

training. Facebook researches Angela Fan et al. also proposed a way to teach an ANN to work with quantized weights and pruning.[46]. Fan et als. idea is very similar to Ding et als and Jacob et als. The difference is that Fan et al. only quantizes or zeroes-out a small random set of weights during training. Ding et al. on the other hand quantized or pruned the whole network. [46] This kind of training frameworks are usable for any ANN, including the DeepRx.

## 4 RELATED WORK

ANN compression is a widely researched field but it lacks maturity. As stated by Davis Blalock in [10] and Gale et al. in [26] the published papers rarely include extensive comparison to other work. Also there are no universal requirements for pruning methods and most of them are designed for specific ANN architectures. Hence, it is hard to find ready-made solutions that would fit the need of Nokia's DeepRx. This section presents proposed pruning methods that are quantization-aware in some way or quantization methods that can also take pruning into account. Each of the methods are assessed on three criterion:

1. The method is applicable on DWC layers.
2. The method uses structured pruning, i.e., it doesn't rely on increasing sparsity by zeroing out values, but actually creates a smaller CNN by removing channels.
3. The method takes into account the combined effect of pruning and quantization.

The first requirement is necessary because of the earlier mentioned structural differences between DWC layers and conventional 2D convolution, which renders many pruning methods unusable the DeepRx. Also K. Paupamah et al. found in [25] that CNNs that utilize DWC layers are more sensitive to pruning. Hence, it is justifiable to consider the pruning of DWC layers as a separate topic to pruning conventional 2D convolutional layers. The second criterion is necessary because, unstructured pruning methods do not achieve improvement in efficiency without separate hardware or software accelerators and the hardware used for the DeepRx doesn't support one. This is further elaborated in Section 2.4. The third requirement is necessary because as stated by Choukroun et al. in [52] more compact networks, like pruned networks, are more sensitive to quantization. The hypothesis is that by taking quantization into account already during the pruning process, or vice-versa, the negative effects of both can be lowered. The ideal pruning algorithm for Nokias DeepRx would take all three criterion into account.

## 4.1 Combined quantization and pruning

Song Han et al. proposed in 2016 a model compression pipeline consisting of three stages: pruning, quantization and finally Huffman encoding. They stated that with this extensive arsenal of compression techniques they were able to achieve up to 49 times smaller models with no degradation in accuracy. [7] Their pruning stage consisted on S. Hangs earlier research published in 2015. [53] The motivation of this pipeline is that the pruning method induces sparsity, essentially increases the amount of zeros in the weights of the model. This kind of model is easier to compress in terms of required storage space. The pruning in Hans compression pipeline is simple: The network is trained to convergence and then iteratively the connections with the smallest values are zeroed-out and the network is "fine-tuned" by training it to compensate for the pruning. Once pruning is done, the network is quantized with product quantization and additionally weight sharing is applied to increase the compression. After quantization the models weights are Huffman coded, which means that weights are represented with codewords, which makes it possible to represent most frequently used weights with fewer bits.

A recent paper published by F. Tung and G. Mori introduced a method called CLIP-Q which attempted to actually combine quantization and an unstructured pruning strategy. [54] Tung and Mori argued that pruning and quantization actually have a "complementary nature" and they can be done in parallel. The process of CLIP-Q starts with a trained model, which is then quantized so that the values falling to the buckets closest to 0, get the value of 0. The loss is calculated over a minibatch during the forward pass with the quantized and pruned weights, which forces the network to learn to perform with the lower bit-depth and increased sparsity. For the backpropagation the weights are again converted to their full-precision so that gradients can be calculated. After this process the network is fine-tuned by training it for some iterations until the quantization-pruning process is repeated. The pruned values are not forced to stay 0, making it possible for pruned values to "come back" if the connections become important as the pruning progresses. Tung and Mori also introduced a Bayesian optimization process for determining the pruning-quantization hyperparameters. A similar approach to their work is the work of van Baalen et al. [55]. Basically, their main focus is on quantization and deciding the optimal bit-depth for each layer separately to obtain a mixed-precision network, where each layer is represented with as low bit-depth as possible. One possible bit-depth option is 0 bits, which essentially means that some weights are pruned from the model.

These approaches promise great pruning ratios with little to none decrease in accuracy. Their problem, however, is that they rely on increasing the sparsity, just converting values to 0. Hence, they do not fulfill criterion 2. As stated many times in this thesis the theoretical efficiency gain of sparsity can't necessarily be translated into more efficient real-life solutions. [26] Also, like in the case of this thesis, the pruning algorithms are

mainly designed to fit one specific ANN. Hence, they perform well in pruning some specific ANNs, but then deliver far worse results on some other ANNs. This makes it hard to compare pruning algorithms to each other. The term "quantization awareness" is also not well defined. For example, in the work of Han et al. they merely do quantization and pruning successively but they are not connected or "aware" of each other in any way. [7] Also, none of the "quantization-aware" pruning methods have been tested on DWC layers, which makes the fulfilment of criterion 1 questionable.

Some pruning strategies however, are not quantization-aware as such but their core idea is universal in a way which can be applied also to quantization-awareness. One example of such pruning strategy is Luo et als. "CURL method" proposed in [31]. Their core idea is to first test the effect of pruning by zeroing out the connections they intend to prune. Then effect of the pruning is assessed by feeding a minibatch through the model. If a significant drop in accuracy is not detected, the weights can be pruned, but otherwise the weights are restored to their original values. Even though the work of Luo et al. doesn't fulfill the criterion 2 or 3 as such, the idea of testing the performance of the model with a minibatch before actual pruning can also be applied to Nokias DeepRx. It can also be used on testing the effect of quantization during the pruning and quantization process. Also, the quantization-aware training processes of Fan et al.[46] and Jacob et al. [44] are universal and can be used also for DWC layers.

## 4.2 Contributions of this work

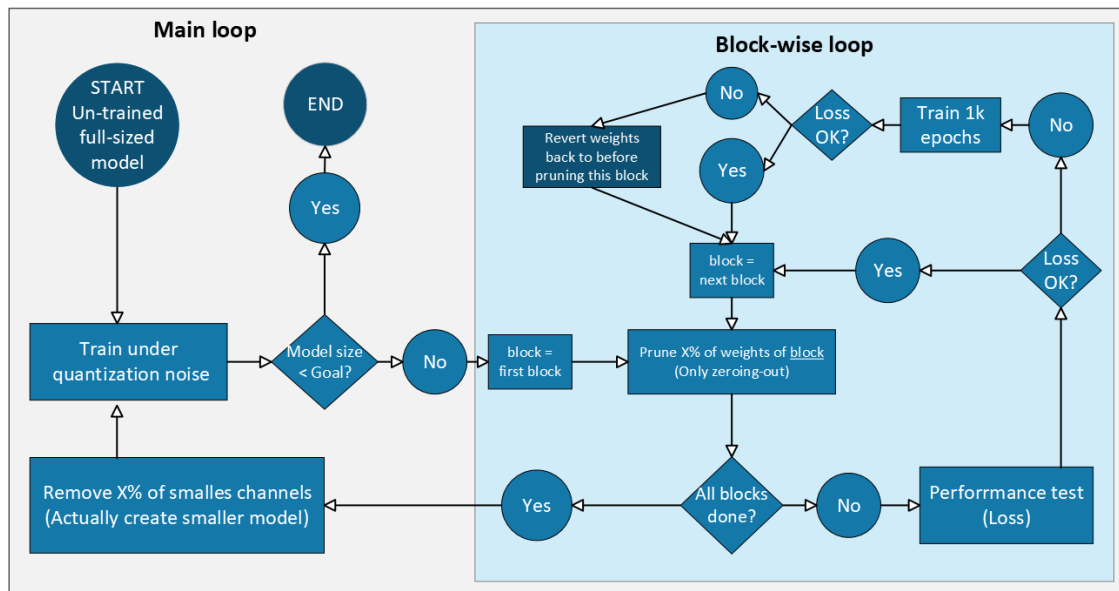
Based on the literature and our tests, we created a training-pruning framework, that fulfills all of the criteria discussed above. Our method is called Quantization-aware multi-stage pruning (QAMP) and its contributions are:

