Otto-von-Guericke-University Magdeburg

Faculty of Computer Science

Department of Databases and Software Engineering

Multi-dimensional optimization of neural network architectures for image processing applications

Master Thesis

Author:

Celine Petrosian Masihi

Examiner and Supervisor:

Prof. Dr. rer. nat. habil. Gunter Saake

2nd Examiner:

Dr. Nadereh Hatami

Supervisor:

MSc. Gabriel Campero Durand

Magdeburg, 22.05.2021

Celine Petrosian
Multi-dimensional optimization of neural network architectures for image processing applications
Master Thesis, Otto-von-Guericke-University Magdeburg, 2021.

Abstract

In recent years, Convolutional Neural Network (CNN) models have emerged as the state-of-the-art approach for supporting applications of machine intelligence in the computer vision domain, including image classification, video annotations and semantic segmentation. Within these kinds of models, there has been a trend of performance breakthroughs being highly related to model complexity, with new models having close to a billion trainable parameters. Consequently, several challenges emerge when trying to deploy such large models for real-time inference on everyday computer devices, which might exhibit limited memory to processors with a low number of floating point operations per second, or finally have constraints on energy usage.

To overcome these challenges, it becomes fundamental to optimize these networks before deployment on edge or embedded devices. Accordingly, in recent years there is a growing interest in CNN model compression and acceleration techniques, which reduce model size and operations required, while trying to not compromise the model accuracy. Existing approaches can be roughly be categorized into: compact network design, data quantization, pruning, low-rank factorization and knowledge distillation. Although each compression technique is designed to provide its best performance, sometimes this comes with assumptions, special software or hardware requirements. Alternatively, for generic approaches, it can be the case that they take a one dimensional perspective and do not contribute to other possible improvements. Compressing models while using more than one approach is termed as joint-way compression, and this has become an emerging area of study, attempting to exploit higher opportunities for model compression.

In this research, we specifically study two common and well investigated approaches of pruning and quantization. We specifically study the compression of the pruned model by post-training quantization of weights and activations, through an evaluated generic stage-wise methodology. With our approach we investigate the effect of hyper-parameters at each stage, on the final performance, and we evaluate the existence of any correlation between them. For our study we specifically adopt two datasets, Cifar10 and ImageNet, and a series of models based on ResNet and MobileNet architectures. Through them, we examine and compare the single and joint-compression techniques based on the compression rate and accuracy.

During the experiments, we observed that in models with small enough weight ranges, after applying Cross-Layer range Equalization (CLE) or after quantization step, the magnitude of some filters turn to zero which enabled us, with our ZP (Zero Pruning without fine-tuning) method, to prune them. We also observed that changing weight ranges using CLE may also be beneficial for the pruning process. Regarding activation quantization, we found out that the optimized method for finding step sizes always performs better than the baseline max abs one.

We furthermore report that the optimal solution for pruning is not necessarily providing the appropriate model for quantization. In some models, such as those robust towards quantization, the pruning has the most influence on the final performance. On other models, it could be the quantization.

Our results suggest that a layer-wise pruning that chooses the best configuration regarding quantization efficiency should be followed. This can be named as quantization-aware pruning, for which we propose that its automation can be considered as promising future work.

.

Acknowledgements

My profound respect and appreciation to Robert Bosch GmbH for providing me with an invaluable learning experience, relying on my skills and capabilities, and giving me the opportunity to collaborate and work with them on my internship, thesis, and student work.

I would like to express my sincere gratitude to my colleagues at Bosch Renningen research center; my supervisor and mentor Dr. Nadereh Hatami, for her continuous motivation and expert advice. M.Sc. Cecilia De La Parra for her kind assistance and contributions during different stages of this research. As well as other experts who provided me with valuable feedbacks.

I would like to thank the knowledgeable, helpful, and compassionate M.Sc. Gabriel Campero Durand.

I am grateful to Prof. Dr. rer. nat. habil. Gunter Saake for allowing me to conduct my thesis research under his authority and his valuable understanding throughout this thesis.

Finally, my special appreciation goes to my family for their encouragement, and support throughout my studies. It was more difficult to complete my dissertation during the last few months than usual due to sickness, and I would not have been able to complete it without my friends' help. They stand by me and never give up on me. My true friends: Foroogh, Negar, Lazo, Nasim, Faezeh and Alireza.

Statement of Authorship

I hereby declare that I am the sole author of this master thesis and that I have not used any sources other than those listed in the bibliography and identified as references. I further declare that I have not submitted this thesis at any other institution in order to obtain a degree.

Signature

Stuttgart , 22.05.2021

Place, Date

Contents

List of Figures 3

List of Tables 5

Abbreviations 6

1 Introduction 9
- 1.1 Motivation . . . 9
- 1.2 Main Contributions . . . 11
- 1.3 Thesis Structure . . . 12

2 Background 13
- 2.1 Basic Background . . . 13
  - 2.1.1 Introduction to Neural Networks . . . 13
  - 2.1.2 Introduction to Convolutional Neural Networks . . . 16
  - 2.1.3 Benchmarked CNN architectures . . . 19
- 2.2 CNN Compression Techniques . . . 24
  - 2.2.1 Quantization . . . 26
  - 2.2.2 Pruning . . . 31
  - 2.2.3 Combined compression techniques . . . 35
- 2.3 Summary . . . 38

3 Design 41
- 3.1 Research Questions . . . 41
- 3.2 Proposed Approach: Post-training Optimization . . . 41
  - 3.2.1 Pruning . . . 42
  - 3.2.2 Quantization . . . 46
  - 3.2.3 Evaluation Criteria . . . 47
- 3.3 Summary . . . 47

4 Experimental Setup 49
- 4.1 Datasets . . . 49
  - 4.1.1 Canadian Institute For Advanced Research Dataset (CIFAR-10) . . . 49
  - 4.1.2 ImageNet Dataset . . . 49
- 4.2 Models . . . 50
  - 4.2.1 ResNet . . . 50
  - 4.2.2 MobileNet . . . 51
- 4.3 Setup . . . 52
  - 4.3.1 Pruning . . . 52
  - 4.3.2 Quantization . . . 53
  - 4.3.3 Cross layer Equalization . . . 54

4.3.4 Bias Correction . . . 55
4.4 Experimental Environment . . . 55
4.5 Summary . . . 55

5 Evaluation and Results - Part1 **57**
5.1 RQ1: Performance of Pruned model . . . 57
5.1.1 Case Study 1: Small Dataset (CIFAR-10) . . . 57
5.1.2 Case Study 2: Large Dataset (ImageNet) . . . 67
5.2 RQ1: Performance of Quantized model . . . 71
5.2.1 Case Study 1: Small Dataset (CIFAR-10) . . . 71
5.2.2 Case Study 2: Large Dataset (ImageNet) . . . 74
5.3 Summary . . . 76

6 Evaluation and Results - Part2 **77**
6.1 RQ2: Combined Compression . . . 77
6.1.1 Case Study 1: Small Dataset (CIFAR-10) . . . 77
6.1.2 Case Study 2: Large Dataset (ImageNet) . . . 82
6.2 Summary . . . 86

7 Conclusion and Future Work **89**
7.1 Conclusion . . . 89
7.2 Future Work . . . 90

Bibliography **91**

List of Figures

1.1 Accuracy results of various CNN models on ImageNet with the number of parameters and Floating-point operation (FLOP)s. [LGWS21] 11

2.1 Perceptron model developed by Rosenblatt et al. in 1957 [Elg20] 14
2.2 Multi-Layer Perceptron (MLP) architecture [Elg20]. Based on task complexity there could be different numbers of hidden layers. 15
2.3 Convolution layer process [Elg20] 17
2.4 The convolutional layer with N filters of size (k,k,3) applied on an input Feature Map (FM) with dimensions (w,h,3) [Gai18] 17
2.5 A simple CNN architecture [ON15] 19
2.6 Max and average pooling with kernel size of 2 and stride=2 19
2.7 AlexNet architecture by [KSH12, Elg20] 20
2.8 VGG(also known as VGG16) network architecture [LCO18] 21
2.9 naive version of inception module [SLJ+15] 21
2.10 Different building blocks of ResNets [HZRS16a]. (a) basic block with identity shortcut, (b) bottleneck block with identity shortcut, (c) bottleneck block with projection shortcut [DRRT+18] [HZRS16a]. 22
2.11 Architectures for ResNet based on ImageNet [HZRS16a]. 22
2.12 (a) Convolution mechanism in CNN , (b) Depthwise Separable CNN [KYWW19]. 23
2.13 MobileNet V2- Bottleneck residual block [KMGB21] 24
2.14 The hierarchy of neural network compression algorithms [DLH+20] 24
2.15 The generic teacher-student framework for knowledge distillation by [GYMT20]. 26
2.16 Symmetric(left) vs. Asymmetric(right) quantization by [GKD+21]. 28
2.17 (a) Deterministic and (b) Stochastic quantization by [KK20]. 28
2.18 Quantization level distributions by [DLH+20] 29
2.19 channel wise weight distribution of first depthwise separable layer in MobileNetV2 by [NBBW19] 30
2.20 Overview of three-stage pruning pipeline by [LW20]. 32
2.21 (a) Unstructured weight pruning [HPTD15], (b) Different types of structured pruning [LMX+20] by [WYH+19]. 33
2.22 pruning criterion: l_p-Norm based vs. relation-based [HLZY19] 34
2.23 Deep compression pipeline proposed by [HMD15] 35
2.24 OPQ pipeline proposed by [HPZ+21] 36
2.25 In-Parallel Pruning-Quantization proposed by [TM18] 36
2.26 Illustration of trainable quantizer (Quantization Interval Learning (QIL)) proposed by [JSL+19] 37
2.27 Multi-stage automated design by [HCZ+19] 38
2.28 APQ proposed by [WWC+20] 38

3.1 Framework high-level view 42
3.2 Detailed workflow of our approach 43

3.3 Pruning a filter results in removal of its corresponding feature map and related kernels in the next layer [LKD+16] . . . 44
3.4 Flowchart of our pruning workflow . . . 45
3.5 Pruning strategy for residual block by [WZZ+18], we skip pruning the last layer in each block. . . . 46
3.6 Pruning strategy for depthwise separable convolutions [TLCC20]. . . . 46

4.1 Sample images from the CIFAR-10 dataset and their correct classifications [KH+09]. The resolution of the images is 32×32. . . . 50
4.2 Sample images from ImageNet dataset [CRK16]. . . . 51

5.1 ResNet Architecture . . . 59
5.2 Different ResNet-32 pruned models. The blue line represents the Pareto frontier containing the best models in the search space. The numbers above each model show the accuracy degradation threshold parameter. . . . 61
5.3 Heatmap of pruning rates per layer in different experiments of ResNet32 on Cifar-10. Each row is a run of experiment with mentioned pruning rate and accuracy degradation tolerance. Each column represents each prunable layer of the network. The annotation in each cell is the pruning ratio of the corresponding layer. . . . 62
5.4 Different MobileNetV2 pruned models. The blue line represents the Pareto frontier containing the best models in the search space. The numbers above each model show the accuracy degradation threshold parameter. . . . 64
5.5 Heatmap of pruning rates per layer in different MobileNetV2 experiments on Cifar-10. Each row is a run of experiment with mentioned pruning metric and accuracy threshold. Each column represents each prunable layer of the network. The annotation in each cell is the pruning rate of the corresponding layer. . . . 65
5.6 MobileNetV2 layer wise pruning visualization. Bar charts depict the pruning rate for each layer. The line chart indicates the model's intermediate accuracy. . . . 66
5.7 ResNet50 pruning visualization . . . 69
5.8 MobileNetV2 pruning visualization . . . 72
5.9 Output channel ranges of the first depthwise separable layer in MobileNetV2 trained on Cifar-10 before and after applying CLE. . . . 74

6.1 Sensitivity of ResNet32 when quantized to 4/8-bit weight precision; measured for both main(blue and red lines) and pruned models (green and purple lines). . . . 80
6.2 Visualization of MobileNet(Cifar-10) models optimized using pruning and quantization. Y-axis is in log scale and Errors are presented in absolute format. . . . 81
6.3 Performance visualization of ResNet-50 models optimized using pruning and quantization. Errors are presented in absolute format. . . . 83

List of Tables

5.1 Performance of models ResNet32-56 pruned with different set of hyper-parameters. . . . 61
5.2 Effect of CLE on ResNet-32,56 models pruning . . . 63
5.3 Performance of MobileNetV2 pruned with different accuracy degradation thresholds and pruning metrics. . . . 63
5.4 MobileNetV2 pruning performance under channel equalization. . . . 66
5.5 Performance of ResNet-50 pruned with different pruning configurations. . . . 68
5.6 Performance of MobileNetV2 pruned with different pruning configurations on ImageNet dataset. . . . 70
5.7 Quantization results of the models based on Cifar-10 dataset . . . 71
5.8 Top-1 accuracy of quantized ResNet-56 and MobileNetV2 using CLE and Iterative Bias Correction (IBC) methods. . . . 73
5.9 Quantization results of the ResNet50 and MobileNetV2 models based on ImageNet dataset . . . 74
5.10 Top-1 accuracy of the ResNet50 after applying cross layer equalization and bias correction. . . . 74
5.11 Quantization results of the models based on ImageNet dataset . . . 75

6.1 Combined compression of ResNet32 and 56 based on Cifar-10 dataset. Original models are pruned by $\alpha = 0.4$ and $\epsilon = 0.01$. Acc Δ represents the accuracy degradation of the baseline model. . . . 78
6.2 Comparison of the filter counts with zero magnitude in ResNet32. The columns represent the name of the layers which can be pruned. WQ means the weight quantized version of the original model, Pruned+WQ means the quantized version of the already pruned model. . . . 78
6.3 Effect of zero magnitude filters pruning . . . 79
6.4 Performance details of MobileNetV2 (Cifar-10) joint-way compressed models. Error values are in absolute format. . . . 82
6.5 Performance details of ResNet-50 joint-way compressed models. Absolute values of the errors are reported. . . . 84
6.6 Performance details of MobileNetV2 (ImageNet) joint-way compressed models . . . 85

List of Abbreviations

AI Artificial Intelligence

ANN Artificial Neural Network

AQ Activation Quantization

BFT Bias Fine Tuning

BN Batch Normalization

CLE Cross-Layer range Equalization

CNN Convolutional Neural Network

Conv Convolution

CPU Central Processing Units

DNN Deep Neural Networks

FC Fully Connected

FLOP Floating-point operation

FM Feature Map

GM Geometric Median

GPU Graphics Processing Units

IBC Iterative Bias Correction

KD Knowledge Distillation

KL Kullback–Leibler

MAC multiply-and-accumulate

MAS Mean Activation Shift

MF Matrix Factorization

ML Machine Learning

MLP Multi-Layer Perceptron

MSE Mean Squared Error

NAS Neural Architecture Search

NN Neural Network

PTQ Post Training Quantization

QAT Quantization Aware Training

QIL Quantization Interval Learning

ReLU Rectified Linear Unit

SGD Stochastic Gradient Descent

SQNR Signal to Quantization Noise Ratio

SVD Singular Value Decomposition

WQ weight quantization

ZP Zero Pruning

1 Introduction

1.1 Motivation

John McCarthy defined Artificial Intelligence (AI) as the science and engineering of creating artificial systems that emulate human intelligence, in order to achieve specified goals [SCYE17]. As a subfield of AI, Machine Learning (ML) includes algorithms that allow systems to learn from data, without explicit programming [SCYE17]. These learning processes occur as a result of a *training* phase, in which a training dataset is utilized. Artificial Neural Network (ANN) is a special group of ML algorithms, which replicates biological brain logic and get inspired by neurons' computation. In essence, ANNs are formed of models with different architectures that organize artificial neurons into layers. Through the last years, ANNs have been shown to be useful for many tasks in AI, ranging from image recognition to speech-to-text transformation. This is due to the capacity of ANNs to learn compact internal representations from complex, high-dimensional data. Given this capacity, ANNs are specially able to learn from raw data, without difficult feature engineering [Elg20]. Furthermore, in recent years, technologies for ANN have been developed to speed up the training of these models, through the use of Graphics Processing Units (GPU), and to scale-up to learning from very large amounts of data. All these features and developments have lead to a popularization of ANNs, through the field of deep learning.

As Neural Network (NN)s have gained in popularity, bigger ANN models have been proposed, often consisting of a deep number of layers, which enables them to learn more complex tasks [ZG17]. One of the efficient kind of ANNs in the digital image processing domain is called CNN. These methods have been shown to outperform traditional methods in *computer vision* tasks, quickly becoming the state of the art in the field[OCC+19]. In addition to computer vision tasks such as image classification, object detection, and semantic segmentation, CNNs can also be applied to tasks in natural language processing, such as speech recognition, or even to time series data analysis [GSL19, LZH+16]. The trend mentioned previously for bigger ANN models is also observed in models with CNNs: An approach that is commonly used to improve the accuracy of models when dealing with complex tasks or datasets is to increase the number of layers and filters within layers [PBR18]. For example, VGGNet was designed with 16 and 19 layers. Later, more efficient model like ResNet was designed to have minimum 18 or maximum 152 layers [BWD+20]. Designing CNN models with new building blocks resulted in more compact models, for instance MobileNetV1 with 4.2M parameters in comparison to VGG16 with 138M parameters [BWD+20]. Figure 1.1 illustrates this trend for different CNN models on ImageNet. It shows model's accuracy in comparison to its size(number of parameters) and Giga FLOPs. In addition, models also tend to require more training (as is the case of NASNet, where the model though limited in size, is a result of automated machine learning.)

These new characteristics of large CNN models, such as great memory consumption and high computational needs for the training of the models have emerged as challenges, specially regarding hardware requirements [LZH+16, DLH+20]. Conventional processing platforms, mostly consisting of Central Processing Units (CPU), with their limited amount of operations per second, have become less adequate for training CNNs compared to GPUs [DLH+20, SCYE17]. However, to employing already trained CNNs on diverse tasks remains a challenge, because often the hardware available (i.e., so-called edge devices such as smart-phones, smart sensors, wearable devices, robots, drones, and etc) cannot provide sufficient memory or operations per second to employ large CNN models [DLH+20]. In these resource-constrained environments, features like real-time inference acceleration, low energy and memory consumption, become critical [CMGS20, NBBW19].

For real-time inference acceleration, we need architectures and methods that can reduce the number of multiply-and-accumulate (MAC) operations or FLOPs achieved in the layers due to the convolution matrix multiplications that dominate the computation cost [PBR18]. Beside algorithm side optimizations, CNN acceleration can be handled by designing hardware accelerators [LGWS21, DLH+20, KSZQ20].

Energy consumption depends on the computations and memory access of weights, FM or intermediate activation and their configuration(e.g., size, resolution, and number of channels) [YCS17].