1. A multi-stage pruning algorithm, combining structured and unstructured pruning
2. A straight-forward way to prune DWC-layers
3. A quantization-aware training-pruning pipeline for models containing DWC layers

Despite of extensive research, we were not able to find a pruning algorithm that would fulfill all 3 criterion set in the beginning of chapter 4. QAMP fulfills all of them and hence, is expected to deliver superior results in pruning the DeepRx model compared to other pruning algorithms. QAMP is inspired by Luo et al. [31], Zhou et al. [49], Hertz and John et al. [56] and Jacob et al. [44] and it combines a novelty "multi-stage" pruning algorithm with a quantization-aware training scheme. The multi-stage pruning first uses unstructured pruning to induce sparsity layer-by-layer and then a channel pruning algorithm is applied to the whole model. QAMP has been designed specifically to be used for the DeepRx network that was introduced in chapter 2 and it is further explained in chapter 5.

## 5 QUANTIZATION-AWARE MULTI-STAGE PRUNING

QAMP is based on the idea that a neural network can not only learn to fulfill its purpose in a classification task etc., but it can also learn to operate in certain conditions. This idea is used in our method to teach the model to work with quantized weights and activations and also to work with a pruned structure. The process utilizes quantization-aware training introduced by Jacob et al. in [44] and magnitude pruning introduced already in 1991 by Hertz and John et al. [56]. The magnitude pruning of Hertz and John et al. has also been used on depthwise-separable convolutions by Hu et al. in [33]. The "multi-stage pruning" is inspired by the incremental quantization process introduced Zhou et al. in [49] but instead of quantization, the idea of it is applied to pruning. The hypothesis is that the process removes the maximum number of channels from the model, while maintaining the performance of the quantized model as high as possible. The process is illustrated in figure 5.1.



**Figure 5.1.** Illustration of quantization-aware multi-stage pruning.

Like depicted in Figure 5.1 the process consists of 2 loops: The main loop and the layer-wise loop. The process starts with training the model to convergence under quantization noise, like proposed by Jacob et al. in [44], which is further elaborated later. After the model is trained the process moves to the layer-wise loop.

## 5.1 Block-wise loop

Like shown in Figure 5.1 and Algorithm 2 the block-wise loop consists of iteratively simulating pruning and testing the performance of the model. The simulated pruning starts at the top of the network, pruning the first residual block (line 13 of Algorithm 2) and then testing the performance of the network with a small batch of input data. (Line 14 of Algorithm 2) If the performance is greatly affected by the simulated pruning of channels, the model is fine-tuned before continuing to successive blocks. Hence, the name "multi-stage": the process moves on in multiple stages. Like in the CLIP-Q method of F. Tung et al. in [54] in the block-wise loop the pruned weights are just zeroed-out, hence the term "simulated" pruning is used. The idea of calculating the performance of the model with a mini batch and to use this information to make pruning decisions is based on Luo et al. work in [31].

---

### Algorithm 2: Quantization-Aware multi-stage pruning

---

```

1 Function Main_loop( $V$ ):
2    $V :=$  Trained network;
3    $L :=$  Number of layers in  $V$ ;
4   while  $V.size > Target$  do
5      $Layerwise\_loop(v)$ ;
6      $Prune(V)$ ;
7      $Train\_with\_QAT(V)$ ;
8   end
9 Function Blockwise_loop( $V$ ):
10   $L :=$  Number_of_blocks_in_ $V$ ;
11  while  $i < L$  do
12     $v_i = Simulated\_prune(Score(v_i))$ ;
13     $loss = Test\_loss(V)$ ;
14    if  $loss > threshold$  then
15       $train\_for\_1k\_epochs(V)$ ;
16       $loss = Test\_loss(V)$ ;
17      if  $loss > threshold$  then
18         $dont\_prune(v_i)$ 
19      end
20    else
21       $i += 1$ ;
22      continue;
23    end
24  end
25  return  $V$ ;

```

---

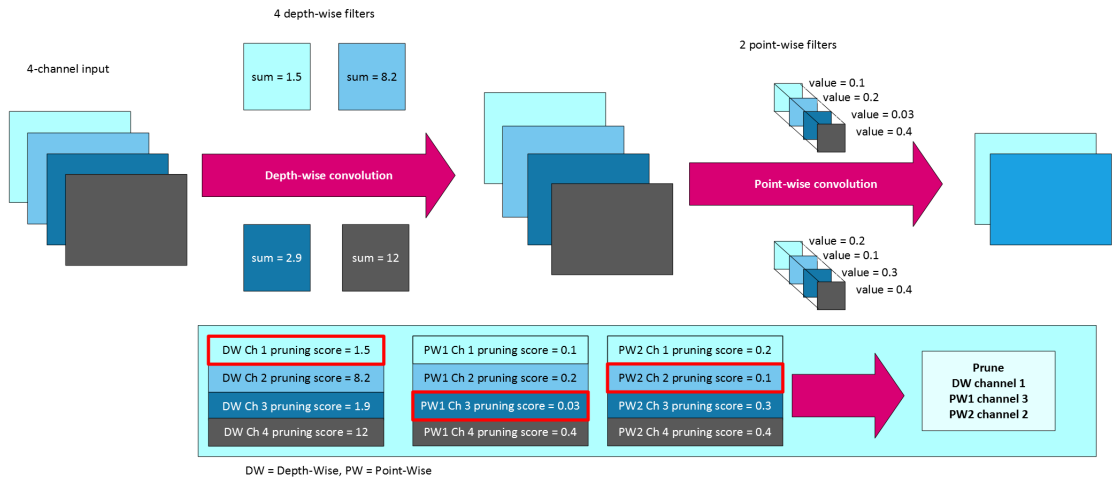
One important feature of the block-wise loop is that it can determine the ideal channel count for each residual block. In Algorithm 2 this is depicted on line 15 and the if-statement on line 17 and the resulting function "*dont\_prune(v)*" on line 18. If after the loss is above the threshold after a block is processed with simulated pruning, the model is fine-tuned for one thousand epochs. If the fine-tuning doesn't help and the performance doesn't increase, the layer is left un-pruned. This is also taken later into account by the real pruning taking place in the main-loop. The threshold on line 17 is a function of the moving mean of the loss of the model during the previous training of the model.

## 5.2 Main loop

Once all the layers of the model have been processed in the block-wise loop, the process moves back to the main loop. There the model is actually pruned (Line 6 of Algorithm 2), this time the connections are not just zeroed-out, they are actually removed. In the main loop the whole model is pruned at once without testing the performance in between layers. The looping of main loop and layer-wise loop is continued until the model size reaches the desired goal. After the pruning process the model is quantized and validated to see how its quantized accuracy is affected. By utilizing the multi-stage block-wise loop, the ideal amount of connections is pruned on each block.

Both loops use magnitude pruning which was originally introduced by Hertz and John et al. but a version of it that supports DWC layers was created by Hu et al. in [33]. Our version of the pruning method works similar to the one of Hu et al., where the sum of all weights in a channel is calculated and then used as a pruning score. Like shown in figure 5.2 this is done in our method on each part of the DWC layer separately so that the smallest values of the depth-wise and point-wise convolutions are removed independently from each other. This is where our method differs from the method of Hu et al., since they use the sum of the whole channel and not each convolution separately. Only the amount of removed filters has to equal in both convolutions but their location can be different in both convolutions.

The motivation of the two loops is that during the layer-wise loop, the model can learn to operate with fewer channels, very much like in the quantization-aware training the model learns how to operate under quantization noise. When the pruned channels are actually removed in the main loop, the model has been prepared for the reduced number of channels. Magnitude pruning was selected for both pruning loops because Gale et al. found in [26] that despite its simplicity, it works better in most cases than more complex pruning schemes. The DeepRx contains residual connections and they need to be taken into account during the pruning. We use a strategy where we prune the inputs and outputs of each block in accordance to each other. This allows the residual connection to work as intended and we can still prune on every part of the network. (See figure 3.1 in chapter



**Figure 5.2.** The depth-wise separable convolutions are pruned based on the sum of values in each channel separately for the depth-wise and point-wise convolution.

3) Only the first layer and the last layer of the network are left un-pruned. Additionally, we found that the pruning process of the DeepRx can be accelerated by employing a "pruning schedule". In the beginning of pruning the network is over-parametrized, but as pruning progresses, the model becomes more sensitive to the loss of channels. Hence, it makes sense to start off with a larger pruning ratio and then lowering it as pruning progresses.

The working principle of the target hardware of the DeepRx creates some restrictions for the architecture of the model. In order to work efficiently, the channel count in each layer of the model needs to be divisible by 16. This obviously needs to be taken into account during pruning of the model. The naive approach would be to just force every instance of pruning to happen in increments of 16, but we found that there is a better way. First, we prune the network normally in order for the block-wise loop to determine a suitable pruning ratio for each block. After the pruning is completed, we run one more pruning iteration, where the channel count is adjusted so that every layer has a channel count that is divisible by 16. This needs to happen so that the performance of the model is not degraded and hence, in some cases we have to "bring back" channels that have been pruned in previous pruning iterations. For this we can take weight values from previous pruning iterations to get the most benefit from the additional capacity. The target hardware is introduced more in depth in section 2.4.

### 5.3 Quantization-awareness

The process is made "quantization-aware" by training the model under quantization noise in the main loop. The quantization-aware training (QAT) is developed by Jacob et al. in [44] and it simulates the effect of fixed-point scalar quantization during training. The idea is that the weights are divided into evenly spaced bins via function 5.1. To put it more precise, the output of the function is a set of values that are evenly spaced by a scale factor  $s$  and shifted by a bias value like shown in Equation 5.1.

$$\begin{aligned}
 x &= \mathbf{min}(\mathbf{max}(x, a), b), \\
 s &= (b - a)/(2^N - 1), \\
 q &= \mathbf{rint}((x - a)/s) * s + a,
 \end{aligned}
 \tag{5.1}$$

where  $[a, b]$  is the min-max range of the function,  $\mathbf{rint}(\cdot)$  is the rounding to the nearest integer and  $q$  is the output value. [44] The output of the function is not an actual 8-bit number, but it simulates the noise that occurs of dividing the weight values into separate bins. For example, in 8-bit quantization the values are separated in 256 bins, represented by 256 integers. During training, the quantized weights are used to calculate the loss of the model, which used to calculate the gradient for the original weights. Fan et al. suggested in [46] similar approaches to quantization-aware training, but they also researched training schemes for more extreme compression, like binary networks. In this thesis we researched only the possibilities of fixed-point scalar quantization, since the target hardware doesn't support any "code books" which are required for more "exotic" quantization schemes.

## 6 EXPERIMENTAL RESULTS

### 6.1 Experiment setup

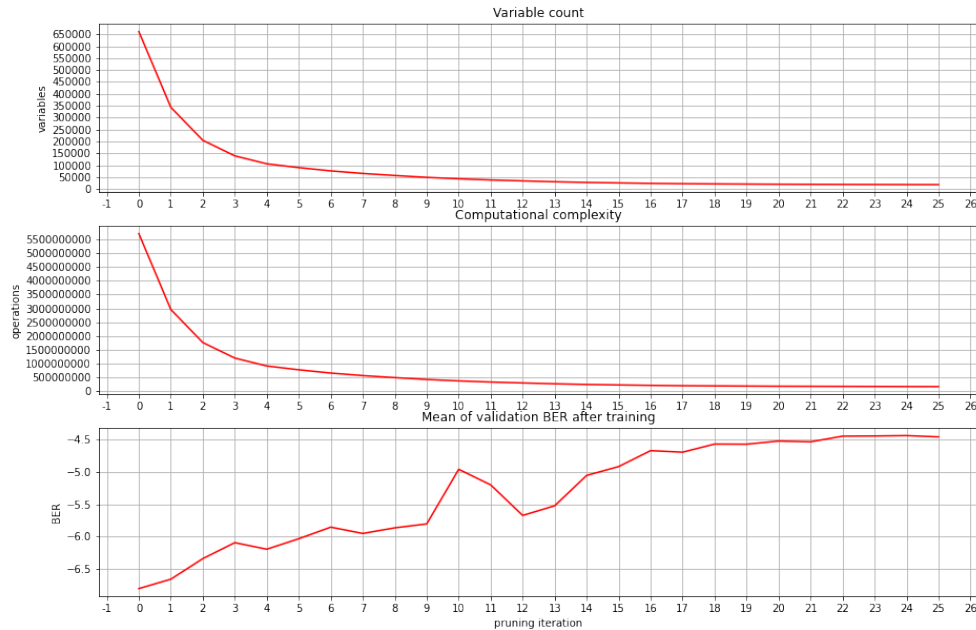
All the tests have been conducted on the DeepRx model introduced in Chapter 2 using Tensorflow ML library (TF) and Python programming language. The data set for training and validation and a basic framework for training the model was provided by Mikko Honkala and Dani Korpi from Nokia Bell Labs. One difference to the model shown in Chapter 2 is that the batch normalization layer in each layer is moved to reside after each convolution. This slight modification didn't have any effect on the performance of the model, but it was necessary in order for the QAT implementation to work at all. Ready-made TF implementations of machine learning tools were crucial for our work since the QAT was adopted straight from the implementation in TF and TF Lite quantization was used for quantizing the models after training.