Memory consumption is directly affected by model size. The model size for a neural network relates to the number of parameters used, but more specifically it is the amount of memory required to store all the weights of the layers.

In order to serve the three features required for hardware with limited budget (i.e., inference acceleration, efficient energy use and low memory consumption), NN compression, which produces a more compact and optimized model with minor accuracy loss [HPZ+21], has turned into a timely and promising research topic.

There are several categories that can be used to classify current network compression methods: efficient network design, tensor decomposition, data quantization, pruning, and Knowledge Distillation (KD) [CWZZ17, DLH+20, CMGS20]. Quantization enforces weights into reducing the precision of their bit representation. Deploying CNN models on FPGAs and smartphones makes the quantization a necessary step [WWC+20]. Pruning is the other common and well investigated approach, which by introducing a degree of sparsity into the network makes it thinner. Orthogonality property of these two approaches [WWC+20, DDG+19] motivated scientists for further *joint-way* compression researches [DLH+20]. There are different ways of combining these techniques; pruning can be incorporated in the quantization step [TM18] or in a more intuitive way they can be applied sequentially in a stage-wise manner [HMD15].

There has been a variety of techniques provided for each of these compression approaches [GKD+21, XHCZ20] which their performance is highly correlated to the hyper-parameters, algorithm details, and also CNN model related characteristics such as architecture, building blocks, and data distribution. However, to date there is still limited understanding about the impact of mentioned parameters when they are used in combination.

In this thesis, we aim to study the performance of the joint-compression approaches on state-of-the-art CNN models, while using benchmark datasets. We propose a generic

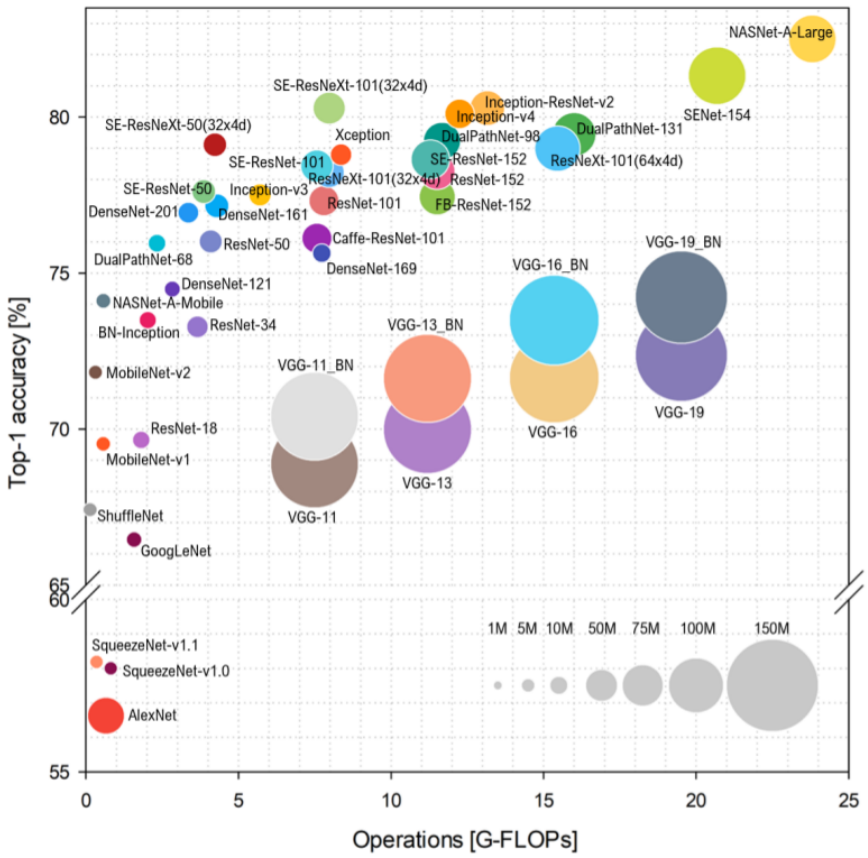


Figure 1.1: Accuracy results of various CNN models on ImageNet with the number of parameters and FLOPs. [LGWS21]

stage-wise framework where the first step is pruning, and the second one is quantization. We investigate the effect of tuning hyper-parameters and method selection for each stage first separately and later jointly.

We have described the scope of our research in this section; the main contributions of this thesis are summarized in the following section.

1.2 Main Contributions

1. We propose a two-stage network compression scheme which consists of two steps: pruning and quantization.

2. In the pruning step of our proposed scheme, we implement an iterative structured filter-based pruning which has, as main hyper-parameters the pruning rate and accuracy degradation threshold. The pruning metric is also configurable.

3. We design and implement a quantization scheme that is also generic and not hardware restricted. It quantizes weights and activations in 8-bit, uniform, and symmetric manners.

4. We implement our design by using models from the ResNet and MobileNet families, and two representative datasets from the computer vision domain, Cifar10 and ImageNet.
5. We investigate the hyper-parameter tuning, configuration selection, and intermediate optimization methods on pre-trained models in each stage separately, by using our selected models and datasets.
6. We also investigate the influence of hyper-parameter and configuration selection in pruning stage on quantization performance.
7. We furthermore present a careful layer-wise analysis of pruning efficiency, to properly adopt quantization and identify its most promising applications. Through this method we identify highly competitive compression schemes for the selected models, and areas of further study.

1.3 Thesis Structure

The rest of the thesis is structured as follows:

- In Chapter 2, we review the basic background about Deep Neural Networks (DNN)s and CNNs, and we explain the popular CNN benchmarks. In a separate section, we go through applicable compression techniques and discuss the research gap which we are specifically investigating.
- Chapter 3 formulates the goals of this thesis. We cover our specific research questions and propose our design, including data pre-processing, classification, and evaluation choices to be made.
- In Chapter 4, we review such essential elements of the experimental setup as the datasets and CNN models that were used, hyper-parameter of the algorithms, programming frameworks and hardware details.
- Chapter 5 is dedicated to first part of the evaluations: performances of compressed models using compression techniques each one applied separately.
- Chapter 6 presents second part of the evaluations: results of the joint-way compressed models and correlations between different parameters.
- Finally, we conclude this thesis and outline some intriguing directions for future research in Chapter 7.

2 Background

Our background knowledge is presented in this chapter. It summarizes the key topics we are exploring in this thesis. Our goal is to provide readers with a solid foundational understanding of our research area in this chapter. It follows the following structure:

- In Section 2.1, we overview the basic concepts of ANNs and then introduce the CNNs. We review few benchmark CNN models including the models we used in the evaluations.
- Section 2.2 presents an overview of the CNN compression approaches. It provides details around chosen approaches that inspired this thesis and the combination of them.
- In Section 2.3, we summarize this chapter.

2.1 Basic Background

Machine learning is a popular branch of artificial intelligence that seeks to solve problems by collecting a dataset and building a model based on such a dataset. Goodfellow et al. distinguish between two general approaches used for machine learning models[GBCB16]: feature engineering and representation learning. Taking image tasks as an example, feature engineering approaches would require to reduce the high-dimensional data contained in an image to a compact series of features that would then be provided to any machine learning model. This creation of features is intended to be appropriate for the task, and this process is called feature engineering. Feature engineering can be time consuming and requires a lot of domain knowledge. On the other hand, representation learning refers to approaches that take as input raw data, such as the image of our example task, and as part of their training create a compact internal representation, hence this is called representation learning. In our work we consider a popular representation learning method: artificial neural networks.

2.1.1 Introduction to Neural Networks

In this section, we first introduce the elementary building block of a NN, which is the *perceptron*. This computational unit is modeled after the biological neurons, the main computational element of the human brain [SCYE17]. A perceptron receives N inputs (signal x_1 through x_N) and multiplies each of them with a corresponding weight w_i. These weights represent the importance of input signals. As illustrated in Figure 2.1, the perceptron takes that weighted sum of the inputs and after adding a fixed term of *bias*

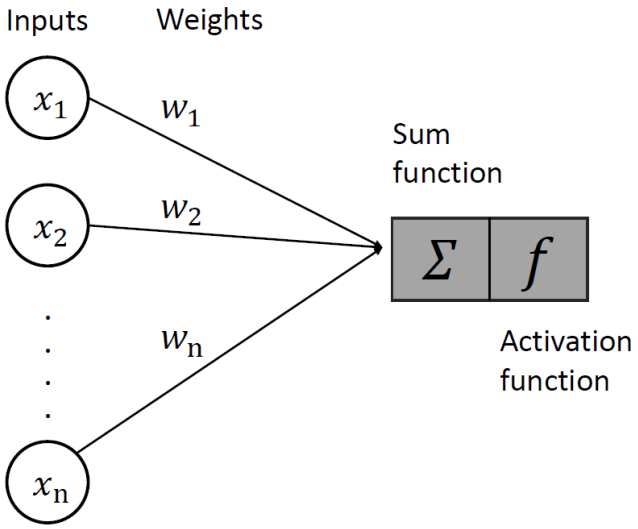


Figure 2.1: Perceptron model developed by Rosenblatt et al. in 1957 [Elg20]

(equation 2.1) passes it through an *activation* function. This activation function acts as a signal transformer, and decides finally whether to activate the perceptron's output based on a threshold.

$$u = \sum_{i=1}^{N} x_i \cdot w_i + b \tag{2.1}$$

Once the input exceeds the threshold, the perceptron is activated (fired) and the output is one, otherwise the output signal becomes zero. Artificial *neurons* are generalized notions of the perceptron, where their output is a non-zero value instead of a binary one. The mathematical formula for an activation function $f(.)$ is given in equation (2.2).

$$f(u) = \begin{cases} 1, & if\ u > \theta \\ 0, & Otherwise \end{cases} \tag{2.2}$$

A perceptron can learn a task of interest by adjusting its weight vector. An individual perceptron is enough for learning simple functions with linear decision boundaries, such as AND or OR functions. However, groups of perceptrons are required to learn more complex functions like XOR. One layer of a NN consists of these perceptrons, and for more complex tasks one network can have multiple layers. These layers can be structured in a *fully-connected* shape and together form the MLP. Fully-connected means that the output of each neuron in the current layer serves as an input to each of the neurons in the following layer. MLPs are also known as feed-forward neural networks. They can be visualized as a finite directed acyclic graph (see Figure 2.2). The main components of a feed-forward NN architecture are as follows:

- *Input layer* Neurons in the input layer use the input dataset as a vector of features and transmit it to the next layer, without making any mathematical calculations.

- *Output layer* It is the last layer that provides the prediction for the model. Depending on the choice of activation function, the output can be represented by real-values (regression) or probabilities (classification). In a classification task, for

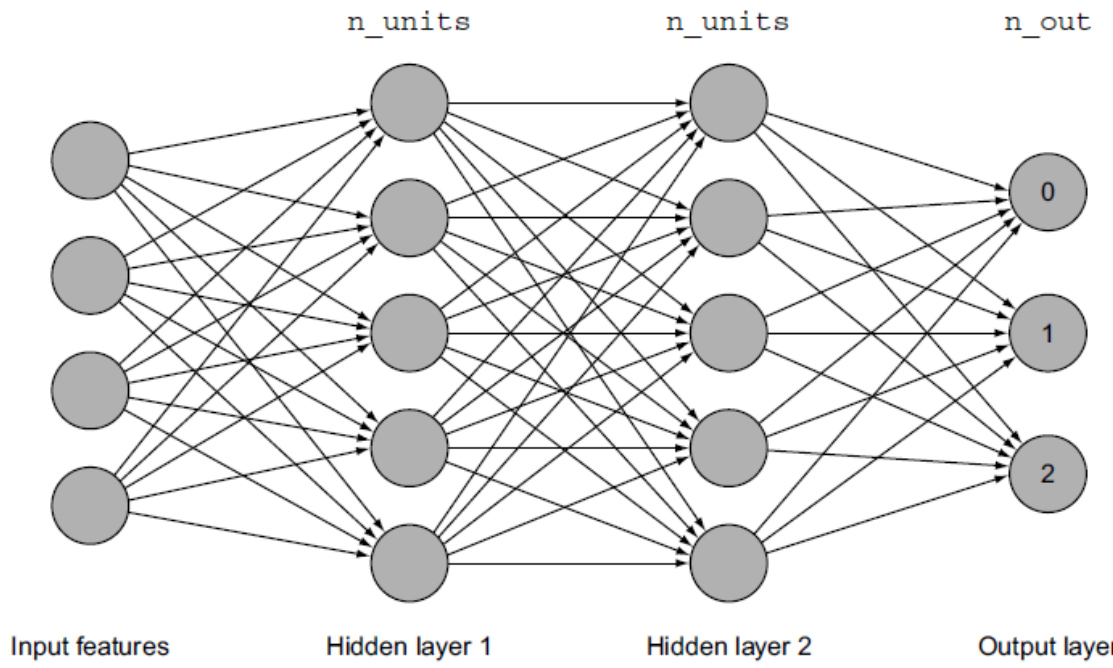


Figure 2.2: MLP architecture [Elg20]. Based on task complexity there could be different numbers of hidden layers.

example, the probabilities from each neuron are used as the likelihood score for each class.

- *Hidden layers* Between an input layer and output layer, there are layers that are called hidden layers. NNs may consist of any number of hidden layers with a variety of neurons in each one. The learning of internal features or internal representations (i.e., representation learning), to assist the learning task, is handled by these layers. The complexity of the learning task determines the number of hidden layers that may be necessary.

At it's simplest form, MLPs can be encoded as matrices. Let's consider two consecutive fully connected layers of $l-1$ and l with m and n neurons. The weight of the connections between them is represented with a $(m \times n)$ matrix of floating-point numbers. The output of neuron j in layer l (y_j^l) can be calculated using equation (2.3):

$$y_j^l = f\left(\sum_{i=1}^{m} w_{i,j}^l \cdot f_i^{l-1} + b_j^l\right) \tag{2.3}$$

where f_i^{l-1} means the output activation of each neuron in layer $l-1$ and $w_{i,j}$ is used to denote the weights between neurons in one layer and neurons in the next layer (an element of the weight matrix).

As the depth of the NN grows (having more hidden layers), the network is able to extract more complex features [SCYE17]. The models that contain two or more hidden layers are generally referred to as DNNs [Elg20]. The number of hidden layers used, and the number of neurons in each, are hyper-parameters that should be considered when designing a DNN. The weights and biases of DNNs are learned in a phase known as *training*. A trained network can provide task-related output using those learned weights.

This phase is called *inference* [SCYE17]. It is not always the case that a greater number of hidden layers results in better performance during the inference stage. This can be caused by a phenomenon called *over-fitting*. In such scenario, the network shows good results on training data because it memorizes the features instead of learning them, therefore it performs poorly during the inference phase, as the model fails to generalize [Elg20].

In sum, MLPs or Artificial Neural Networks, are a basic kind of machine learning model that is based on the structure and functioning of biological neurons, it is furthermore an approach based on representation learning[GBCB16]. ANNs are formed of perceptrons organized into input, output and hidden layers. Each perceptron contains a series of weights that and biases, which coupled with an activation function, determine the kind of signal that the perceptron will produce based on its input. In order to be employed for the task, the weights and biases of a model can be learned through a training stage, and then they can be applied in an inference stage. The data of a model can be represented as matrices. Feed-forward fully-connected neural networks are one of the basic models of this family. For such models, adding hidden layers can be useful, but they can lead to over-fitting. In the next section, we will discuss a special kind of neural network proposed for tasks with spatial relationships in the input data, CNNs. Apart from CNNs, there are many other kinds of ANN models, such as recurrent neural networks (RNNs)[LBE15] that are specially developed to deal with sequential data, and graph neural networks (GNNs)[DJL+20] that are specially developed to deal with non-Euclidean data.

2.1.2 Introduction to Convolutional Neural Networks

The CNN is a kind of DNN that has been applied for many tasks, specifically in vision-related tasks, such as image classification. CNNs are designed to recognize patterns/features contained within a single image input, or more generally a tensor of data. They use fewer parameters than traditional ANNs. One of the first works in modern CNN was done by LeCun et al.[LBBH98], reaching breakthroughs in handwritten digit recognition.

Feature extraction is achieved with *convolutional layers*. They replace a fully connected layer in a conventional ANN. Convolution is a mathematical operation that refers to the step of combining two inputs into a third function. In CNN context, the input image is the first function where the second function, *convolution filter* is operating on the given image. The resulting convolved image is referred to as a *FM*. The convolution process is illustrated in Figure 2.3. The convolutional layer operates on the input image by sliding a filter over it periodically. Each time it also affects a certain spatial portion of the input image, known as *receptive field*. As shown, in this example, the filter has a *kernel* size of (3×3) and through a matrix multiplication over the receptive field, it calculates one element (pixel) of the feature map. This filter is a weight matrix that is learned during the training process of the CNN. Every convolutional layer employs different filters to discover unique patterns from incoming information. Then these FMs are stacked to form the final output of the convolutional layer. Shallow convolutional layers grab low-level patterns (edges and curves) and the next layers based on those FMs extract more complex or high-level features (in the case of a face recognition task, these can include: ear, nose, eyes and etc.)

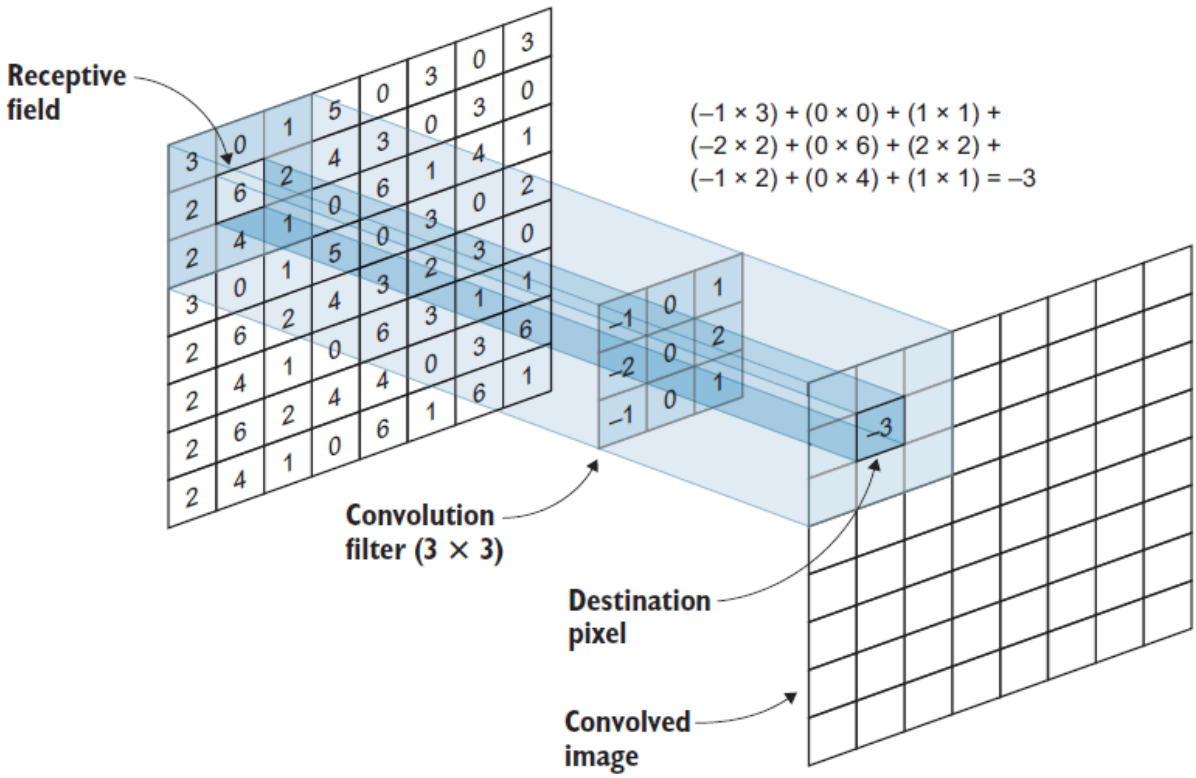


Figure 2.3: Convolution layer process [Elg20]

Figure 2.3 is a two-dimensional representation of the convolution process when in fact images are presented in three dimensions: height, width, and depth, where the depth stands for color channels (RGB). Each filter kernel is built in such a way that it covers the entire dimension of its input feature map (Figure 2.4).