The actual model that will be deployed on the target hardware is much larger than the one introduced in Chapter 2. Hence, the drop in model size, parameter count etc. shown in this thesis, do not yet tell if DeepRx would actually be implementable on the target hardware. There is also no way to measure the materialized drop in latency or power consumption since the target hardware is not yet implemented nor a simulator for it exists. The target of this section is to present the drop in model complexity and parameter count and show the trade-off between model accuracy and model size. We also test the magnitude pruning of Hu et al. in [33] since it is easy to implement by modifying the QAMP algorithm.

In our training we used a Layer-wise Adaptive Moments optimizer for Batch training optimizer (LAMB) to calculate the gradients. The pruning ratio was changed during the pruning-training process, starting out from 30% and then lowering it on each iteration. The training-pruning process was continued until the performance of the reference receiver couldn't be reached anymore or until the minimum size that still fits the hardware requirements was reached. Different to the QAMP method introduced in the previous section, the pruning in this thesis is done without QAT. This is because we ran into multiple problems with the implementation of QAT, which made it basically impossible to use within the pruning loop. We however we still got excellent results with a quantized DeepRx.

## 6.2 Results

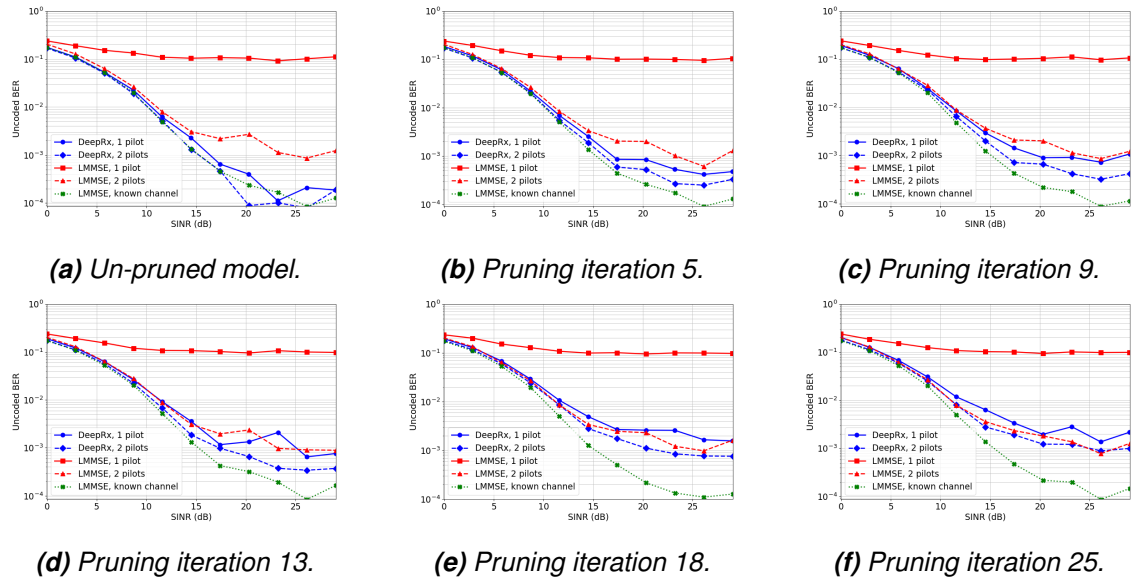
Before pruning, the computational complexity of the model is 5.7 GOPs, in other words one inference with the model requires  $5.7 * 10^9$  operations and it has  $6.6 * 10^5$  variables. The accuracy of the model in the beginning was shown in chapter 2 Figure 2.5. The training-pruning process of the DeepRx is shown in Figure 6.1.



**Figure 6.1.** The top Figure shows the number of variables in the model, the middle Figure the computational complexity and the bottom Figure the mean of BER across a validation set. The vertical axis shows each pruning iteration.

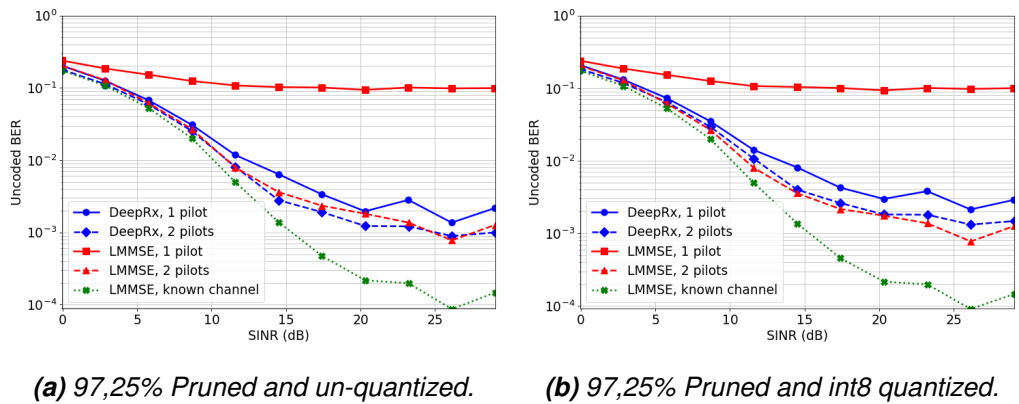
As can be seen from Figure 6.1, the computational complexity of the model is lowered from its original  $5.7 * 10^9$  to under  $0.2 * 10^9$  after the 20th pruning iteration and the amount of variables is lowered from 660 thousand to under 20 thousand! The bottom figure of 6.1 is the mean of BER across a small validation set after the pruned model has been trained. The BER curve shows a rising trend as the model is being pruned which means that the performance of the model degenerates during pruning. It's remarkable that while the reduction in size is dramatic, the performance degenerates only slightly.

From Figure 6.2b can be seen that the first instances of pruning didn't affect the accuracy of the model greatly, even though over 80% of the parameters of the models were pruned at that point. But by comparing Figure 6.2a and 6.4d can clearly be seen how the performance slowly degenerates. We went for a maximum pruning ratio and picked iteration 25 as our final pruned model. This decision was made because after that iteration all the layers of the model had 16 channels, fulfilling the hardware requirements perfectly. Despite



**Figure 6.2.** The radio performance of the DeepRx after some pruning iterations.

being so small, the performance of the model was still on par with the reference model. After pruning iteration 25 the computational complexity of the model is only  $0,157 \times 10^9$  operations, which compared to the starting point of  $5,716 \times 10^9$  operations, is 97,25% smaller!