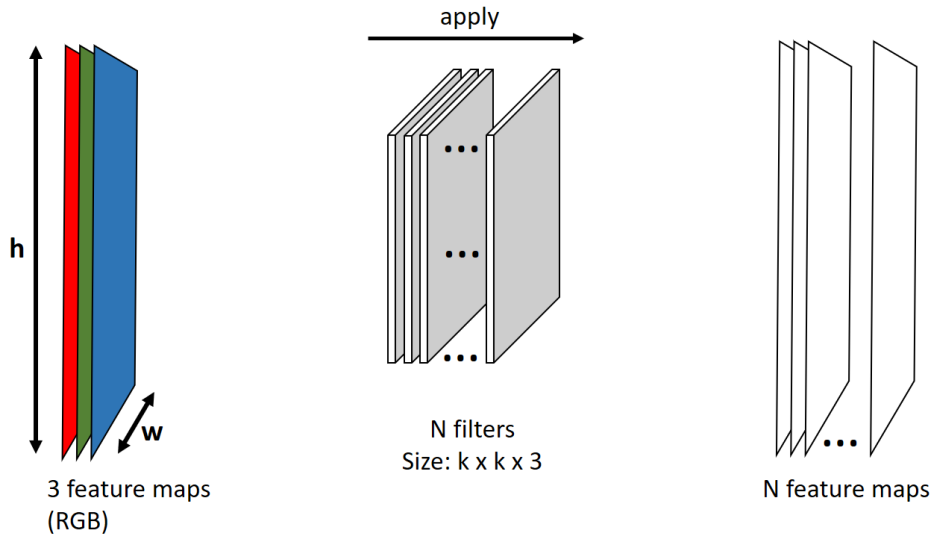


Figure 2.4: The convolutional layer with N filters of size (k,k,3) applied on an input FM with dimensions (w,h,3) [Gai18]

As shown in Figure 2.5, hidden layers in a CNN usually include convolutional layers, pooling layers, Fully Connected (FC) layers and non-linearity layers [AMAZ17].

However, other layer types may also be used in the model architecture, depending on the design of the network. We will briefly review these layer types in the following part.

- *Convolution (Conv) layer* Convolutional filters are used to derive an activation map from the input data. Kernel size, number of the kernel, activation function, *stride(S)* and *padding(P)* are hyper parameters of this layer [Gai18]. Padding and strides helps to control the output shape. Stride is the number of pixels by which the kernel slides over the input image. For example, stride of 1 means receptive field shifts one unit at a time. Padding often called zero padding adds a pad of zeroes with length P around the input image [Elg20].

- *Pooling Layer.* Using more convolutional layers, increases the number of parameters(weights) to be learned during the training phase. This directly impacts the computational complexity as well. To solve this problem, pooling layers help to decrease the spatial resolution of the FMs [Elg20]. they are placed after the convolutional layers and do not have any parameters. The pooling layer is defined as the sliding window which overlays the FM. Two main types of these pooling layers are *Max* pooling and *Average* pooling layers [Elg20] (Figure 2.6). Average pooling layer replace the numbers inside the window with the average value of them while Max pooling layer takes the max pixel value of them. Based on their role, pooling layers can be conceived as a layer in charge of reducing the input via aggregation operations.

- *Global average pooling layer.* This layer replaces the values inside a FM with the average of their values [LCY13]. As a result of this process an array that is three dimensional get reduced to a vector.

- *FC Layer.* These layers often serve as classifier on top of the feature extractor part [Gai18].

- *Dropout Layer.* One of the layers in the model design to avoid *over-fitting* in training phase is a dropout layer. Based on a hyper-parameter, part of a layer's neurons are turned off in a random way [HSK+12, SHK+14]. The goal of this approach is to force the model to learn redundant paths for a given output, and hence making it less likely the over-fitting [Gai18].

- *Batch Normalization (BN) Layer.* BN layer can be placed after a FC layer or a Conv layer. It normalizes adjacent output channels based on statistics of the current mini-batch [Gai18]. This can help training acceleration. It also adds some kind of regularization to the network [IS15].

- *Non-linearity layer.* Conv and FC layers are responsible for linear transformations in the network and non-linearity can be introduced by activations [ZW19] which can accelerate the training and reduce the over-fitting [Gai18]. Rectified Linear Unit (ReLU) activation function family is one of the most popular one in CNNs. If the input values are positive, ReLU returns the same result, otherwise it returns 0 [Gai18].

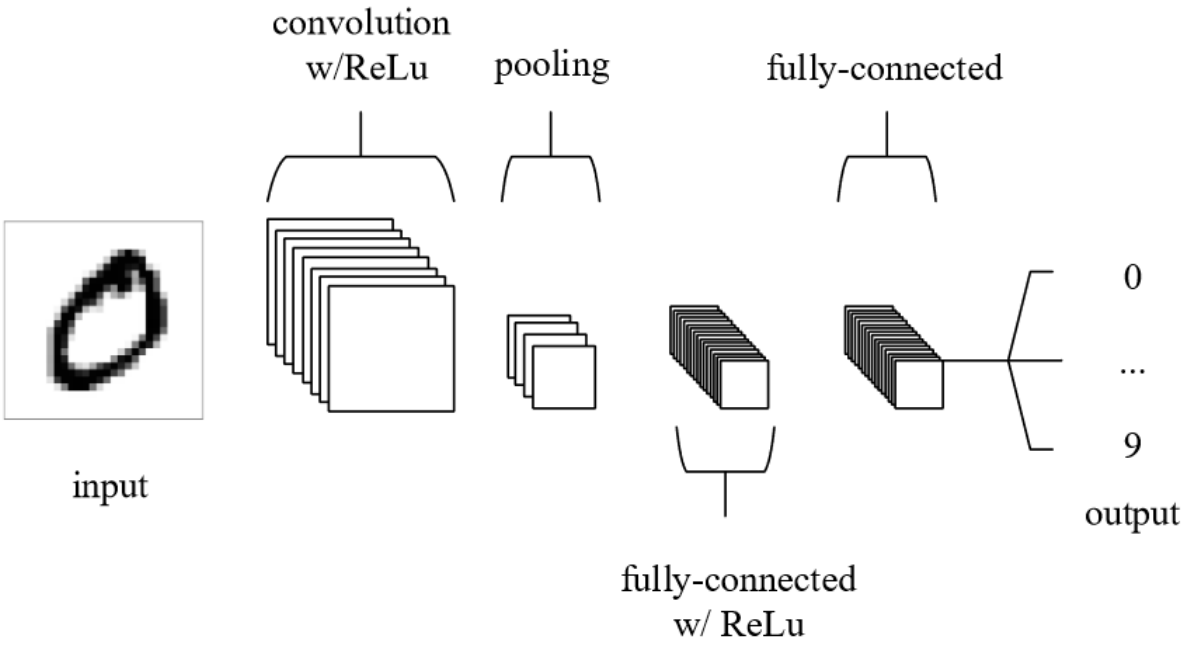


Figure 2.5: A simple CNN architecture [ON15]

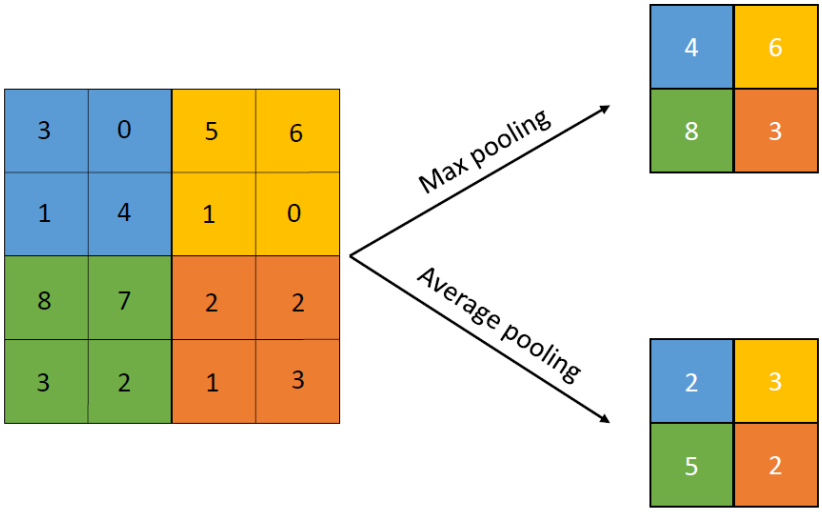


Figure 2.6: Max and average pooling with kernel size of 2 and stride=2

2.1.3 Benchmarked CNN architectures

In the previous section, we discussed the components that comprise a CNN. In this section, we cover some popular CNN architectures used in image classification tasks. In an image classification pipeline, an image classifier, which in our case is a CNN model, receives an input which could be an image or a sequence of images captured through a video. A set of preprocessing steps has to be applied on these inputs to standardize them. This process is called *data augmentation*. The next step is feature extraction: a CNN model tries to extract different features from an image using its various filters located in different depths. These features help to find and define objects. The next and last step is the image classification itself. The CNN model uses the extracted features to label or classify the input image. The output is a vector of probabilities, from which

the highest probability should identify the correct class. This is a typical structure for a CNN: composition of *feature extraction* and *classification* [Elg20].The feature extractor is made up of stacked convolutional layers, while the building blocks of the classifier are fully connected layers.

In the next sections, we cover a series of example models, that illustrate the common structures of CNNs used of for image classification.

AlexNet

AlexNet is the winner of ILSVRC[1] an image classification challenge in 2012. It is designed by Alex Krizhevsky [KSH12] and by its time it was the first deep neural network in computer vision domain and its drawbacks motivated scientists to design the next deeper CNNs [Elg20]. As Figure 2.7 represents, AlexNet has the usual feature extraction section with stacked convolutional and pooling layers. The classifier consists of two FC and one softmax layer. They also utilized ReLU as non-linearity activation function. To deal with the great risk for over-fitting caused by having a model with 60 million parameters, they applied dropout layers.

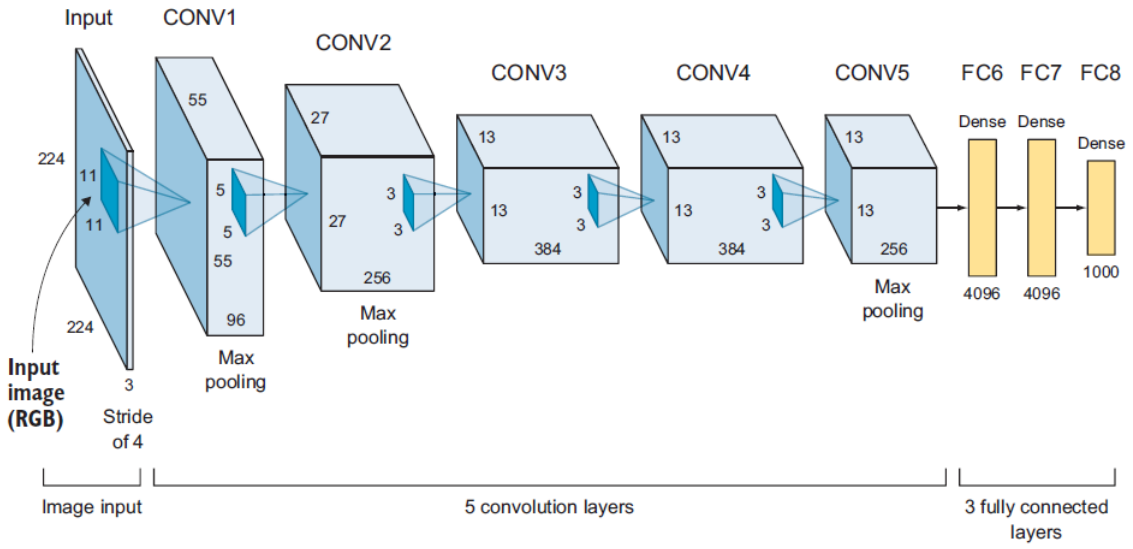


Figure 2.7: AlexNet architecture by [KSH12, Elg20]

VGG Net

This NN was a runner-up in the ILSVRC 2014 competition, developed by Simonyan and Zisserman [SZ14]. It proved that the performance of the network directly relates to its depth. This network (Figure 2.8) has 16 stacked layers; 13 of them are convolutional ones, which perform 3×3 convolution and 2×2 max pooling. In between, there are ReLU activation layers. The remaining 3 layers are FC layers containing the most parameters [Elg20]. It has $\sim$ 140 million parameters and managed to achieve an accuracy of 70.0% although it is computationally expensive [LCO18].

[1] http://www.image-net.org/challenges/LSVRC

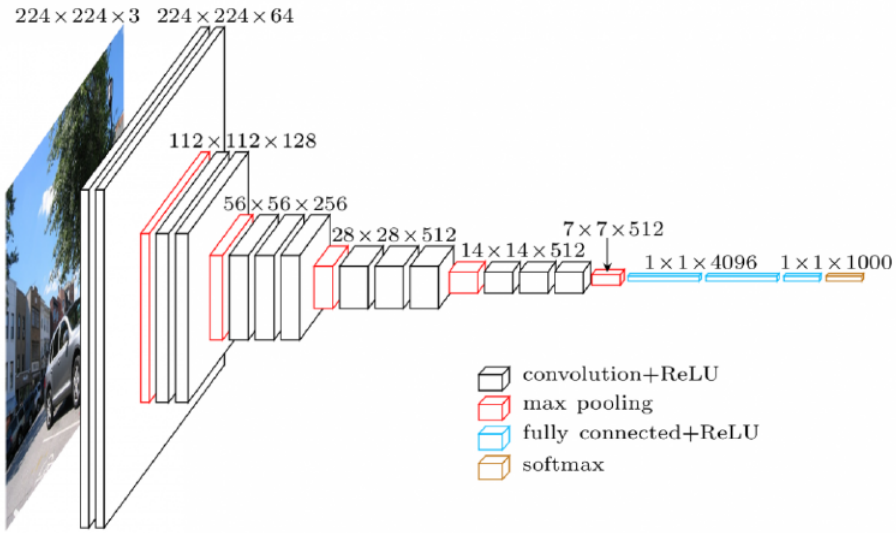


Figure 2.8: VGG(also known as VGG16) network architecture [LCO18]

GoogleNet

This CNN model was the winner of the ILSVRC 2014 challenge, and it was introduced by Google [SLJ+15]. This network with 22 layers is deeper than VGG16, yet it has 12× fewer parameters and yields more accurate results [Elg20]. This model is based on a new module named *inception* Figure 2.9. There are multiple Conv layers with different filter sizes in this module. GoogleNet differs from classical architectures by stacking inception modules rather than stacking layers.

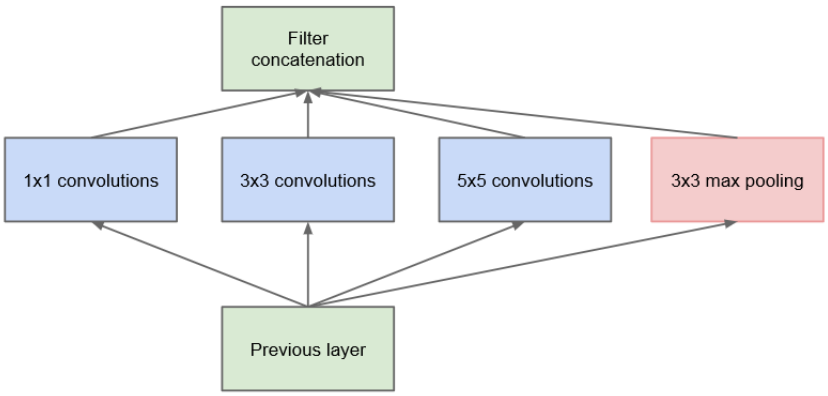


Figure 2.9: naive version of inception module [SLJ+15]

Residual Networks (ResNet)

ResNet is another architecture proposed, that has its strengths in an [HZRS16a, HZRS16b] *identity shortcut* or *skip connection*. Instead of just stacking convolution layers, this connection adds the input to the output of convolution block and forms a *residual block*, depicted in Figure 2.10. Using these shortcuts the network is transferring the information between shallow and deeper layers and prevents from accuracy degradation [LGWS21].

In cases when the dimension of input and outputs do not match up, we need another 1×1 convolutional layer in the identity shortcut to resize the input which results in a *projection shortcut*(Figure 2.10-c). In each residual block, convolutional layer is followed by a batch normalization layer and a ReLU activation function. This structure can be repeated inside each block based on model configuration. In basic blocks there are two repetitions of them (Figure 2.10-a) and *bottleneck* blocks, are composed by stacking three convolutional layers with filters of size 1×1 (to reduce input dimension), 3×3 and 1×1 (to raise the dimension) respectively (Figure 2.10-b). Based on different model requirements,

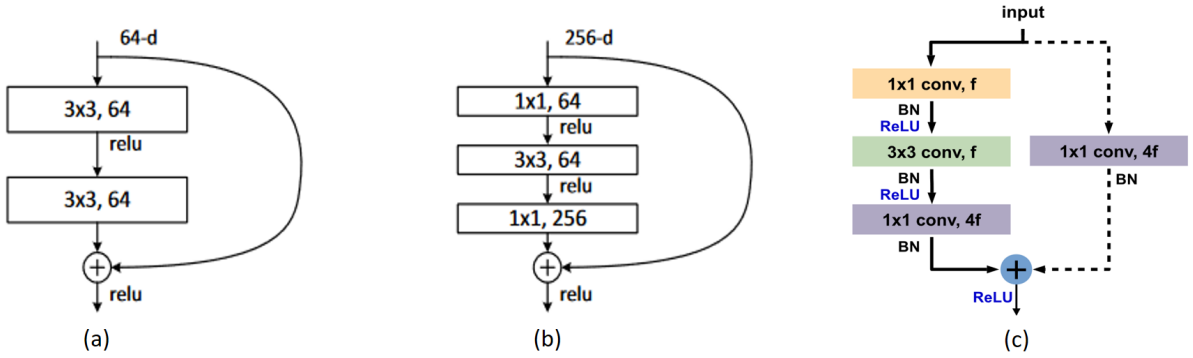


Figure 2.10: Different building blocks of ResNets [HZRS16a]. (a) basic block with identity shortcut, (b) bottleneck block with identity shortcut, (c) bottleneck block with projection shortcut [DRRT+18] [HZRS16a].

a different number of these blocks can also be stacked to make a stage. As an example, the variety of shallow and deep resnet models for ImgaeNet dataset are illustrated in Figure 2.11.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.11: Architectures for ResNet based on ImageNet [HZRS16a].

MobileNet

In 2017, Howard et al. [HZC+17] first introduced the light weight class of models called MobileNet, which are efficient for embedded vision tasks on mobile devices[ES21]. The

idea of MobileNet is to replace the computationally expensive convolution layer (Figure 2.12-a) by cheaper *depth-wise separable* convolution. This factorized-like convolution is made up of two different convolution operations: a 3×3 *depthwise* Conv layer followed by a 1×1 *pointwise* convolution (Figure 2.12-b). In other words, the depthwise Conv is getting features of each channel individually and then through pointwise convolution it is creating a linear combination of each of them, which results in less parameters consumption and acceleration [XWL+19].

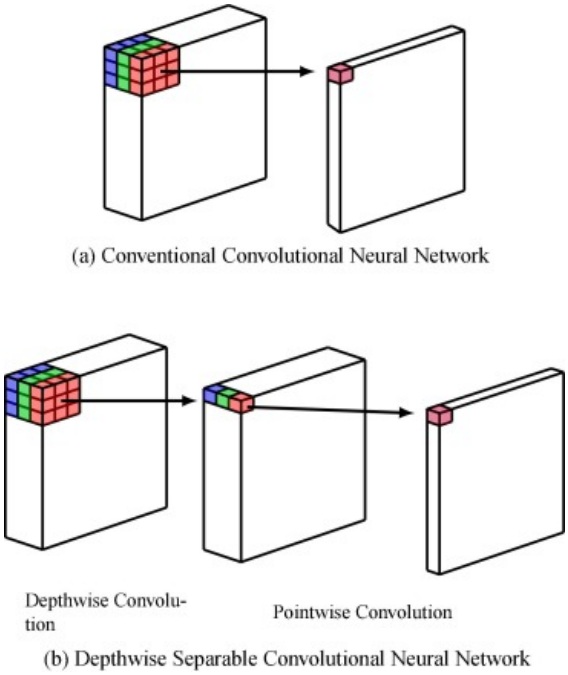


Figure 2.12: (a) Convolution mechanism in CNN , (b) Depthwise Separable CNN [KYWW19].

Each of these layers is followed by a BN layer and a ReLU activation layer, respectively. This was the main building block for MobileNet V1 and later Sandler et al. presented the MobileNet V2 [SHZ+18] . Each building block in V2 is a combination of inverted residuals and linear bottlenecks which is named a *Bottleneck Residual* block, illustrated in Figure 2.13. As illustrated in Figure 2.13, this block includes an expansion layer which is a 1×1 convolution and it is used to increase the input dimension. This layer is followed by BN layer and ReLU6 for introducing non-linearity. Next is a depthwise Conv with BN and ReLU6 layers. Then is the projection layer to decrease the dimensionality followed only by a BN layer. MobileNet V2 is faster and more compact than the first version also its performance is better than MobileNetV1, and some competitor models like ShuffleNet,and NASNet [ES21].

After introducing CNNs, and reviewing common models, among which we identified MobileNet as a special class of models that is improved to be light-weight and thus amiable to use in edge devices; in the following section we look closer into methods for compressing CNN. This kind of methods would be useful for any kind of CNN architecture, be it a large model like VGG models or a more compact model like MobileNet, helping to make the end model more memory efficient and better for reducing energy consumption and improving real-time inference operations.

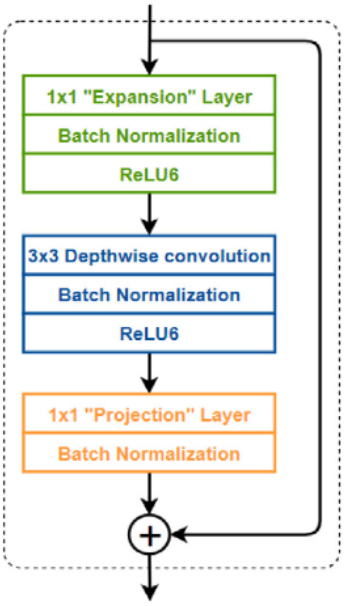


Figure 2.13: MobileNet V2- Bottleneck residual block [KMGB21]

2.2 CNN Compression Techniques

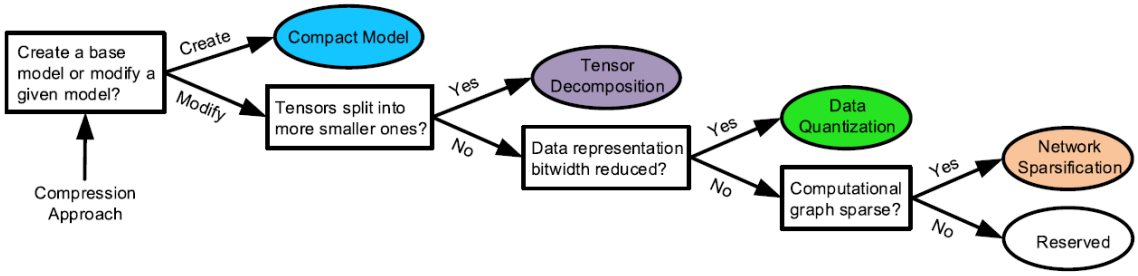


Figure 2.14: The hierarchy of neural network compression algorithms [DLH+20]

Recently CNN models are getting attention in different applications. In some usecases, these models should be deployed on embedded or mobile devices, which imposes hardware restriction. To make conventional CNNs fulfill these requirements, a large amount of research has been going on related to DNNs compression and acceleration approaches. The goal of these approaches is to simplify the original model without significantly reducing its accuracy. This efficiency improvement can have different perspectives: size and/or latency reduction. In size reduction, the goal is to produce a memory friendly version of the network with less or smaller parameters. In latency reduction, the purpose is to shrink the computation cost, which then provides a reduced inference time (the time which takes the model to make a prediction). This can result in an energy efficient network. These optimization approaches can be categorized into different groups. [DLH+20] classifies these approaches into four categories, as depicted in Figure 2.14. During compact model design, there is no base model to optimize. The aim is to design a compact model [DLH+20]. The goal of tensor decomposition is to approximate the tensor/matrix parameters using a low-rank version of the main tensors. These methods may also be named low-rank factorization. Quantization techniques are designed to reduce the weight tensors'

size. In so doing, bit-widths for data representation are reduced. On the other hand, in pruning methods, the focus is on reducing the redundancy of parameters in the network by removing parameters that are not that useful. Some papers refer to this category as sparsification. Other surveys, [CMGS20, CWZZ17], suggest also another category named KD. By distilling knowledge into a compact neural network, we can reproduce the output of large networks. We will discuss these approaches through the rest of this chapter, as they are relevant for our research. We dedicate two separate subsections to pruning and quantization techniques, as these are two approaches which we investigate in this research. These approaches are specially relevant as they are very general and can be applied in combination with any other of the approaches discussed.

1. *Compact network design* The focus of research in this category is on designing an efficient network architecture. For instance, GoogleNet [SLJ+15] and Squeezenet [IHM+16] apply pointwise convolution (1×1) to reduce the number of FMs' channels before applying computationally expensive 3×3 or 5×5 convolutions [ZYYH18]. Since the goal of our research is to optimize a trained model, this category is beyond the scope of our work. We refer the interested reader to the survey paper by Deng et al. [DLH+20], for more details. Another relevant approach is using Neural Architecture Search (NAS)-based solutions such as Proxyless NAS [CZH18], which automatically designs the optimized architectures.

2. *Low-rank factorization* Low-rank Matrix Factorization (MF) is a technique to compress representation of data by using an existing latent structure in the data. To achieve this, a matrix $A \in \mathbb{R}^{m\times n}$ with rank of r is decomposed to smaller dimension matrices. In CNNs, using the factorization, layers can be decomposed and represented by new sequences of smaller layers, to reduce the number of parameters and improve overall speed up. Both convolutional and FC layers can be viewed as a 2 dimensional matrix and therefore low-rank factorization techniques can be applied on them. Decomposing FC layers can target reducing the model size. While, factorizing convolutional layers can accelerate the model at inference time [CMGS20]. Singular Value Decomposition (SVD) [KL80], is a popular factorization method used in CNN optimization. SVD decomposes the original weight matrix W of size $m \times n$ into three matrices which satisfies:

$$A = USV^T \tag{2.4}$$

where $U \in \mathbb{R}^{m\times r}$, $V^T \in \mathbb{R}^{r\times n}$,$S \in \mathbb{R}^{r\times r}$ and $r \leq min\{m, n\}$. U, V^T are orthogonal matrices and S is a diagonal matrix of singular vectors. To eliminate the limited compression rate of 2D matrix factorization, *tensorized* decomposition also received attention in CNNs [DLH+20]. Tucker decomposition [KPY+15, GKP+19] and Tensor Train decomposition [NPOV15, GPNV16] are research that studies in this direction. Moreover, finding an optimal approximation of a low-rank factorization can be formulated as an optimization problem. The authors, in [LS18], solved this optimization problem for a trained CNN by applying two constraints of MAC operations and memory usage. Existing techniques in this category are mostly examined on small scale datasets [DLH+20] and for some of them extensive model retraining is required for convergence [CWZZ17]. Further investigations are also required to prove effectiveness of these methods for Conv layers while there are more challenging than FC layers [DLH+20].

3. *Knowledge Distillation* The goal of KD techniques is to transfer information. They can be used for model compression where training of a smaller model (student) is supervised by provided knowledge by a bigger network or an ensamble of networks (teacher/s) [HVD15, UGK+16]. Figure 2.15 illustrates a generic version of this method. In this process, different aspects impact the learning quality of the student. The type of the knowledge which is being transferred is one of them. In some works, knowledge of logits (inputs to the softmax layer) of a teacher model is considered [HVD15, MFL+20]. More substantial information can be provided through intermediate layers. Output of hidden layers in a teacher network supply this knowledge as a *hint* [RBK+14]. In some cases, multiple teachers can give more diverse knowledge rather than only one teacher [ZSD+20, YSP+20]. KD-based approaches shows a lot of accuracy degradation when they are compressed intensively [GKD+21]. Another issue is related to training: most of the existing methods make the assumption that training data is available during optimization [CCEKL20]. For more details about methods in this category, we refer the interested reader to the survey by Gou et al. [GYMT20].

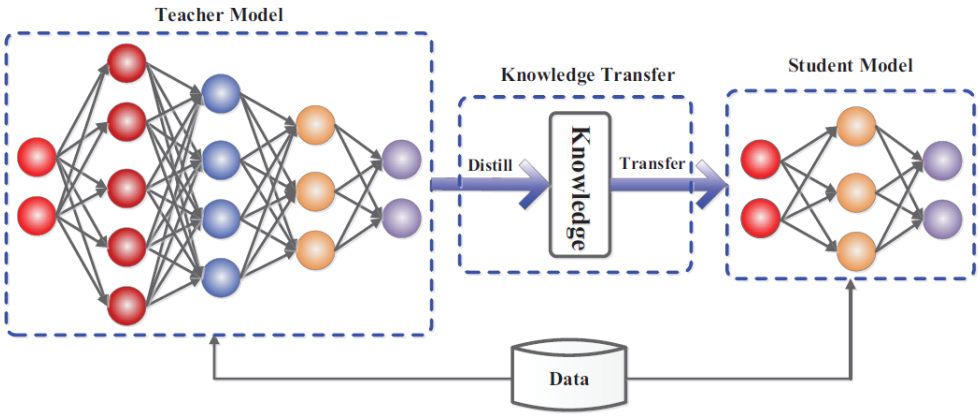


Figure 2.15: The generic teacher-student framework for knowledge distillation by [GYMT20].

2.2.1 Quantization

Floating point 32-bit representations are often too precise for many networks. By converting FP32 parameters to lower bit representations, bandwidth, energy, and memory consumption can be significantly reduced [LGWS21]. The idea of neural networks quantization dates back to the 1990s [FCC90, BTOK91]. In an extreme case each real value is represented with one bit which is named *Binarization*. Existing binarization schemes like XNORNet [RORF16], BinaryConnect [CBD15] and BinaryNet [CHS+16] learn the binary weights during the training. In another research, this idea is extended to activations and authors proposed a so called binarized neural network (BNN) [HCS+16]. Another kind of low-bit quantization is *ternarization*. In Ternary Weight Networks only weights and in Ternary NN both weights and activations are constrained to $\{-1, 0, 1\}$ values [LGWS21]. One of the main issues related to these approaches with precision bit less than 4 is their failure in accuracy compensation and it gets worse even with deeper models such as GoogleNet [DLH+20, CWZZ17]. For CNN models quantized with bits greater than eight, minor accuracy changes are expected.

Quantization Details:

Different components of the network can be quantized including weights, gradients, activations, error, and weight update [DLH+20]. These quantization techniques can happen during the training or after training which are named Quantization Aware Training (QAT) and Post Training Quantization (PTQ) respectively. From these aprpoaches, quantization after training is specially applied with the intention of saving energy and inference time. In some models specially bit-width lower than 8 , quantization needs a training step to compensate for the accuracy loss. The main drawback of QAT methods is their data dependency and computational cost, as they require hundreds of training epochs [GKD+21]. On the other hand, the PTQ approach does not need fine-tuning and a small size of unlabeled data is sufficient for it. However, this brings often lower accuracy specially when the precision is low [GKD+21]. Considering $x \in [\alpha, \beta]$, the real value which should be quantized and k number of bits for the quantization, then then quantization function Q can be defined as below [WJZ+20]:

$$Q_k(x, k, \Delta, z) = f(g(\Delta.x + z), -2^{k-1}, 2^{k-1} - 1) \tag{2.5}$$

where $\Delta \in \mathbb{R}$ is the scaling factor or also named quantization *interval/step-size* which can be calculated using various methods, $z \in \mathbb{I}$ stands for zero-point to adjust the zero point, $f(.)$ is a clipping and $g(.)$ is rounding function [Kri18, LGWS21].

Existing quantization methods can also differ based on the following parameters for their algorithm:

- *Range mapping* range mapping of $[\alpha, \beta]$ defines the type of quantization. In *symmetric* quantization (a variant of scale quantization [WJZ+20]), α and β have equal values (Figure 2.16-left) and z value in Equation 2.5 will be zero and step size is calculated by following equation:

$$\Delta = \frac{2^k - 1}{\alpha} \tag{2.6}$$

  In *asymmetric* quantization mapping ranges are not equal (Figure 2.16-right) and step size is calculated using Equation 2.7.

$$\Delta = \frac{2^k - 1}{\beta - \alpha} \tag{2.7}$$

  Two common approaches to define the clipped data ranges are as below:

$$[\alpha, \beta] : \begin{cases} \alpha = -\beta = max(|x_{min}|, |x_{max}|) & Symmetric \\ \alpha \neq -\beta, \alpha = x_{min}, \beta = x_{max} & Asymmetric \end{cases} \tag{2.8}$$

  Although asymmetric solutions are much accurate, having z parameter increases the computational cost which is a barrier at inference time [GKD+21, Kri18].

- *Level Projection* Existing approaches can map a real value data to its quantized value in two ways: stochastic and deterministic. In deterministic quantization, each high-precision value is exactly mapped to a a single discrete level (Figure 2.17(a)) [ZWN+16, MEA+18]. While in the other group, the real value is mapped to its

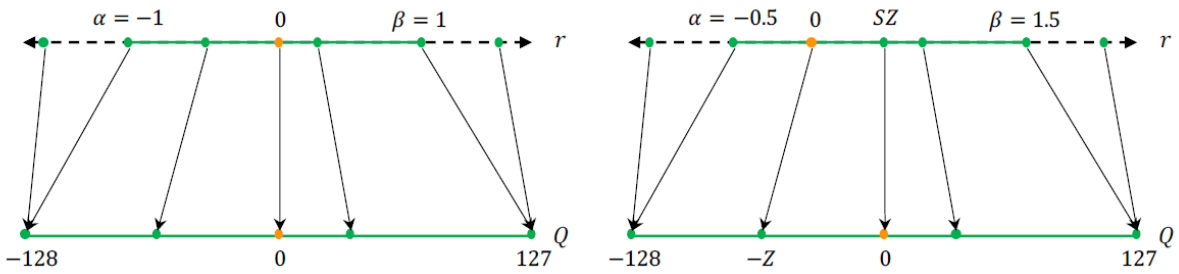


Figure 2.16: Symmetric(left) vs. Asymmetric(right) quantization by [GKD+21].

adjacent discrete levels with a degree of probability (Figure 2.17(b)). In recent work [KK20], authors proposed an approach which is a mixture of both schemes.

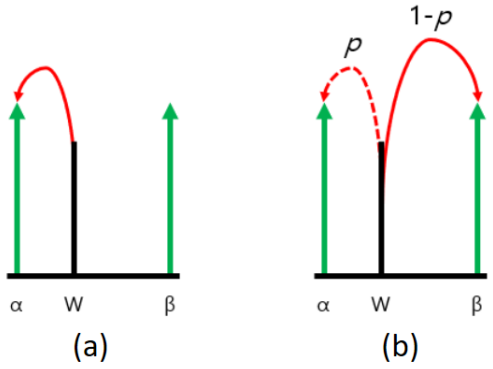


Figure 2.17: (a) Deterministic and (b) Stochastic quantization by [KK20].

- *Level Distribution* In some methods, quantization levels or values (y-axis in Figure 2.18) are evenly spaced which results in *uniform* or linear quantization (Figure 2.18-a). There are other categories which quantization step size and levels can vary based on the data distribution [LGWS21]. These *non-uniform* methods may achieve a better performance while providing more resolution for important regions [DLH+20]. Methods using Logarithmic distribution (Figure 2.18-b) are typical examples of non-uniform quantization [MLM16]. For further improvements, this task can be formulated as an optimization problem. The quantizer is learning to find the best step size by minimizing the difference between a real value tensor and a quantized version of it [DLH+20, GKD+21] (Figure 2.18-c). In comparison to non-uniform methods, uniform schemes are more common while they are simple and can be easily deployed on general hardware [GKD+21].