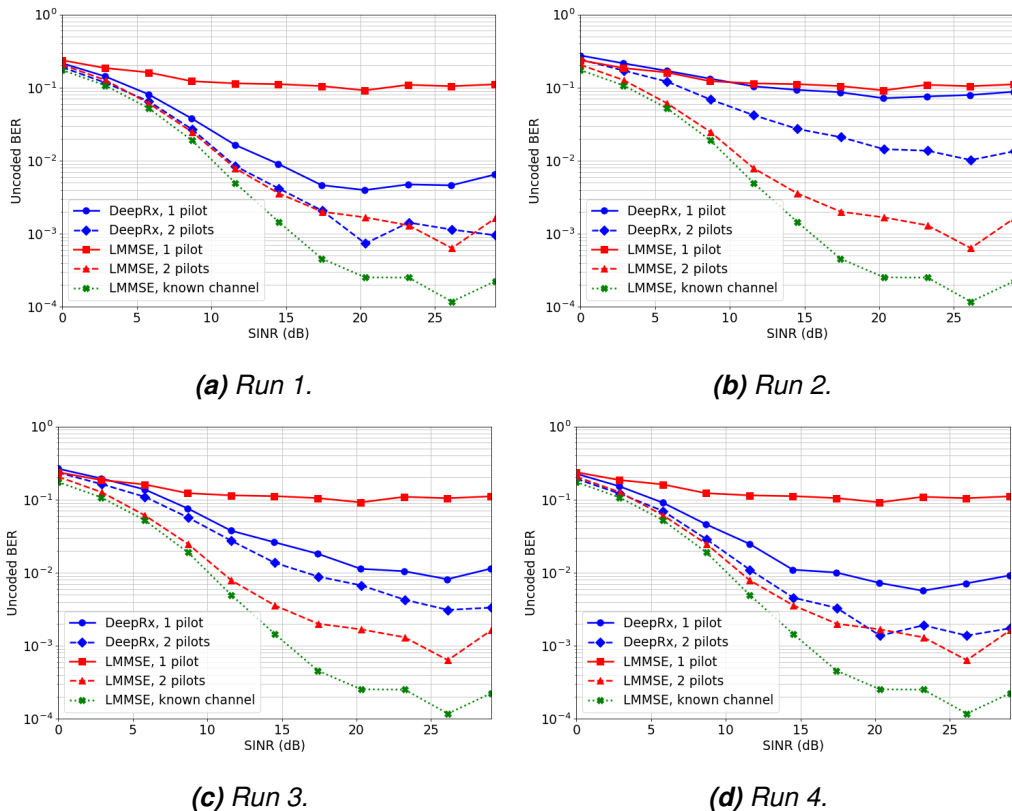
**Figure 6.3.** The radio performance of the pruned DeepRx (a) before quantization and (b) after quantization.

After the pruning process hit the smallest possible architecture that fulfills the hardware requirements, we quantized the model and the result was unexpected. We assumed that the performance of the quantized model would be far worse compared to the un-quantized one, but it seems that quantization affects the accuracy only slightly. The performance of the quantized and pruned model is shown in Figure 6.3b and for comparison the performance of the un-quantized and pruned DeepRx is in Figure 6.3a. Two Tesla V-100

GPUs were available for the training, but even with them the whole pruning and training process took about 25 hours. Hence, there was no time to do a proper hyper parameter optimization. The hyper parameters affect the training of the model greatly and during the thousands of training epochs, the mistakes accumulate. This could have affected the outcome of the pruning to some extent.

### 6.3 Ablation study

In this section we test how much different aspects of the QAMP pipeline affect the result. First, we focus on the benefit of pruning in general. Figure 6.4 shows the radio performance of four models that we trained from scratch to be as small as our pruned model from the previous chapter.

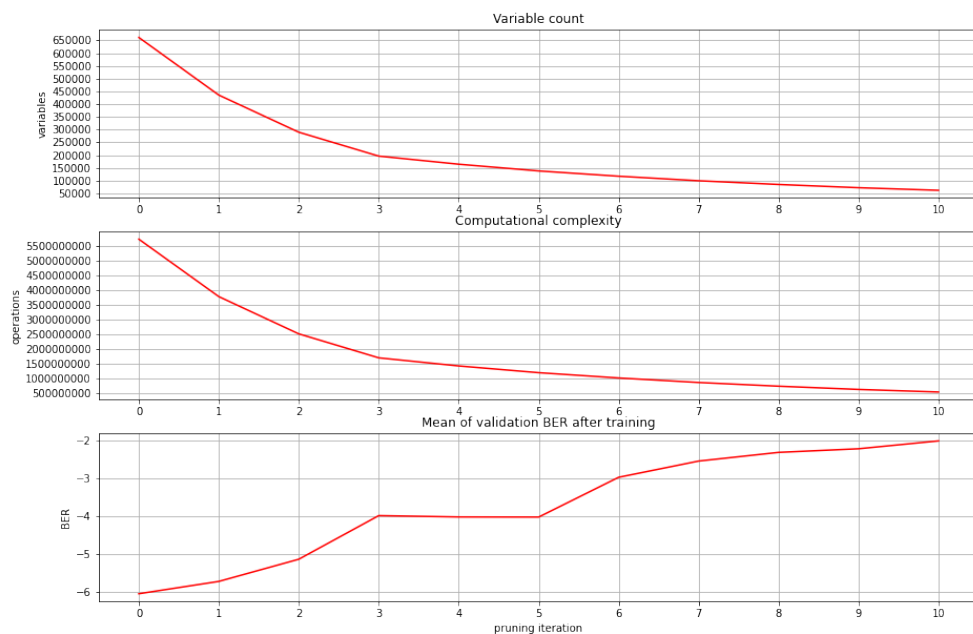


**Figure 6.4.** The radio performance of models trained from scratch with the same architecture as the pruned DeepRx.

As can be seen, the accuracy of run 1 and run 4 are close to our pruned model. We expected that none of these models would come even close to the radio performance of the pruned model. This indicates that our pruning method can still be improved. On the other hand, these models benefited from the QAMP algorithm since they were created with the same architecture that QAMP produced during the pruning process. One major advantage of pruning compared to training small models from scratch is that pruning consistently yielded good results, whereas the results of training small models from scratch

were pretty much random. All the four runs shown above, were trained with the exact same parameters, but the accuracy of the models vary hugely. In some cases, the model didn't converge at all and in some cases the performance was close to the pruned model. With pruning we didn't see this kind of problems, the pruned models converged always the same way.