- *Precision* Different layers in a network have different quantization properties. Not all the layers have the same sensitivity towards same quantization setting. This indicates that varying the bit-width for layers can help in getting more optimal results [NBBW19]. The approach proposed by [SLW16] is beneficial for small networks while it deploys an exhaustive search to find bit-width configurations. A better solution for mixed precision quantization is layer wise bit-width optimization [ZMDCF18]. Existing optimization approaches can be divided in two groups of minimizing quantization error and minimizing loss function [Guo18]. Lin et al.

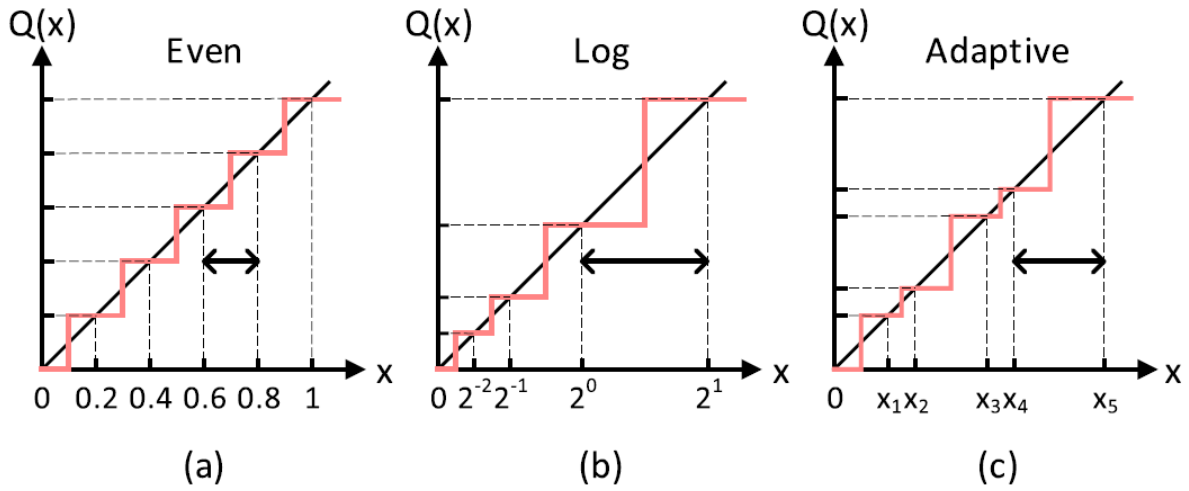


Figure 2.18: Quantization level distributions by [DLH+20]

[LTA16] measures the quantization error using Signal to Quantization Noise Ratio (SQNR) to measure the resulting error in each layer. Although this approach does not certainly guarantee the best accuracy of the model [KYH+18].

- *Granularity* Quantization can be done based on statistics of different components of the network such as: layers and channels [GKD+21]. In *layerwise* quantization, clipped data range to calculate the step size mentioned in Equation 2.8 is determined using data distribution through the whole layer. This method can cause a degree of error with outlier presence [GKD+21]. Another approach is allocating each channel of a filter a separate step size [Kri18]. *Channelwise* quantization can handle the outliers and yields better accuracy. However, this comes with computational overhead and in some methods we need special hardware to benefit from it [NBBW19, HPZ+21].

Quantization Noise

Low bit-width quantization induces some degree of error to the network which ultimately affects the model performance. Different models based on their architecture and weight distribution can have different performance degradation. Usually there are layers which are more sensitive towards low-width quantization. There are different approaches to deal with the quantization error. The straightforward approach is applying quantization-aware training [MFAG19]. By training the quantized model from scratch, it gets an opportunity to compensate its accuracy loss with the help of less problematic layers [GG20]. However, this approach is time-consuming and relies on the training dataset which is not applicable in some cases. In addition, this approach does not provide any insight about the source of the error [GG20]. One of these sources is imbalanced data ranges in different channels of a weight tensor. As plotted in Figure 2.19, authors have [NBBW19] observed a huge difference between output channels of a filter which with applying the same quantization configuration can not distinguish between a channel with range of [-100,100] and another channel with smaller weight range like (-0.8,0.8). Consequently, all weights in later channels will turn to zero after quantization. This problem is mostly observed in depthwise separable layers in MobileNet [NBBW19]. To solve this problem, some authors

have[Kri18] proposed channel-wise quantization, which applies customized quantization per output channel. However, this method introduces an extra computational overhead

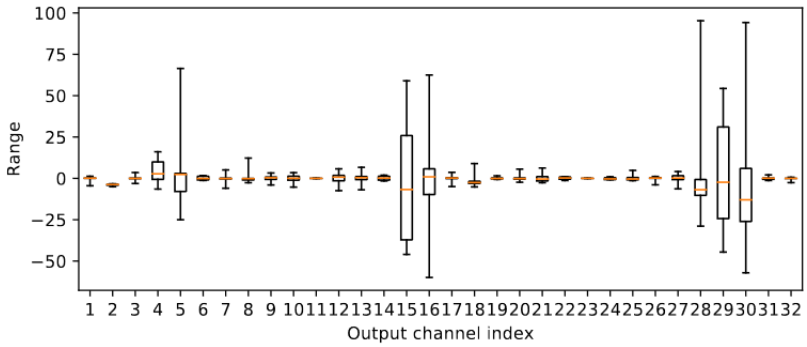


Figure 2.19: channel wise weight distribution of first depthwise separable layer in MobileNetV2 by [NBBW19]

and it is also hardware restricted [GG20]. Authors in further work[SFZ+18] developed a new architecture design for separable convolutions in MobileNetV1. In this design, they just keep the BN layers and activation after pointwise convolution and replace ReLU6 with ReLU. To use this approach, we have to retrain the model with the new architecture and then apply the quantization. Also, authors did not investigate its performance for the other networks. There is another category of solutions named post-training methods, such as channel equalization [NBBW19, MFAG19]. Nagel et al. proposed CLE to solve this range inbalance. In this approach, they benefit from the ReLU activation scaling equivariance property:

$$A(s.x) = s.A(x) \tag{2.9}$$

For each consecutive layers in a network, each output channel i has its own scaling factor which can be reached using the following formula:

$$s_i = \frac{\sqrt{s_i^{(1)} s_i^{(2)}}}{s_i^{(2)}} \tag{2.10}$$

which $s_i^{(1)}$ and $s_i^{(2)}$ are maximum absolute value for channel i for those two consecutive layers. This process can be applied iteratively for each pair until all layers get scaled:

$$\begin{aligned} y &= f(W^{(2)} f(W^{(1)} x + b^{(1)}) + b^{(2)}) \\ &= f(W^{(2)} S \hat{f}(S^{-1} W^{(1)} x + S^{-1} b^{(1)}) + b^{(2)}) \\ &= f(\widehat{W}^{(2)} \hat{f}(\widehat{W}^{(1)} x + \hat{b}^{(1)}) + \hat{b}^{(2)}) \end{aligned} \tag{2.11}$$

where $S = diag(s)$ is a diagonal matrix based on the scaling factors for each neuron. The prerequisites to apply this technique are a network with folded BN layers, activation function should be ReLU, and for chosen pair of layers output of the first layer should directly connect to the input of the second layer without any split in between [NBBW19]. Another paper which follows almost the same idea is presented by [MFAG19]. They named it inversely proportional factorization or *channel equalization*. Their approach proposes two algorithms, one-step equalization and two steps equalization, which is an optimal equal-

ization. They prove that this method increases the SQNR and improves the quantization performance. In one-step variation, it calculates the scaling factor using weight tensor and the layer's activation output FMs. It takes the maximum absolute value per channel for each of them. Then divides the corresponding values by their max values, which results in scaling vectors S_w and S_a for weight tensor and activation output respectively. At the end, a final scaling vector is calculated by:

$$S = min(S_w, S_a, S_{max}) \tag{2.12}$$

where S_{max} is to limit the maximum values. This method can be applied for a network with homogeneous activation function that satisfies Equation 2.9. Another possible source of noise is Mean Activation Shift (MAS) [FAG19] or biased error [NBBW19]. In some cases, a large number of rounding errors in quantization results in a shift in the activation output of the quantized layer. This can mainly cause errors for layers with small parameters per output channel like depthwise convolution which has 9 weight parameters per output channel.[NBBW19, FAG19] suggested using the bias term of the channels to compensate this quantization induced shift. IBC is an empirical bias correction method which is fast and requires a small dataset. Since bias correcting of each layer changes the statistics of output activation, it should be done in an iterative manner. Considering the $act_{l,i}^{orig}$ and $act_{l,i}^{quant}$ as the mean of activation values for each channel i in the layer l of the original and quantized models, this error is calculated by:

$$\Delta_{l,i} =< act_{l,i}^{orig} > - < act_{l,i}^{quant} > \tag{2.13}$$

later the error is added to the bias vector of the layer. Authors claimed that a batch size of 8-64 images from the calibration set is sufficient to get good results. Further, authors [NBBW19] propose the same approach, however they use pre-activations values instead of activations. This bias correction can also be done analytically for the models which have BN layers and are data-usage restricted [NBBW19]. Another method to reduce the biased error is Bias Fine Tuning (BFT) [FAG19]. This method formulates the problem as an bias optimization one. It uses a QAT process with only bias terms as trainable parameters to reduce the accuracy degradation.

In sum, quantization can be incorporated to a model during (QAT) or after the training (PTQ). Higher accuracy and higher computational costs are associated with quantization during training. Some important algorithmic choices are the range mapping, the level projection or distribution, and noise compensation strategies.

2.2.2 Pruning

In this category of compression techniques, methods apply sparsification aiming to decrease the number of operands. Pruning methods target compactness, acceleration or both of them. Most of the previous work on pruning adopts a three-stage pipeline which is depicted in Figure 2.20. Input of this pipeline is an already trained over-parameterized network. The pruning algorithm based on an importance metric, evaluates the importance of operands. The next step is the selection and pruning of them. At the end, the model needs a fine-tuning stage to recover its accuracy [LW20, LSZ+18]. Different from quantization, most works in the pruning category needs retraining to regain model accuracy. In

following parts, we will talk about the details of each stage.

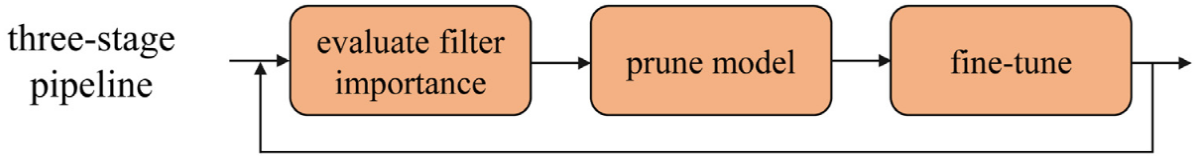


Figure 2.20: Overview of three-stage pruning pipeline by [LW20].

Pruning details

In general, the proposed methods for pruning can vary based on different techniques and choices used for *pruning level, importance metric, pruning rate,* and *pruning schedule*. These aspects are described shortly below:

- *Pruning Level* Network pruning methods can be applied on different levels of granularity [LLS+17]. The fine-grained level (e.g., weight-level) pruning gives the highest flexibility and provides a higher compression rate [HMD15], as shown in Figure 2.21(a). However, these methods result in an *unstructured* sparsified model which requires special software (sparse libraries) or hardware accelerators [ZWHD20, PBP+18] to take benefit of fast inference [CMGS20, DLH+20]. To overcome the drawbacks of non-structured pruning, a *structured* way of pruning was introduced [MLY+21]. It can be applied in the format of: *layer pruning, channel pruning, filter pruning, block pruning* and *shape-wise pruning*, visualized in Figure 2.21(b) [LMX+20, LLS+17, WZW+19]. The coarsest level of pruning is layer level, which is restricted by the depth of the network [LLS+17]. Block pruning which is focusing on removing building blocks of the model such as residual blocks is also limited to the network structure [WNK+18]. With more flexibility, in filter and channel pruning the whole specified filter(s) and channels of a layer get removed respectively [LW20]. Layers are pruned shape-wise, such as in column pruning, by removing filters from the same position [LW20, WZW+19].
- *Importance Metric* Filter pruning approaches require some measure to calculate the importance of the filters/weights. Based on this metric for each layer a list of candidates are selected to be pruned away. This selection should be in a way which causes the least performance drop. The most intuitive way is pruning based on the weight *magnitude*. [HPTD15, HMD15] authors commonly discard the weights which are below a given threshold. The magnitude criterion is also extended to filter level pruning. In [LKD+16] L_1-Norm of the filter (sum of its absolute weights) is utilized for this purpose. They assume that filters with a smaller magnitude have the less impact on the FMs and can be pruned. [HKD+18] applies L_2-Norm to select the filters and prune them softly. In other category of metrics, we can use the information resulted in each filter's activation channel to discriminate it from the others [LGWS21]. This information may be the amount of sparsity in the activations. Hu et al. [HPTT16] proposed calculating Average Percentage of Zeros (APoZ) for positive activations of a layer to determine the importance score of each corresponding filter. More zeroes indicate that the related filter is redundant and

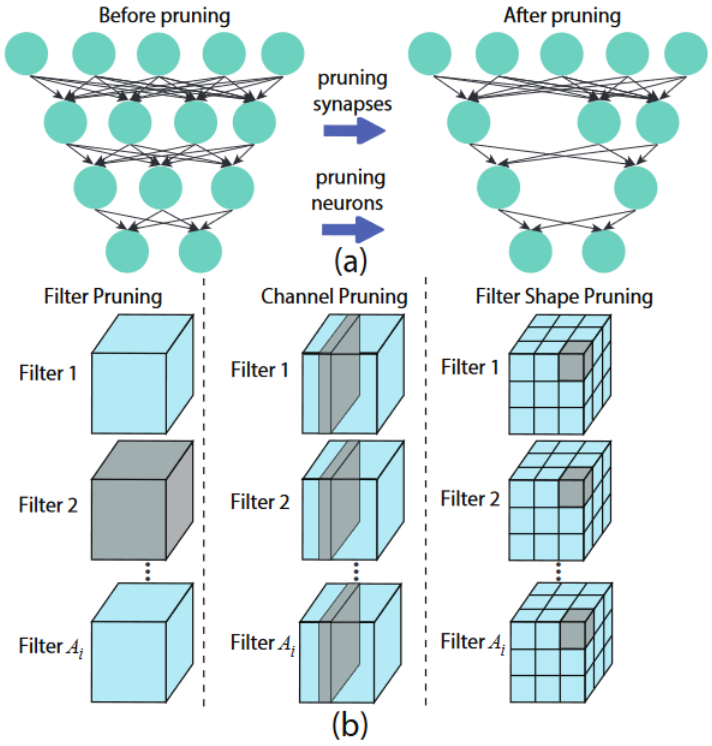


Figure 2.21: (a) Unstructured weight pruning [HPTD15], (b) Different types of structured pruning [LMX+20] by [WYH+19].

can be removed. This can be accomplished by defining layer-wise thresholds [LW19]. This metric might miss redundancy of values which are close enough to zero or are always the same [LW17]. In [LW17], authors proposed entropy-based criteria. It believes that if the activations of a filter are always similar (smaller entropy value), this filter is less informative and can be removed. In another work, authors used information in input FMs rather than output FMs for their investigation [LLZ+19]. They calculate an indicator using filters kernel sparsity and entropy (KSE) which shows the importance of each input FM. One advantage of this approach is that the output dimension of the layers remains the same [LLZ+19].

In contrast to magnitude based pruning criterion which does not consider the correlation between the filters [HLW+19], in another group of the pruning methods, *relational* criterion is investigated. The difference between these two categories of metrics is illustrated in Figure 2.22. Authors in [HLW+19] utilize Geometric Median (GM) to find this relation. They believe that filters which are close to the GM provide the same information. Consequently, those filters are redundant and can be replaced with the remaining ones. [WZH+20] also uses the same metric for an image retrieval acceleration task. Another approach which also considers dependencies between filters is provided by [SVRN20]. It calculates the correlation score for each pair of filters using *Pearson correlation coefficient*. Pairs with larger correlation tend to present redundant information and get lower importance score. Close to this idea authors in [DDGH19], recognize correlated filters by applying K-means clustering and then a proposed Centripetal Stochastic Gradient Descent (SGD) to make each cluster's members identical and later model is getting trained with this redundant pattern and then transformed to a slimmer network independent of fine

tuning. Another investigated point is the adaptive filter pruning criteria. In recent work [HDL+20], authors suggested a framework which based on each layers characteristics, it is able to learn the best pruning criteria. Meta filter pruning proposed by [HLZY19] also examine the effect of changing data distribution during training step of pruning process. It chooses the appropriate pruning criteria for each layer based on some meta attributes.

After choosing the appropriate pruning metric, the components based on the pruning level will get scores and then will be ranked based on it. This ranking can happen locally (e.g., filters in a layer) [HPTD15] or globally, which compares the score of the components regardless of their placement in the network [LAT18].

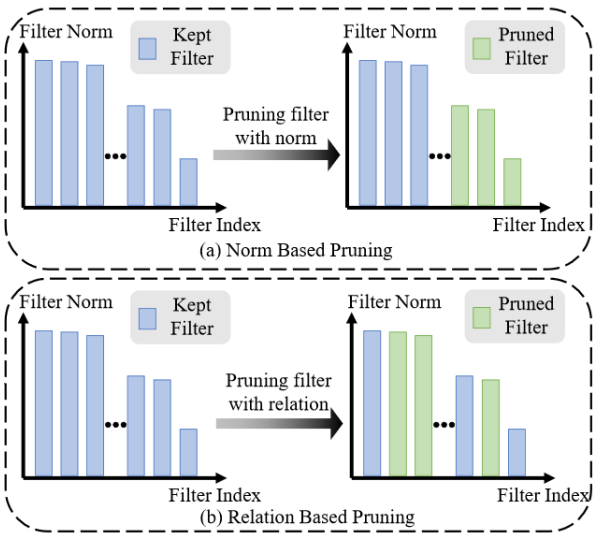


Figure 2.22: pruning criterion: l_p-Norm based vs. relation-based [HLZY19]

- *Pruning Rate* Another effective hyper-parameter in the pruning process is the pruning rate which defines the number of objects which should be removed in each layer. This ratio can be a globally predefined one for all layers [LAT18, FC18, HLW+19]. It can also be customized for each layer. This can be done empirically or using some heuristics related to layer sensitivities [HPTD15, LKD+16, MTK+16, LWL17]. Authors in [BOFG20] claim that customized pruning rate for each layer performs better than the uniform one, although these traditional methods for determining the appropriate pruning ratio rely on human efforts. In another group of approaches, this process can be automated using AutoML-based methods such as feedback loop [YHC+18] and reinforcement learning [HLL+18], or MetaPruning [LMZ+19] which does not need any hyper-parameter tuning like reinforcement learning based ones [YGZL20].