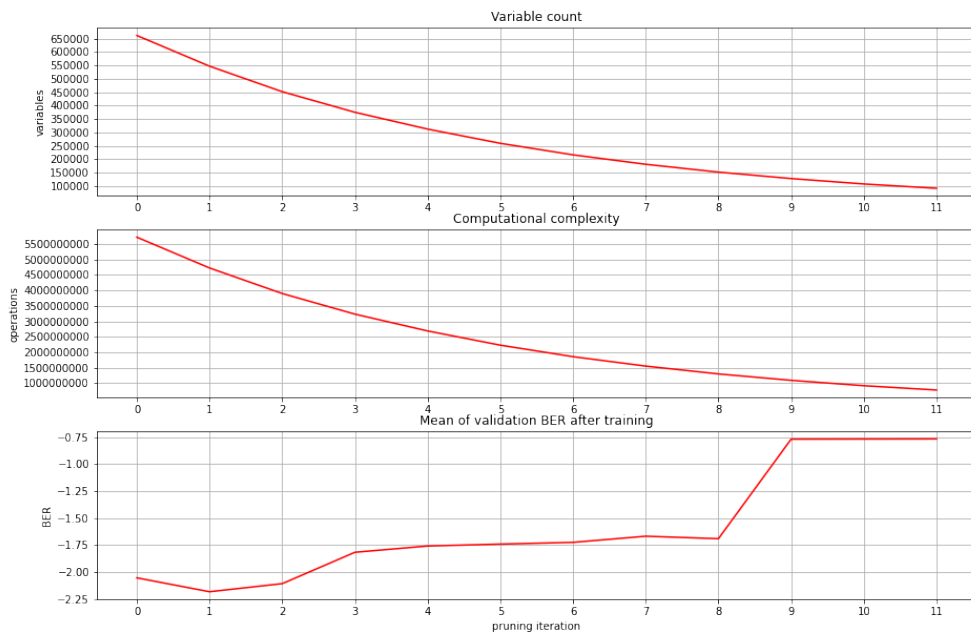
We also wanted to see how much the block-wise loop affects the pruning result. For this purpose, we disabled it, which changed the working principle of the pruning algorithm so that it only prunes a fixed percentage of channels on each layer. The heuristic for the pruning is still the absolute value of the sum of the values in each channel. This makes this version of our pruning algorithm similar to the magnitude pruning for depth-wise convolutions introduced by Hu et al. in [33]. The results are shown in Figure 6.5.



**Figure 6.5.** Pruning process without the block-wise loop.

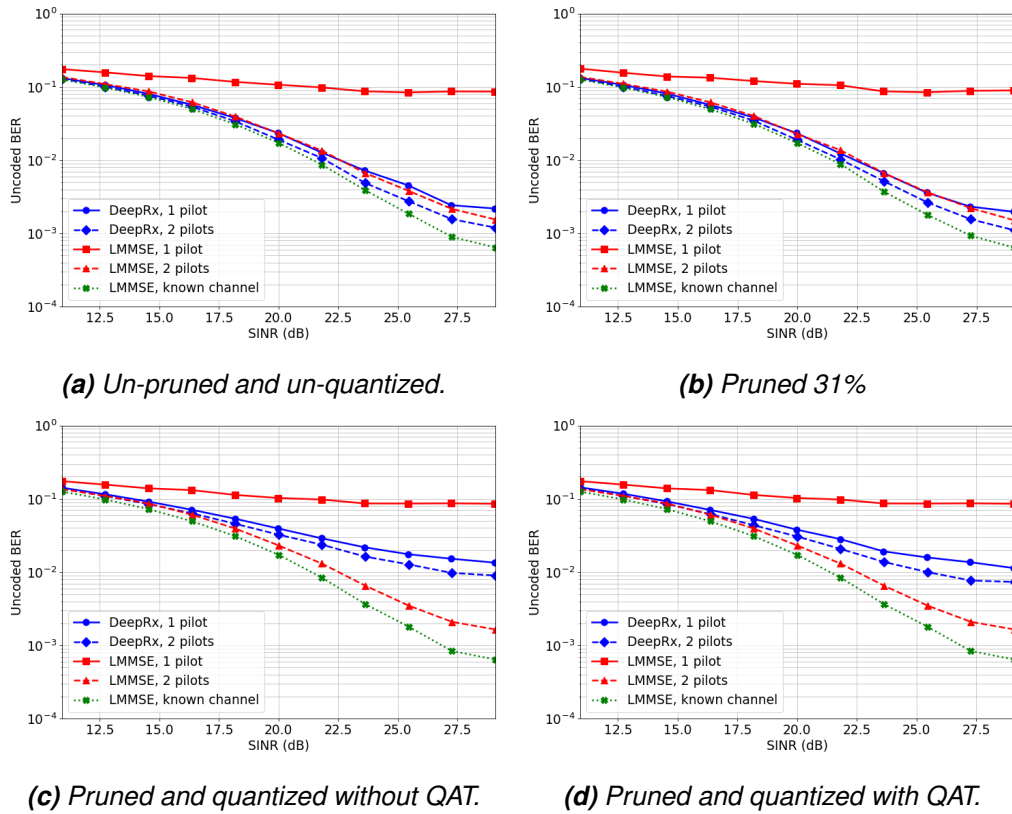
By comparing Figure 6.5 to Figure 6.1, can be seen that disabling the block-wise loop reduces the efficiency of the pruning algorithm significantly. We had to lower the pruning ratio in order to get proper results and even with this remedy, the bit error rate reached unacceptable values already after 5 pruning iterations. At this point the computational complexity of the model was over  $1 * 10^9$  operations larger than the model that QAMP was able to produce. One interesting finding about this experiment is that the pruned DeepRx performed also significantly worse than the small DeepRx trained from scratch. The performance of a radio receiver depends greatly on the input data. The results shown

so far in this thesis were done with 16 QAM (Quadrature Amplitude Modulation) data. This means that each received OFDM symbol can represent 16 different values. In the real world, a higher QAM value is used, which makes the classification of the received OFDM symbols harder. Hence, we also ran tests on 256 QAM data, where each OFDM symbol can represent 256 different values, making the task of the DeepRx much harder. This can be seen from Figure 6.6 and Figure 6.7a, where the BER of the un-pruned DeepRx is much worse compared to the 16 QAM test.



**Figure 6.6.** Pruning test using 256 QAM data. The top Figure shows the number of variables in the model, the middle Figure the computational complexity and the bottom Figure the mean of BER across a validation set. The vertical axis shows each pruning iteration.

As can be seen from Figure 6.6 the DeepRx is a lot harder to prune if 256 QAM data is used. Already after 3 pruning iterations the accuracy of the model is far worse than the reference receiver. Initially we tried to use the same pruning ratio as in the 16 QAM tests, but for 256 QAM we had to reduce the pruning ratio to a fixed value of 10% per pruning iteration. The smallest model that could beat the reference receiver with 256 QAM data had still a computational complexity of  $3,9 \times 10^9$  operations, which means that the model was pruned 31%. Since pruning iteration 2 and 3 have a huge difference in accuracy, we expect that with careful hyper parameter optimization the pruning result with 256 QAM data can still be improved. For the purposes of this test however this performance suffices. Figure 6.7 shows the accuracy of the DeepRx after pruning and quantization with 256 QAM data.



**Figure 6.7.** The accuracy of the DeepRx before pruning (a), after pruning (b), after quantization without QAT fine tuning (c) and with QAT fine tuning (d).

If Figures 6.7b and 6.7c are compared to Figures 6.3a and 6.3b can be seen that the 256 QAM data doesn't only affect pruning, it also affects quantization as well. The drop in accuracy after quantization is significantly greater if 256 QAM data is used. While with 16 QAM data using QAT wasn't even necessary, with 256 QAM data it's crucial! In this test the QAT didn't produce as good results as we hoped, but we believe that there is room for improvement. Creating a QAT implementation that works inside of the QAMP loop might remedy the issue. Based on this test we conclude that a well-working version of QAT will play a major role in the model optimization for the DeepRx. Also, it seems that the classification of 16 QAM OFDM symbols is a relatively easy task for a CNN, but as the QAM value is increased, the task becomes significantly harder.

## 7 CONCLUSIONS

In this thesis we gave an introduction to CNNs and their applications in radio receivers, discussed the optimization of CNNs and gave an overview on existing research on quantization and pruning of CNNs. Through testing and literature research, we concluded that to compress the DeepRx, the best way is to use channel pruning and to utilize quantization. Based on these findings we built our own CNN optimization method, QAMP, and tested its ability to compress the DeepRx. The hardest part in building a working version of QAMP was the architecture of the DeepRx: residual connections and depth-wise separable convolutions are among the hardest CNN structures to prune.

QAMP is a quantization-aware pruning method that utilizes a multi-stage pruning loop that can determine an optimal pruning rate for each residual block of the model. We achieved a tremendous pruning ratio of 97,25% and still managed to quantize the model with very little loss in accuracy! However, we also managed to train an equally small model from scratch to almost the same precision as our pruned model. We also found out that the performance of QAMP is severely reduced if 256 QAM data is used. These findings indicate that there is still room for improvement in QAMP. We also lack a state-of-the-art reference point for comparing the effectiveness of QAMP. This is because pruning algorithms need to be designed to be model-specific. Hence, we would have to implement competing methods ourselves, for which there is no time. We however compared QAMP method to plain magnitude pruning, which QAMP over-performed significantly. One problem in judging the success of our research is that there is no way to test the actual implementation of the DeepRx yet and the model of this thesis is still a proof-of-concept version of the actual one. The results however are encouraging and show that with the help of QAMP, it could be possible to productize this kind of receiver. Pruning in general seems to be a great way to train small models in a predictable way.

We discovered multiple interesting future work items: Refining our implementation of the QAMP algorithm, Pruning and quantizing a bigger version of the DeepRx network and utilizing knowledge distillation to enhance our pruning strategy. Knowledge distillation could for example come into play when the "normal" pruning has been continued to the point that the performance of the model starts to fall.

## REFERENCES

- [1] Lecun, Y. and Denker John Solla, S. Optimal Brain Damage. Vol. 2. Jan. 1989, 598–605.
- [2] MAYORAZ, E. N. On the power of democratic networks. eng. *SIAM journal on discrete mathematics* 9.2 (1996), 258–268. ISSN: 0895-4801.
- [3] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K. and Lang, K. J. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.3 (1989), 328–339.
- [4] Nowlan Steven J. Hinton, G. E. Simplifying Neural Networks by Soft Weight-Sharing. *Neural Computation* 4.4 (1992), 473–493. DOI: 10.1162/neco.1992.4.4.473. eprint: <https://doi.org/10.1162/neco.1992.4.4.473>. URL: <https://doi.org/10.1162/neco.1992.4.4.473>.
- [5] Rumelhart, D., Hinton, G. E. and Williams, R. J. Learning representations by back-propagating errors. *Nature* 323 (1986), 533–536.
- [6] Yosinski, J., Clune, J., Nguyen, A., Fuchs, T. and Lipson, H. *Understanding Neural Networks Through Deep Visualization*. 2015. arXiv: 1506.06579 [cs.CV].
- [7] Han, S., Mao, H. and Dally, W. J. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. 2016. arXiv: 1510.00149 [cs.CV].
- [8] Yu, F. and Koltun, V. *Multi-Scale Context Aggregation by Dilated Convolutions*. 2016. arXiv: 1511.07122 [cs.CV].
- [9] Cedex, S. A. 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Study on channel model for frequencies from 0.5 to 100 GHz (3GPP TR 38.901 version 16.0.0 Release 16). (2019).
- [10] Blalock, D., Ortiz, J. J. G., Frankle, J. and Gutttag, J. *What is the State of Neural Network Pruning?* 2020. arXiv: 2003.03033 [cs.LG].
- [11] Sifre, L. and Mallat, S. *Rigid-Motion Scattering for Texture Classification*. 2014. arXiv: 1403.1687 [cs.CV].
- [12] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV].
- [13] Zhang, K., Cheng, K., Li, J. and Peng, Y. A Channel Pruning Algorithm Based on Depth-Wise Separable Convolution Unit. *IEEE Access* PP (Dec. 2019), 1–1. DOI: 10.1109/ACCESS.2019.2956976.

- [14] Honkala, M., Korpi, D. and Huttunen, J. M. DeepRx: Fully Convolutional Deep Learning Receiver. eng. *IEEE transactions on wireless communications* (2021), 1–1. ISSN: 1536-1276.
- [15] Chang, R. W. Synthesis of Band-Limited Orthogonal Signals for Multichannel Data Transmission. *Bell System Technical Journal* 45.10 (1966), 1775–1796. DOI: 10.1002/j.1538-7305.1966.tb02435.x. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/j.1538-7305.1966.tb02435.x>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1966.tb02435.x>.
- [16] Zou, Q., Tarighat, A. and Sayed, A. Joint compensation of IQ imbalance and phase noise in OFDM wireless systems. eng. *IEEE transactions on communications* 57.2 (2009), 404–414. ISSN: 0090-6778.
- [17] Union, I. T. *Recommendation ITU-R P.372-14 08/2019 Radio noise*. 2019.
- [18] Zhao, Y. and Haggman, S. .-. Sensitivity to Doppler shift and carrier frequency errors in OFDM systems-the consequences and solutions. *Proceedings of Vehicular Technology Conference - VTC*. Vol. 3. 1996, 1564–1568 vol.3. DOI: 10.1109/VETEC.1996.504021.
- [19] Cox, C. *An Introduction to LTE : LTE, LTE-Advanced, SAE, VoLTE and 4G Mobile Communications*. John Wiley and Sons, Incorporated, 2014. URL: <https://ebookcentral.proquest.com/lib/tampere/detail.action?docID=1690922>.
- [20] Neumann, D., Wiese, T. and Utschick, W. Learning the MMSE Channel Estimator. *IEEE Transactions on Signal Processing* (2018), 1–1. ISSN: 1941-0476. DOI: 10.1109/tsp.2018.2799164. URL: <http://dx.doi.org/10.1109/TSP.2018.2799164>.
- [21] Chang, Z., Wang, Y., Li, H. and Wang, Z. Complex CNN-Based Equalization for Communication Signal. *2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP)*. 2019, 513–517. DOI: 10.1109/SIPROCESS.2019.8868708.
- [22] Korpi, D., Honkala, M., Huttunen, J. M. J. and Starck, V. *DeepRx MIMO: Convolutional MIMO Detection with Learned Multiplicative Transformations*. 2020. arXiv: 2010.16283 [eess.SP].
- [23] He, K., Zhang, X., Ren, S. and Sun, J. *Identity Mappings in Deep Residual Networks*. 2016. arXiv: 1603.05027 [cs.CV].
- [24] Baldwin, A. *Internal discussion with Andrew Baldwin*. Nokia Bell Labs, 2021.
- [25] Paupamah, K., James, S. and Klein, R. *Quantisation and Pruning for Neural Network Compression and Regularisation*. 2020. arXiv: 2001.04850 [cs.LG].
- [26] Gale, T., Elsen, E. and Hooker, S. *The State of Sparsity in Deep Neural Networks*. 2019. arXiv: 1902.09574 [cs.LG].