- *Pruning and fine-tuning schedule* As mentioned in [HDL+20], pruning can be applied in a format of greedy or single-shot. In greedy manner, pruning and fine-tuning steps get repeated. Model can be pruned layer-wise or filter-wise followed by fine-tuning steps to compensate the accuracy loss [DDGH19]. In one-shot pruning, method prunes the network at once and then train it [HLW+19, LAT18].

In sum, pruning unlike quantization is commonly organized into a three stage pipeline that includes an evaluation of the importance of operands, selection of pruning tasks,

pruning and finally model fine-tuning. The core configurable parameters are pruning level, importance metric, pruning rate and pruning schedule.

2.2.3 Combined compression techniques

This section provides an overview of some combined optimization approaches. As previously mentioned, in this research, focus area is only on two of the compression techniques, pruning and quantization. These two approaches are the most common ones in practice [YGZL20]. The pruning techniques have been demonstrated to deliver outstanding results for a wide range of tasks, including image classification and object detection [XHCZ20]. Pruning by providing a thinner network is mostly affecting the structure, while quantization is reducing the bit-precision without any structural change. This reveals the orthogonality of these two approaches [WZZ+18, WWC+20] and provides a good opportunity for applying them jointly.

One of the early works in this area is *Deep Compression*, proposed by Han et al. in 2015 [HMD15]. As shown in Figure 2.23, it is a three-stage pipeline combining pruning, quantization and Huffman coding steps. The first compression method is weight pruning.

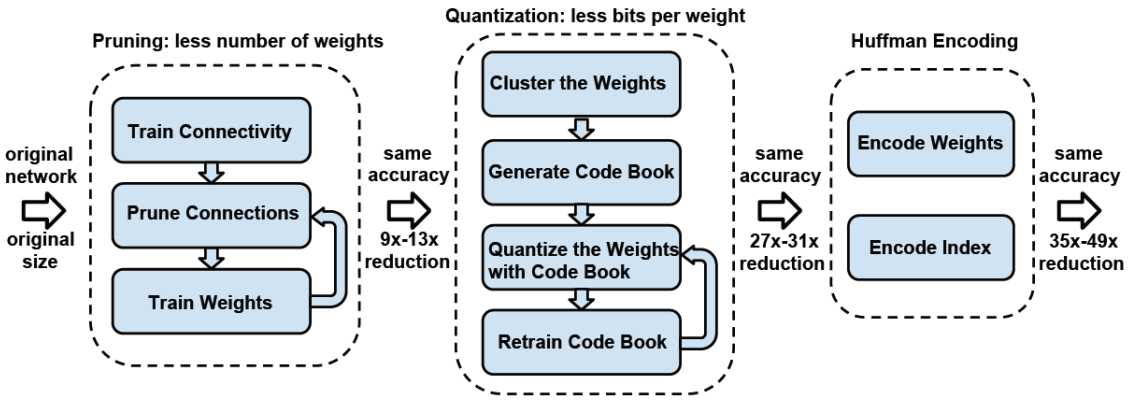


Figure 2.23: Deep compression pipeline proposed by [HMD15]

In this step a network is fully trained and then weights below a certain threshold turn to zero, resulting in a sparse network. The second compression method is quantization. A k-means clustering algorithm is deployed to find the centroids based on the number of defined bit-widths, and assign each weight to one of them. This mapping is presented in a *codebook* format, and during fine-tuning this codebook is getting updated as well. In the last step, Huffman encoding is responsible to reduce the count of required bits for weight representation in the quantized codebook. Figure 2.24 illustrates the one-shot pruning and quantization framework [HPZ+21] proposed in another recent article. They apply both compression techniques at the same time. Their proposed channel-wise unified quantization does not produce any extra overhead, unlike traditional solutions. In pruning technique, it applies a magnitude based weight pruning, which optimizes the pruning rate for each layer. At the end, a fine-tuning step updates the weights of the network by keeping the compression parameters fixed.

In other category of combined approaches, authors take benefit of the complementary features of these two compression techniques, instead of applying them in separate stages.

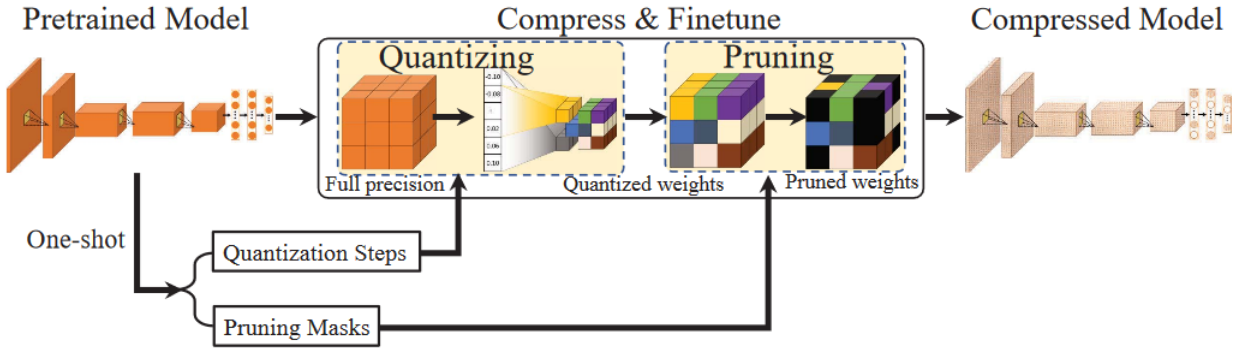


Figure 2.24: OPQ pipeline proposed by [HPZ+21]

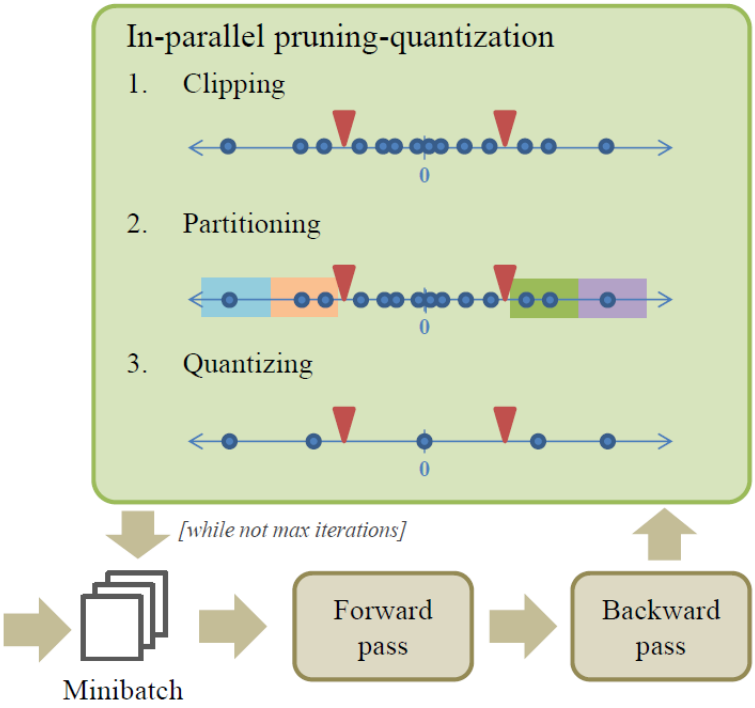
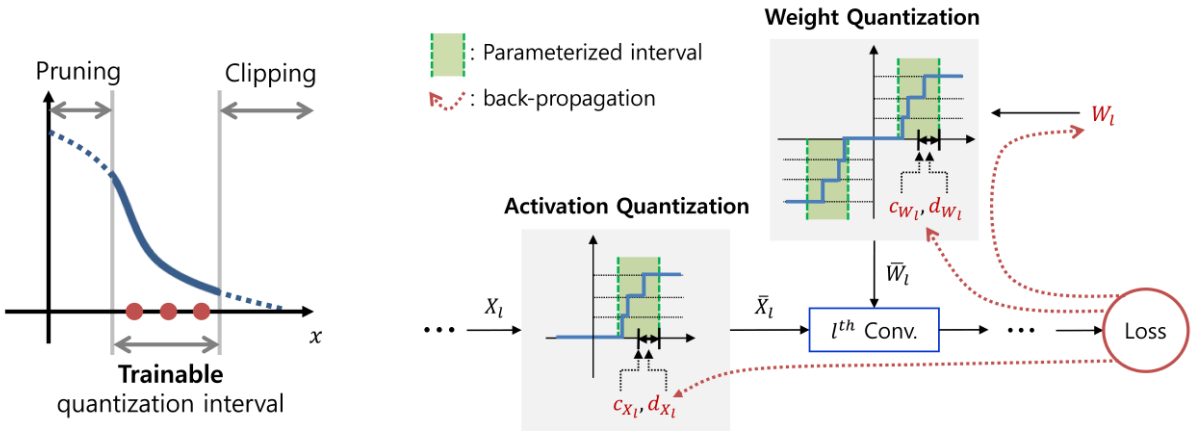


Figure 2.25: In-Parallel Pruning-Quantization proposed by [TM18]

In Clip-q [TM18], they perform pruning-quantization and fine-tuning in parallel on each layer through three steps (illustrated in Figure 2.25): in the first step, the *clipping* step, based on two learnable hyper-parameters (red triangles in step 1) and another static parameter, some of the full-precision weights turn to zero. Due to its adaptive characteristic during training, those removed connections could be restored in next iterations. In the second step, *partioning*, linear partitioning of b bit-width is being applied on remained weights. Colored segmentation in step 2 of Figure 2.25 represents those intervals. In the next step, *quantizatiom*, it averages the weights inside each interval to calculate the corresponding quantization level. Parameters in each step get updated after each training mini-batch [TM18].

In another work of this category, Jung et al. proposed the QIL method [JSL+19]. This method finds the optimal quantization interval for each layer of weights and activations. In this optimization problem, they train a quantizer with the aim of minimizing network



(a) Quantization Interval (b) A convolutional layer of our low bit-width network

Figure 2.26: Illustration of trainable quantizer (QIL) proposed by [JSL+19]

classification loss. As Figure 2.26 (a) shows, this trainable quantizer is based on two functions of pruning and clipping. Given a desired quantization bit-width, this method during the training finds the proper thresholds for these two opertaions. Clipping threshold is employed to restrict the upper bound of values while pruning threshold regulates the removal of small weights.

A limitation of the mentioned solutions is that they are not generalizable. Unstructured pruning despite of its high compression rate, results in a compressed sparse network that is subject to limitations. Weight pruning is producing sparse matrices which bring its extra memory overhead [ZG17] and speed-up of the operations in these matrices requires specific matrix operations [ZTZ+18]. Additionally, the main focus of QIL and Clip-q is quantization, and the pruning is primarily used for improving the results for such focus area. Other than the QIL, none of the other methods consider activation quantization. Furthermore, QAT methods require a sufficient number of epochs for fine-tuning in order to achieve satisfactory results. As you can see, further compression of the pruned models using quantization has left a research gap in combined or joint-way compression techniques. A well optimized solution for pruning a model is not necessarily going to result in optimal quantization of the pruned network. As a result, we focus on 8bit, uniform, and layer-wise PTQ of both weights and activations and filter pruning which leads to a structured compressed model. A magnitude based threshold pruning in the mentioned combined techniques already illustrates the opportunity to investigate the effect of pruning metrics on the resulting weight distribution and consequently on the quantization step.

Search-based compression combination

As we mentioned in earlier sections, the search area of compression techniques can grow exponentially and hyper-parameter tuning plays a crucial role in getting an optimal solution. This can also get worse when the optimization methods get combined. In recent work, researchers take the advantage of AutoML based techniques like automated quantization [WLL+19] and automated pruning [HLL+18] alongside Neural Architecture Search(NAS)

[ZL16, LSV+17, LZN+18] to solve this problem by framing it as a search-based optimization task.

The first approach proposed by Han et al. [HCZ+19] is to apply these automated techniques sequentially, as depicted in Figure 2.27. In the first stage, they design a specialized model for a provided hardware. To accelerate this step, instead of relying on naive reinforcement learning based NAS techniques, they apply Proxylessnas approach which considers the hardware characteristics as well [CZH18]. In the next stages they take the optimization further by applying automated channel pruning [HLL+18] and automated mixed-precision quantization [WLL+19].

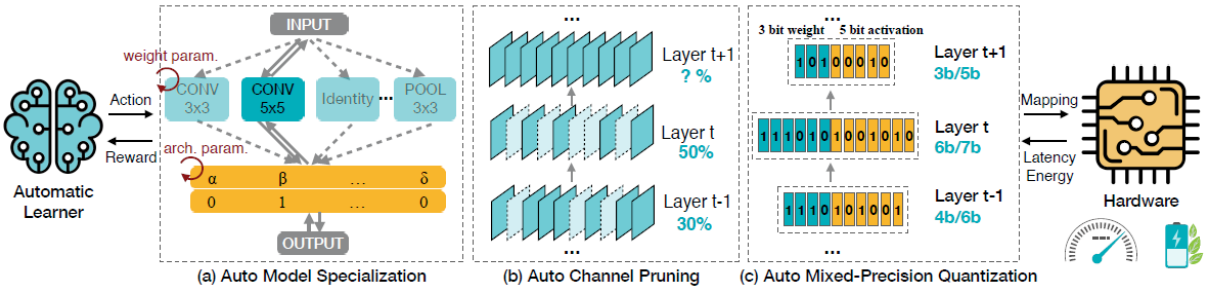


Figure 2.27: Multi-stage automated design by [HCZ+19]

In contrast to the previous work, authors in [WWC+20] believe that multi-stage approach usually leads to a sub-optimal solution and it also requires more time and energy for searching. For example their proposed approach for ResNet34 reduced the search time by 600× GPU hours with 2.3% accuracy boost. This superiority is a result of joint search for architecture and compression policies. For this purpose, they proposed a *predictor-based* method. The building blocks of this framework are presented in Figure 2.28. As we can see this framework also includes hardware feedback.

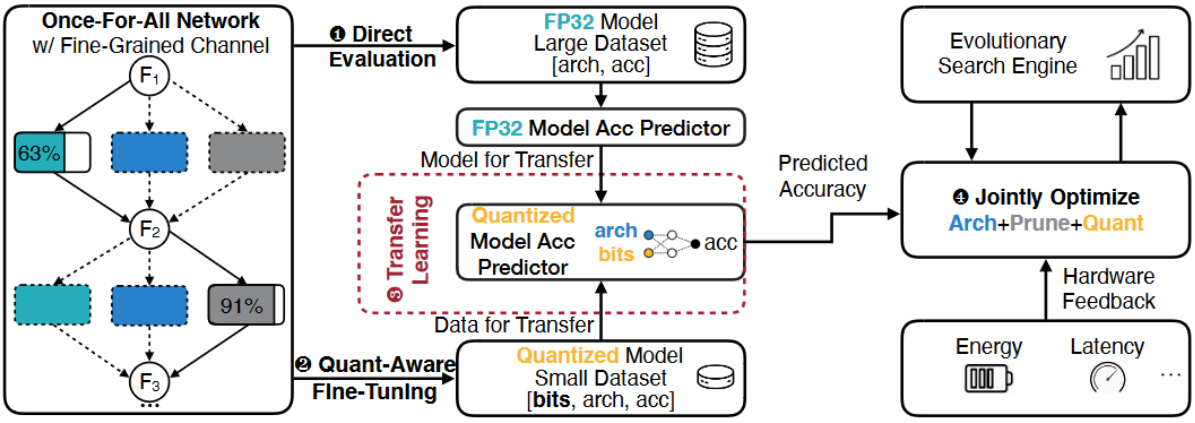


Figure 2.28: APQ proposed by [WWC+20]

2.3 Summary

In this chapter, we provide an introduction towards ANNs and then describe building blocks of CNN models in image classification task. We outline some of the benchmark

models including ResNet and MobileNetV2. Later we explore the background knowledge about CNN compression techniques and describe in details pruning and quantization approaches which are our point of interest. Inspired by joint-way compression, we provide an overview of combined pruning and quantization techniques.

3 Design

In Chapter 2, we presented the necessary background knowledge concerning the topics covered in this thesis. In this chapter we define the precise research questions addressed in this research, as well as the proposed design of the approach we adopt for the task of CNN model optimization. This chapter is structured as follows:

- We begin by establishing the research questions to be evaluated in our work (Section 3.1).
- In (Section 3.2), we present our proposed approach for optimization of a CNN model.
- Finally, we provide a summary of the main topics we discussed in this chapter in (Section 3.3)

3.1 Research Questions

The purpose of this research is to evaluate how well different compression algorithms perform on a CNN model optimization task when they are combined. Based on the literature review, as collected in previous chapters, we determined the following evaluation questions that we intend to address in this thesis:

1. Individual contributions from methods: What are the trade-offs regarding loss of accuracy vs. improvements in model compression ratio, for basic quantization and pruning algorithms, when individually applied.
2. Combined contributions from methods: How do the trade-offs between algorithms (regarding loss of accuracy vs. improvements in model compression ratio), change when applying them in combination?

3.2 Proposed Approach: Post-training Optimization

In order to address our research questions, we propose a workflow that supports CNN model optimization of any chosen pre-trained model on any dataset. As depicted in Figure 3.1, the input to this optimization process is a trained model. This model should be accompanied by a set of data, as our approach is not completely data-free. This trained model is passed through a set of compression techniques, which in our study correspond to quantization and pruning, as we discussed earlier (refer to Section 2.2). The output of this step is a compressed version of the model with a degree of accuracy loss, which can be used as a feature extractor for other tasks or exported to a target device. This was

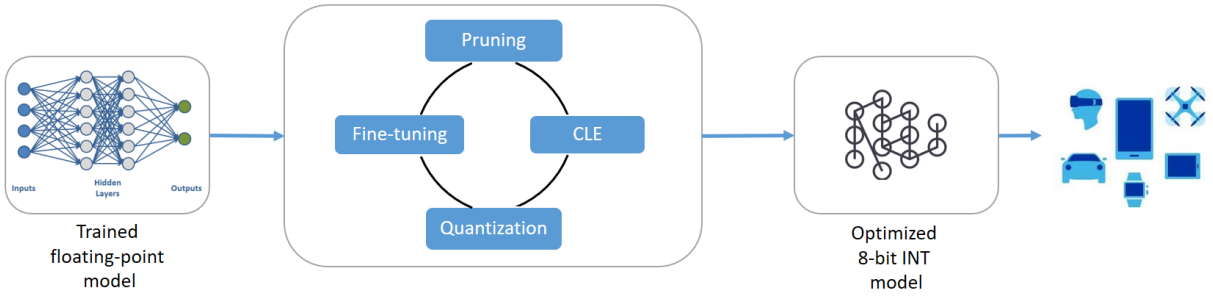


Figure 3.1: Framework high-level view

a high level overview of the approach, in the following sections we will discuss details of each compression step and their interrelation.

Our proposed methodology is a pipeline with two main steps of quantization and pruning. The quantization step can only happen after the pruning stage, as we are considering a post-training quantization. Figure 3.2 illustrates the flowchart of this methodology. In the first stage, an iterative filter pruning method is being applied on the trained model, for which details are provided in the next section. An optimization on this stage could be applying a CLE method before pruning. In some cases, this configuration results in a better performance for pruning. A prerequisite step to CLE is BN layers folding. This could be a limitation for some complex models, while they get over-fitted during pruning. The output of this stage is a compressed model with fewer parameters, for which the quantization stage can decrease the size of its parameters. The weights and activation of the layers get quantized separately, and this stage also provides optimization techniques which can be applied, based on the source of the produced noise in the model. Details are described later in Subsection 4.3.2. There is also the possibility of applying pruning after weight quantization (WQ), which we name it Zero Pruning (ZP). In some cases, WQ turns some weights to zero and at the end produces filters with zero magnitude. Zero pruning is applied to prune this filters away while it does not require any fine-tuning. The result of this stage is the final optimized model. In following sections, we explain the details in each stage.

3.2.1 Pruning

In this section, we will describe our pruning algorithm individually. To have a better understanding, first we will describe the terminology used for standard convolution of CNNs.

Let L_i denote the ith convolution layer where $i \in [1, 2, ..., N]$ and N is the total number of layers. The set of filters at layer L_i or *kernel matrix* can be represented as F_{L_i}, where $F_{L_i} = \{f_1, f_2, ..., f_{c_{i+1}}\}$ and each filter is of dimension (c_i, h_k, w_k). In these notations, c_i, c_{i+1} represent the number of input and output channels of the kernel matrix and h_k, w_k denoting its height and width of 2D kernels K (e.g., 3×3). This convolution layer (L_i) takes an input feature map, $X_i \in \mathbb{R}^{c_i \times h_i \times w_i}$ - h_i, w_i are the height and width of the feature

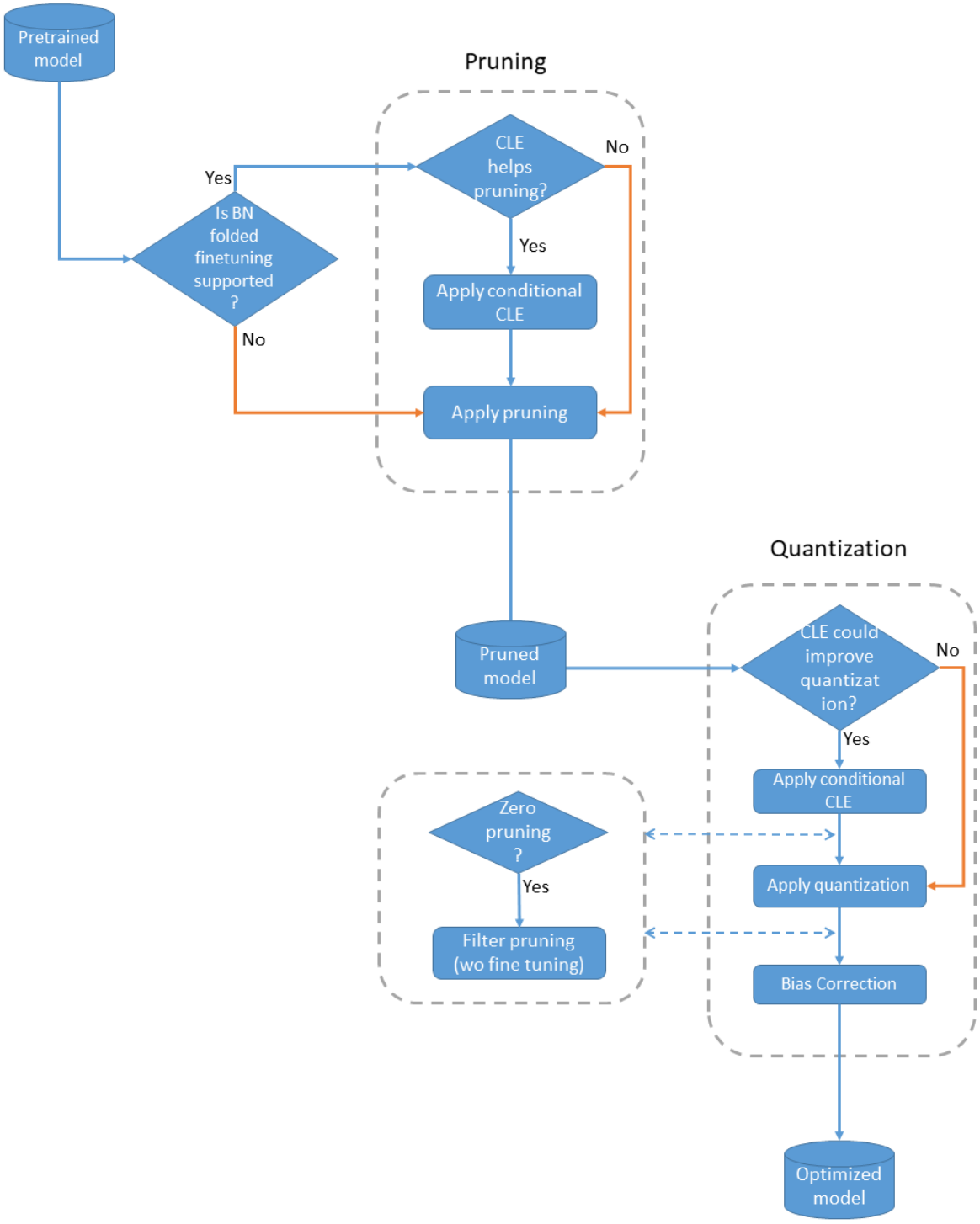


Figure 3.2: Detailed workflow of our approach

map - and transforms it into output feature map:

$$X^{(i+1)}_{c,:,:} = \sum_{j=1}^{c_i} X^{(i)}_{j,:,:} * F^{(i)}_{j,c,:,:} \quad , \quad c \in \{1...c_{i+1}\} \tag{3.1}$$

As illustrated in Figure 3.3, when a filter $f_c \in F_{L_i}$ where $c \in \{1, 2, ..., c_{i+1}\}$ is pruned, its corresponding feature map is also removed and the kernels from next convolution layer $F_{L_{i+1}}$ which operate on that feature map are also removed.

Now that we have a general understanding of how pruning a single convolution layer affects the other layers, we will give more details about the pruning procedure within a layer.

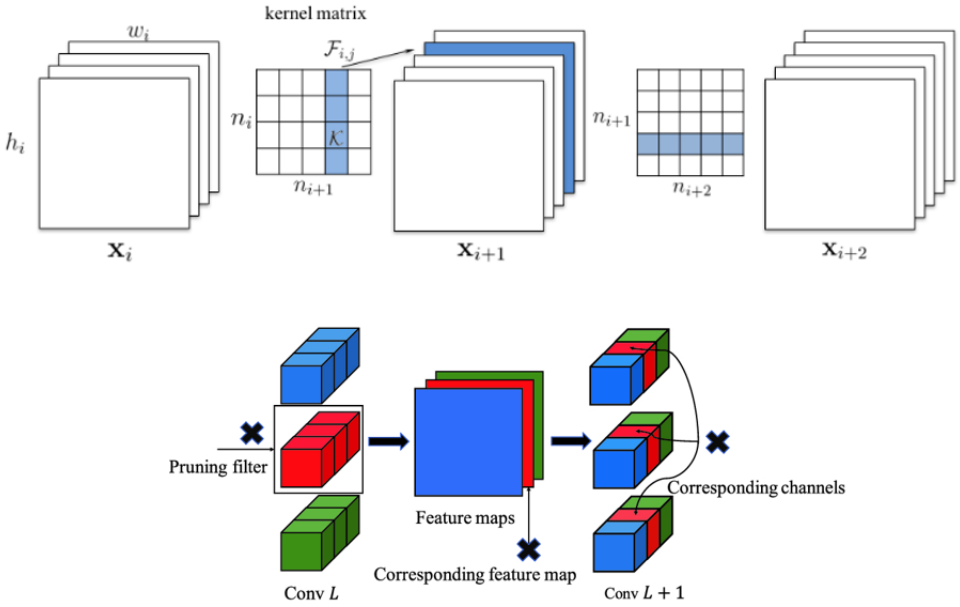


Figure 3.3: Pruning a filter results in removal of its corresponding feature map and related kernels in the next layer [LKD+16]

Our pruning method is an iterative layer-wise filter pruning which is mostly inspired by L1-Norm based filter pruning [LKD+16]. [LKD+16] is a type of predefined structured pruning which is based on some manual sensitivity analysis, for choosing the pruning rate for each filter. In our method, this sensitivity is included in the algorithm using two hyper-parameters, (α, ϵ). By using a global pruning rate α , the algorithm chooses the count of filters which should be pruned in each layer's pruning iteration. While based on ϵ which is the threshold for validation accuracy degradation, it makes a decision whether to continue the pruning of current layer or not. In another word, it checks the pruning tolerance for the layer. By combining these two parameters, a final local pruning rate for each layer can be achieved. Other two hyper-parameters which can affect the pruning performance are the dataset size and number of epochs used during fine-tuning. At the end of each layer pruning, fine-tuning step is happening to help the network to regain part of its accuracy loss. The overall filter pruning procedure for a set of prunable layers of a model (depicted in Figure 3.4) is as follows:

1. Before starting the pruning, it adds mask layers after the provided layers list, which should be pruned.

2. After choosing the first prunable layer, it selects the least important filters. Any importance metric discussed in the literature review (refer to Subsection 2.2.2) can be used for this section. Based on this metric a score will be calculated for each filter. Regarding the pruning rate hyper-parameter α and these scores, it selects the

final candidate list. α can be in the format of percentage (like 20% of the filters) or specific count.

3. Related indices for candidate filters in the mask layer will turn to zero.
4. After calculating the accuracy based on the validation set, if the degradation is less than the ϵ, then it will select the next set of candidates from the remaining filters.
5. After pruning one layer, the model is tuned using a train set provided for a certain number of epochs.
6. Following the removal of entire specified layers, masks are removed, and the model is reconstructed from the remaining filters.
7. At the end, fine-tuning of specific epochs can be applied.

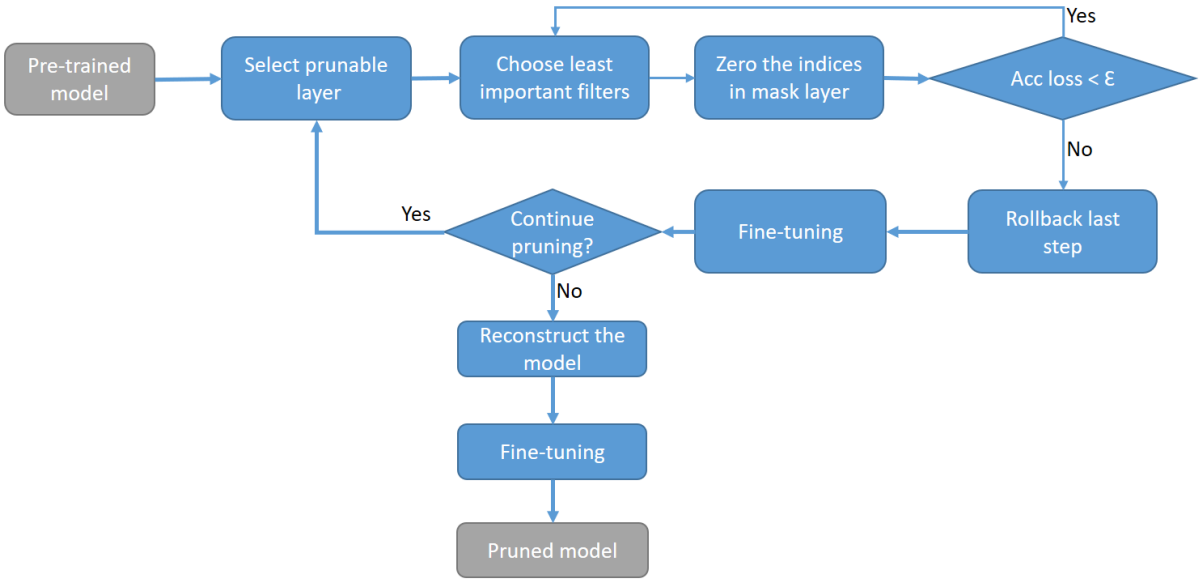


Figure 3.4: Flowchart of our pruning workflow

Pruning Strategy

The network architecture has also an effect on the pruning process. Pruning single-path CNNs such as VGGNet is straight forward without any limitations. Although there are many restrictions for pruning special building blocks of multi-branched networks. In ResNet networks, for example, due to the spatial dimension compatibility, the output channel number of each layer of each block must be consistent in order to finish the sum operation. This issue is illustrated in Figure 3.5. Because of these constraints, the final layer of the residual block needs to be skipped along with the projection shortcuts(refer to Section 2.1.3).

As mentioned in Section 2.1.3, MobileNet-V2 consists of stacked bottleneck blocks. Each block starts with an expansion layer E_i which contains 1×1 point wise convolution followed by a depth-wise convolution D_i and the third layer is another point wise convolution layer P_i. Figure 3.6 illustrates the straight forward way of structured pruning for this network. Pruning redundant filters of E_i causes redundant channel reduction in the second layer(D_i)

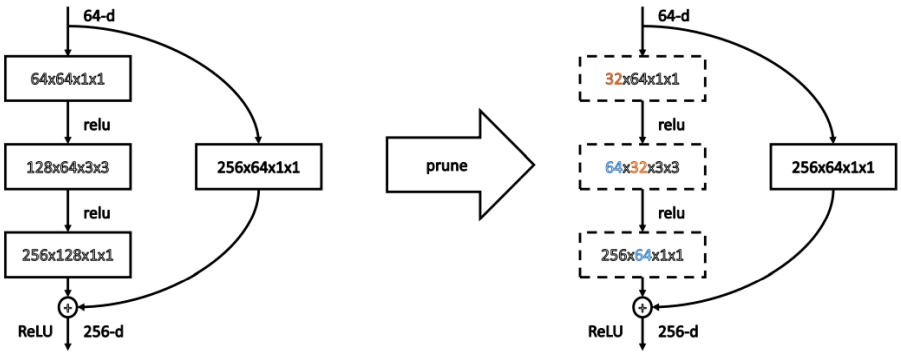


Figure 3.5: Pruning strategy for residual block by [WZZ+18], we skip pruning the last layer in each block.

and third layer (P_i) in the same block. In case of pruning the third layer, channels in the first layer of next block (P_{i+1}) should also be pruned away. Also, if there is any parallel shortcut in the structure, it will get affected as well. In our research, we only apply the first type of pruning.

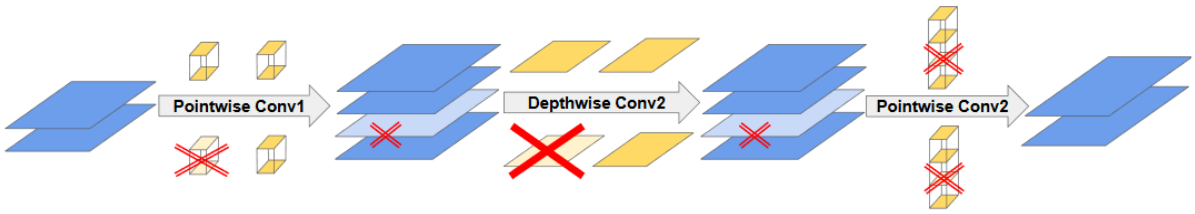


Figure 3.6: Pruning strategy for depthwise separable convolutions [TLCC20].

3.2.2 Quantization

In our quantization workflow, we follow a similar approach like the one proposed in [NBBW19]:

- First step is folding the batch norm layers in their preceding Conv layers.
- Next, we apply an 8-bit filter-wise WQ.
- Then we check the accuracy of the model. If the performance degradation is high, we check the possible sources of noise. In this methodology, we apply CLE [NBBW19] and IBC methods to decrease the noise caused by imbalanced filter ranges and MAS [FAG19] respectively (refer to Section 2.2.1 for more details). CLE should be applied before WQ and IBC should be applied after WQ and before Activation Quantization (AQ). If it did not improve the performance, we need an extra layer-wise noise analysis.
- Next step is 8-bit AQ. Since the AQ is more challenging, we apply different techniques to find appropriate step size and among them choose the best performance. If the final accuacy loss is not acceptable, a layer-wise quantization and noise analysis are necessary to find problematic layers.

3.2.3 Evaluation Criteria

Following are key metrics we use in evaluating experimental designs:

- *Top 1 accuracy* It scores the portion of correctly predicted labels. The class with the highest probability is the predicted label and if it corresponds to a target label, it is a correct prediction.

- *Relative error* It evaluates the change in accuracy with respect to the original model. This metric is beneficial when we comparing the efficiency of a method applied on different base models. The following formula is used to calculate it. The error is the accuracy difference between the original model and the resulting model:

$$error_{relative} = \frac{|error|}{100 - Acc_{original}}$$

- *Compression ratio* This metric is used to measure the model compression quality [CWZZ17]. Assuming that p represents the parameters of the main model M and p_c is the same for the compressed version of the model M_c. Then the compression rate α is calculated as follows:

$$\alpha\left(M, M_c\right) = \frac{p}{p_c}$$

- *Pruned(%)* This metric represents the percentage of the parameters in a pruned model which have been removed:

$$(1 - \frac{p_c}{p}) \times 100$$

- *FLOP* Floating-point operations is used for CNN models complexity evaluation. Based on [MTK+16], FLOPs for a FC layer with input I and output O are calculated by:

$$FLOPs = (2I - 1)O$$

  And for Conv layers, FLOPs are calculated using:

$$FLOPs = 2HW\left(C_{in}K^2 + 1\right)C_{out}$$

  where H, W are the height and width of the feature maps, C_{in}, C_{out} refer to the number of input and output channels, K is the kernel size, and 1 stands for the FLOPs in bias term.

3.3 Summary

In this chapter, we present our thesis's design, which is a general approach for combined way compression. We finalize our research questions formulated to be answered as part of this thesis. Additionally, we broke our design into stages of pruning and quantization. We

describe details of each stage. Finally, we conclude this chapter by providing evaluation metrics we use in this thesis.

4 Experimental Setup

We provide in this chapter all the information necessary to reproduce the evaluation results of this research:

- Section 4.1 is devoted to the description of experimental datasets.
- The Section 4.2 outlines the models used in the experiments.
- In Section 4.3 we provide implementation details for different steps of the methodology as well as selected hyper-parameters.
- Finally, in Section 4.4 we provide a detailed specification of the hardware and software used.