- [27] Wang, Y., Zhang, X., Xie, L., Zhou, J., Su, H., Zhang, B. and Hu, X. *Pruning from Scratch*. 2019. arXiv: 1909.12579 [cs.CV].
- [28] Yang, Q., Wen, W., Wang, Z., Chen, Y. and Li, H. *Integral Pruning on Activations and Weights for Efficient Neural Networks*. 2019. URL: <https://openreview.net/forum?id=HyevnscqtQ>.
- [29] Lee, N., Ajanthan, T. and Torr, P. SNIP: Single-shot network pruning based on connection sensitivity. *International Conference on Learning Representations*. 2019. URL: <https://openreview.net/forum?id=B1VZqjAcYX>.
- [30] Srivastava, R. K., Greff, K. and Schmidhuber, J. *Training Very Deep Networks*. 2015. arXiv: 1507.06228 [cs.LG].
- [31] Luo, J.-H. and Wu, J. *Neural Network Pruning with Residual-Connections and Limited-Data*. 2020. arXiv: 1911.08114 [cs.CV].
- [32] He, K., Zhang, X., Ren, S. and Sun, J. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].
- [33] Hu, Y., Sun, S., Li, J., Wang, X. and Gu, Q. *A novel channel pruning method for deep neural network compression*. 2018. arXiv: 1805.11394 [cs.CV].
- [34] Tu, C.-H., Lee, J.-H., Chan, Y.-M. and Chen, C.-S. Pruning Depthwise Separable Convolutions for Extra Efficiency Gain of Lightweight Models. *International Conference on Learning Representations 2020* (2019).
- [35] Hu, H., Peng, R., Tai, Y.-W. and Tang, C.-K. *Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures*. 2016. arXiv: 1607.03250 [cs.NE].
- [36] Luo, J.-H., Wu, J. and Lin, W. *ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression*. 2017. arXiv: 1707.06342 [cs.CV].
- [37] Zhang, K., Cheng, K., Li, J. and Peng, Y. A Channel Pruning Algorithm Based on Depth-Wise Separable Convolution Unit. *IEEE Access PP* (Dec. 2019), 1–1. DOI: 10.1109/ACCESS.2019.2956976.
- [38] Tung, F. and Mori, G. *Similarity-Preserving Knowledge Distillation*. 2019. arXiv: 1907.09682 [cs.CV].
- [39] Kim, J., Bhalgat, Y., Lee, J., Patel, C. and Kwak, N. *QKD: Quantization-aware Knowledge Distillation*. 2019. arXiv: 1911.12491 [cs.CV].
- [40] Oguntola, I., Olubeko, S. and Sweeney, C. *SlimNets: An Exploration of Deep Model Compression and Acceleration*. 2018. arXiv: 1808.00496 [cs.LG].
- [41] *Imagenet Dataset*. URL: <http://www.image-net.org/>.
- [42] Wang, T., Wang, K., Cai, H., Lin, J., Liu, Z., Wang, H., Lin, Y. and Han, S. APQ: Joint Search for Network Architecture, Pruning and Quantization Policy. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

- [43] Abotaleb, A. M., Elwakil, A. T. and Hadhoud, M. Hybrid Genetic Based Algorithm for CNN Ultra Compression. *2019 31st International Conference on Microelectronics (ICM)*. 2019, 199–202. DOI: 10.1109/ICM48031.2019.9021521.
- [44] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H. and Kalenichenko, D. *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*. 2017. arXiv: 1712.05877 [cs.LG].
- [45] Qin, H., Gong, R., Liu, X., Bai, X., Song, J. and Sebe, N. *Binary neural networks: A survey*. Sept. 2020. DOI: 10.1016/j.patcog.2020.107281. URL: <http://dx.doi.org/10.1016/j.patcog.2020.107281>.
- [46] Fan, A., Stock, P., Graham, B., Grave, E., Gribonval, R., Jegou, H. and Joulin, A. *Training with Quantization Noise for Extreme Model Compression*. 2020. arXiv: 2004.07320 [cs.LG].
- [47] Lin, D. D., Talathi, S. S. and Annapureddy, V. S. *Fixed Point Quantization of Deep Convolutional Networks*. 2016. arXiv: 1511.06393 [cs.LG].
- [48] Ge, T., He, K., Ke, Q. and Sun, J. Optimized Product Quantization. *IEEE Transactions on pattern analysis and machine intelligence, vol. 36* (2014). URL: <https://www.microsoft.com/en-us/research/wp-content/uploads/2013/11/pami13opq.pdf>.
- [49] Zhou, A., Yao, A., Guo, Y., Xu, L. and Chen, Y. *Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights*. 2017. arXiv: 1702.03044 [cs.CV].
- [50] *Tensroflow Quantization-Aware Training*. URL: [https://www.tensorflow.org/model\\_optimization/guide/quantization/training](https://www.tensorflow.org/model_optimization/guide/quantization/training).
- [51] Ding, Q., Zhang, R., Jiang, Y., Zhai, D. and Li, B. Regularized Training Framework for Combining Pruning and Quantization to Compress Neural Networks. eng. *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*. IEEE, 2019, 1–6. ISBN: 9781728135557.
- [52] Choukroun, Y., Kravchik, E., Yang, F. and Kisilev, P. *Low-bit Quantization of Neural Networks for Efficient Inference*. 2019. arXiv: 1902.06822 [cs.LG].
- [53] Han, S., Pool, J., Tran, J. and Dally, W. Learning both Weights and Connections for Efficient Neural Network. *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama and R. Garnett. Vol. 28. Curran Associates, Inc., 2015, 1135–1143. URL: <https://proceedings.neurips.cc/paper/2015/file/ae0eb3eed39d2bcef4622b2499a05fe6-Paper.pdf>.
- [54] Tung, F. and Mori, G. Deep Neural Network Compression by In-Parallel Pruning-Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.3 (2020), 568–579. DOI: 10.1109/TPAMI.2018.2886192. URL: [https://openaccess.thecvf.com/content\\_cvpr\\_2018/papers/Tung\\_CLIP-Q\\_Deep\\_Network\\_CVPR\\_2018\\_paper.pdf](https://openaccess.thecvf.com/content_cvpr_2018/papers/Tung_CLIP-Q_Deep_Network_CVPR_2018_paper.pdf).

- [55] Baalen, M. van, Louizos, C., Nagel, M., Amjad, R. A., Wang, Y., Blankevoort, T. and Welling, M. *Bayesian Bits: Unifying Quantization and Pruning*. 2020. arXiv: 2005.07093 [cs.LG].
- [56] Hertz, J., John, Krough, Flisberg, A., Palmer and G., R. *Introduction To The Theory Of Neural Computation*. Vol. 44. Dec. 1991. DOI: 10.1063/1.2810360.