4.1 Datasets

Through our experiments, we examined a number of different network architectures on the CIFAR-10 [KH+09] and ImageNet (ILSVRC-2012) [RDS+15] datasets for image classification. We present these datasets shortly in this section.

4.1.1 Canadian Institute For Advanced Research Dataset (CIFAR-10)

CIFAR-10 [KH+09] is a small size dataset established by Alex Krizhevsky. It consists of 60,000 color images, of which training and testing sets have 50K and 10K data samples respectively. Figure 4.1 shows that these images belong to one of the 10 classes and have resolution of 32×32.

4.1.2 ImageNet Dataset

There are thousands of photos in the ImageNet dataset that have been hand-annotated. This dataset was released in 2009 [DDS+09] and since then it has continued to be used in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [RDS+15], which is an annual competition to benchmark the algorithms. For this work, we refer to dataset ImageNet used in the ILSVRC-2012. It contains more than 1.2 million color images in 1000 categories with resolution of 224×224. We use the validation set including 50K images with batch size of 64 for network evaluation while test set samples are not labeled. Figure 4.2 shows some examples of ImageNet images.

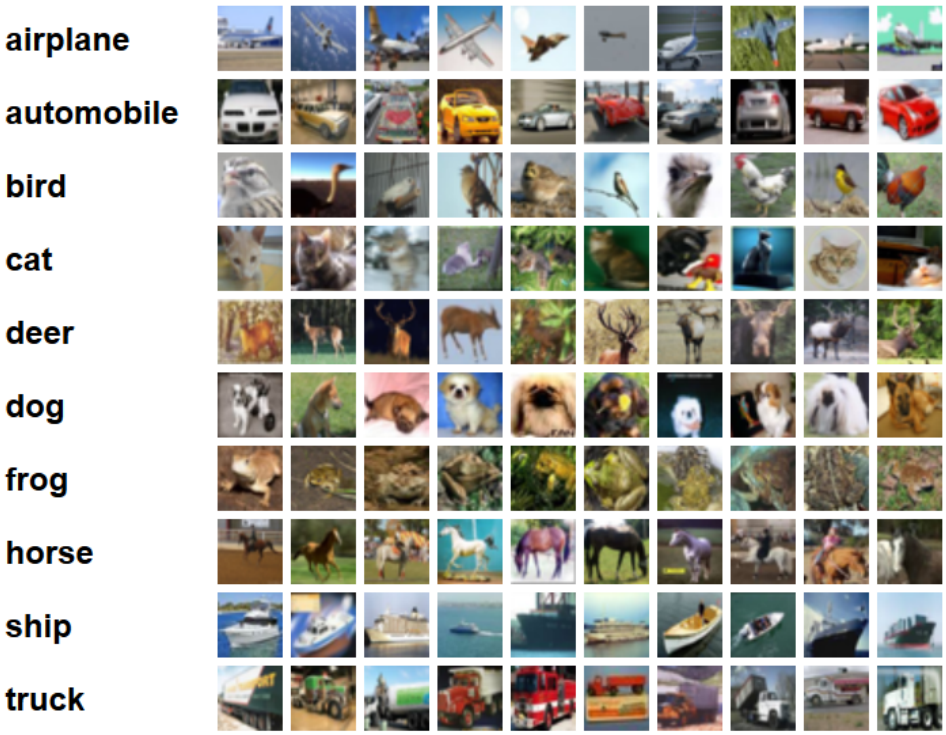


Figure 4.1: Sample images from the CIFAR-10 dataset and their correct classifications [KH+09]. The resolution of the images is 32×32.

4.2 Models

GoogleNet, AlexNet, and VGG are all over-parameterized networks. As a result, it is easy to obtain significant performance improvements by narrowing the parameters of these architectures [YSX+19]. We choose to examine architectures using efficient parameters, such as MobileNet and ResNet. We have already explained the building blocks and architecture details of these networks in Subsection 2.1.3. In the following subsections, we will discuss the preparation of the trained models. In all models, we choose the architectures with BN layers after the convolutional layers, which is compatible with BN layer folding.

4.2.1 ResNet

- **ResNet on CIFAR-10:**
  We trained the ResNet-32 and ResNet-56 models from scratch using the code provided in PacktPublishing GitHub repository[1]. Training the models with the provided training hyper-parameters, we reached accuracy of 92.53% and 92.30% for ResNet-32 and ResNet-56, respectively.
- **ResNet on ImageNet:**
  In this work, we use pre-trained ResNet50-V1 model provided by the Keras library. This model achieves up to 74.99% accuracy on ImageNet, with almost 25 million

[1] https://github.com/PacktPublishing/Advanced-Deep-Learning-with-Keras/blob/master/chapter2-deep-networks/resnet-cifar10-2.2.1.py

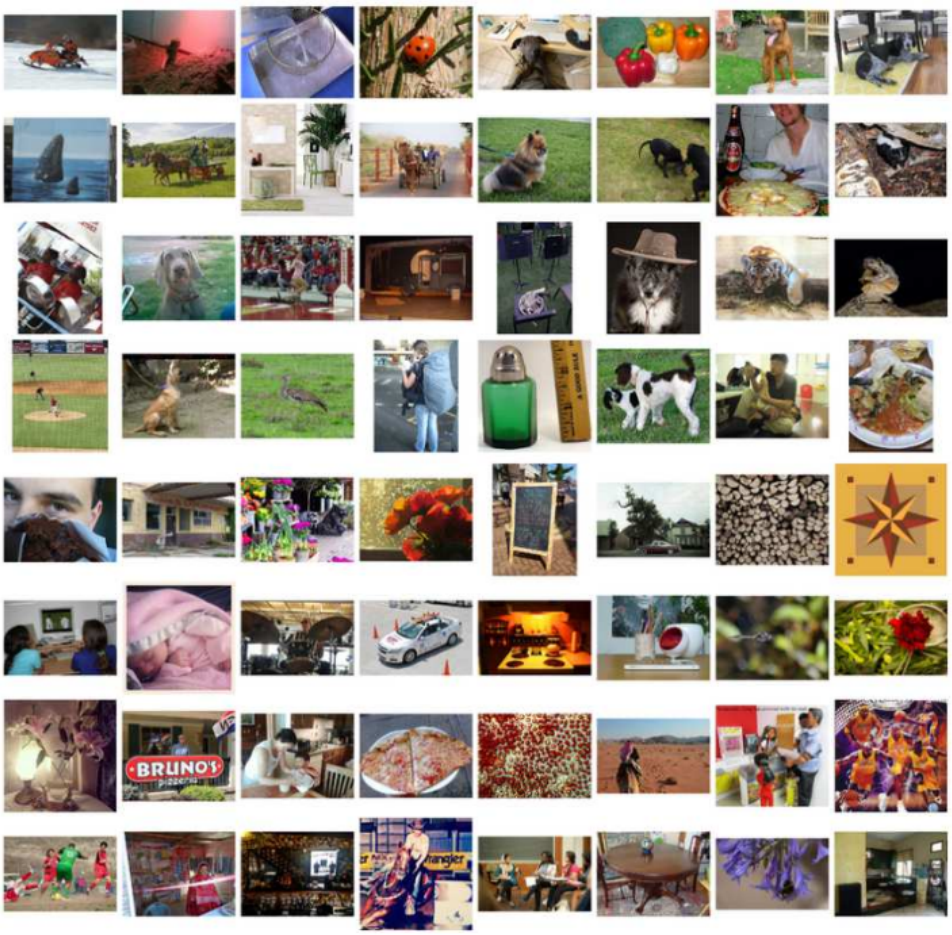


Figure 4.2: Sample images from ImageNet dataset [CRK16].

parameters. For fine-tuning and validation, we use the same data pre-processing function that is compatible with the ResNet50 provided by Keras.

4.2.2 MobileNet

- **MobileNet on CIFAR-10:**
  We trained this model from scratch using the code provided in yumaloop GitHub repository[2]. It reaches 90.60% top-1 accuracy on Cifar-10 with almost 2 million parameters.

- **MobileNet on ImageNet:**
  In our experiments we use the pre-trained MobileNet-V2 model provided by Keras library. With almost 3.5 million parameters, this model is accurate to 71.72%. The same data pre-processing step provided by Keras is used during fine-tuning and validation.

[2] https://github.com/yumaloop/mobilenetV2-cifar

4.3 Setup

We provide implementation details for different steps of the methodology, as well as selected hyper-parameters. The repository URLs are provided when we have just re-implemented methods based on our existing framework. As mentioned previously, due to practical hardware limitations, only uniform, single precision quantization and structured pruning are considered.

4.3.1 Pruning

In order to begin this process, the model should be prepared for pruning. We do this by adding binary mask layers, with each layer's output channel the same length after prunable layers. After finishing the process, a new model will be constructed based on the mask data, and those mask layers will be removed. As explained in Subsection 3.2.1, in our layer-wise and iterative pruning scheme, for each layer we rank the filters based on an importance or pruning metric. In our experiments, we utilize two types of magnitude-based and relation-based metrics to calculate the scores of filters in each layer. In the following paragraphs, we describe the implementation details of them:

- **L1-Norm:**
  This criteria considers the magnitude of the filters in a kernel as an importance metric. This can be calculated by the sum of the absolute weights of a filter or its L1-norm, $\|f\|_1$. Filters with smaller magnitude have less importance and can be pruned [LKD+16].

- **Geometric Median:**
  Based on this criteria, a similarity matrix (D) of filters in a layer is calculated. The distance between filters can be measured based on euclidean distance (L1 or L2-norm) or cosine similarity. To find the relation between filter f and the other filters, we take the summation of those distances:

$$SumD(f) = \sum_{i=1}^{c_{i+1}} D(f, f_i) \tag{4.1}$$

  Filters with smaller SumD have a higher similarity to their neighbours and can be replaced by them [HLW+19]. In our experiments, we use euclidean distance(L1-norm) to measure the distance.

After ranking the filters, there are the following hyper-parameters that define the final filter indices selection:

- **Pruning rate (α)**: It defines the number of filters which should be pruned away in each filter. It can be in a format of percentage or count. It is set globally (one pruning rate for all layers) while using the next hyper-parameter it can be customized based on each layer's tolerance.

- **Accuracy degradation threshold (ϵ)**: After selecting the candidate filter indices based on pruning rate, we set the mask index of them to zero and calculate the intermediate accuracy of the model. If the accuracy difference between this model

and the base model is smaller than a provided threshold (ϵ) then we select those indices for pruning.

- **Fine-tuning hyper-parameters**: Fine-tuning can happen during pruning and after pruning.

  - **During pruning**: After pruning each layer we apply a tuning step to regain the decreased accuracy. The number of epochs and training dataset size can affect the performance. In our experiments, we use the whole training set unless another setting has been mentioned. We only apply one epoch of tuning.

  - **After pruning**: There are cases which our pruned models need extra fine-tuning to improve their performance. For each experiment if this step is applied, we have provided the supplementary details.

It is possible that in some evaluations based on the experiment complexity and time constraint, we only utilize one of the pruning hyper-parameters (α,ϵ). More details related to pruning experiment setup is provided at the beginning of each evaluation part.

4.3.2 Quantization

In this work, we apply quantization methods which are supported by most of the existing hardware. We focus on a layer-wise, uniform, and symmetric 8-bit quantization of weights and activations. While in WQ, it only includes weight tensor and biases remain in 32-bit. This heterogeneous approach is also followed by [JKC+18, VSM11]. To calculate the step size for AQ, we need a calibration set. In our experiments, it is a sample set of 64 images unless another number is reported.

In this section, we review a symmetric uniform quantization scheme. Uniform quantization is a linear mapping of a high-precision real value $x \in \mathbb{R}$ into a low-bit precision signed integer value. The uniform quantization operation is defined by Equation 4.2 and Equation 4.3 [VSGA19]:

$$clamp(x, a, b) = \begin{cases} b, & x \geqslant b \\ a, & x \leqslant a \\ x, & Otherwise \end{cases} \tag{4.2}$$

$$Q_k(x, \Delta) = clamp(round(\frac{x}{\Delta}), -2^{k-1}, 2^{k-1} - 1) \tag{4.3}$$

where k denotes the desired bit-width and Δ is the step size. Round(.) is a rounding operation which maps the real value to an integer value which is the nearest one to real value. In this work, we use the following methods to calculate the quantization step size Δ:

- Maximum Absolute Value : This approach is used in some works to find the weight quantization step size [GMG16, JKC+18]. For a given real valued data distribution of X, step size is calculated by:

$$\Delta = \frac{max(|x_{min}|, |x_{max}|)}{2^{(k-1)} - 1} \tag{4.4}$$

  This method has also been used for step size calculation in feature maps which needs a calibration batch to calculate the step size [GPMG18]. Placing this value in Equation 4.3, each x value is proportional to the $max(|X|)$ and in case of outliers, this approach may include strong noise [VSGA19, KMGB21].

- Minimum Mean Squared Error (MSE): Converting the high precision NN into its low precision discrete value version results in a quantization noise (error). Since in PTQ there is no training possibilty to compensate this performance degradation, setting an optimal quantization step size can be beneficial. This optimization problem can be solved by minimizing MSE [VSGA19, GKD+21] as follows:

$$\Delta = \underset{\Delta}{argmin}(\|X - X_q\|_2) \tag{4.5}$$

  We use this method to quantize the output of the layers. In another word, we are trying to minimize the L2-Norm of the difference between main output and its quantized version by adjusting step sizes.

- Minimum Propagated Quantization Error: This method is proposed by Vogel et al. [VSGA19]. In addition to the tensors weight distribution, it also incorporates the effect of quantization noise on the structure by propagating it through the network.

$$\Delta = \underset{\Delta}{argmin}(\left\| \underbrace{\hat{y}_k - y_k}_{propQE} \right\|_2) \tag{4.6}$$

  where y is the output of the kth layer and $\hat{y}$ denotes its quantized version. This method has larger computational overhead in comparison to the previous method. However, it provides higher accuracy [DlPGK20].

4.3.3 Cross layer Equalization

We re-implemented the Keras version of cross layer weight equalization algorithm introduced by [NBBW19] mentioned in Section 2.2.1. We took the same hyper-parameters for the main loop iteration from the function named *cross_layer_equalization* from the DFQ repository[3]. Our code is also inspired by another implementation from Aimet optimization repository[4]. Based on our experiments, we found out that there are cases which after applying CLE all the weights of the layer will turn to zero. For this purpose, we applied a conditional version of it that is checking the mean and variance of the weight distribution.

[3] https://github.com/jakc4103/DFQ
[4] https://github.com/quic/aimet

4.3.4 Bias Correction

As mentioned previously, Section 2.2.1, IBC is another quantization optimization method. We re-implemented the same algorithm provided by Finkelstein et al. [FAG19]. The only difference is that we are using the pre-activations output of the layer instead of the activation. We are also using an unlabeled dataset size of 512.

4.4 Experimental Environment

In this section, we overview the essential details about the experiment setup we deploy, running the experiments of this thesis. We employ the following configurations:

Machine Configuration

- **Operating System** Ubuntu 18.04.5 LTS
- **Processor** 2× Intel(R) Xeon(R) CPU E5-2680 v4, 2.4 GHz
- **Memory** 512 GB RAM
- **Graphics** GeForce GTX 1080 Ti

Programming Framework

- **Programming Languages** Python (Version 3.7.2)
- **Programming Tools** PyCharm community edition (Version 2018.3.3), Jupyter Notebook (Version 6.0.3)
- **Libraries** Scikit-learn (Version 0.22.2), Numpy (Version 1.18.1), Pandas (Version 1.0.3), Tensorflow (Version 1.14.0), Matplotlib (Version 3.2.1)

4.5 Summary

In this chapter, we provided an overview of benchmark datasets and CNN architectures which we evaluate in our research. Furthermore, we provided essential details for the optimization methods. In the final part of this chapter, we explain how we conducted our experiments, including hardware and software details, so that the results can be reliably reproduced in the future.

5 Evaluation and Results - Part1

This chapter discusses the results of the experiments we conducted in response to the first research question in (Section 3.1). The following sections are included in this chapter:

1. In Section 5.1, we discuss the effect of pruning techniques on model optimization.
2. Section 5.2 is dedicated to examining the performance of quantized models.
3. In Section 5.3, we summarize some basic facts and points of interest on this evaluation set.

5.1 RQ1: Performance of Pruned model

In this section, we evaluate and present the performance results of the models which are pruned and quantized in two separate stages.

In pruning section, we present the experiments in two categories organized by the choice of dataset and model architecture. We selected Cifar-10 as a small dataset and three models (ResNet-32,56 and MobileNetV2) are examined based on it. Also we use ImageNet as a big dataset and two networks (MobileNetV2 and ResNet-50) are evaluated using it. First we will explain in detail the pruning process for ResNet-32 network as an instance and then present the experiment results for other networks.

5.1.1 Case Study 1: Small Dataset (CIFAR-10)

Network : ResNet

- **Experiment Setup:**
  As previously explained (Subsection 3.2.1), the pruning ratio(α) and model accuracy degradation threshold(ϵ) are two main hyper-parameters of our pruning algorithm. We chose pruning rates of (10,20,30,40,50)% and ran experiments for sets of thresholds (0.01,0.02,0.03,0.04,0.05). In order to better understand the influence of those parameters, we present Table 5.1 as a part of the results of pruning ResNet-32 with different settings. We repeated the experiments using the same hyper-parameter sets for the ResNet-56 network. The applied pruning metric is L1-Norm based. The algorithm just prunes the first layer of each residual block within each stage by using the pruning strategy mentioned in Section 3.2.1. We tuned the model after pruning each layer just for one epoch and used the whole validation dataset for this purpose. The optimizer used during tuning is ADAM with a learning rate of 10^{-5}.

As a matter of simplicity, to be able to use the resulting model for the next step of the experiments we folded the BN layers at the beginning.

- **Expected Results:**
  We expect to see more pruning opportunities for higher thresholds and lower pruning rate sets. Also we expect to observe more compression rates in ResNet-56 rather than ResNet-32. As ResNet models get deeper, there is more possibility to include redundancy.

- **Results:**
  The first look at the results in Table 5.1 reveals that for a fixed pruning rate (α), increasing the accuracy degradation threshold (ϵ) always acts in favour of more parameter reduction, which results in more compression rate, although with an understandable cost in terms of accuracy loss. It is observable that increasing (α,ϵ) at the same time does not result in most efficient networks. As an instance, experiment R32-5.3(30,0.03) acts worse than R32-1.1(10,0.01) which has a smallest (α,ϵ). We can see that for $\epsilon = 0.01$ the best compression rate and speed up belongs to the $\alpha = 10\%$.

  These experiments can be considered as establishing a search space for solving a multi-objective optimization problem. They are plotted based on compression objectives (i.e., compression rate and accuracy), as shown in Figure 5.2. In our case we want to maximize the compression rate while minimizing any degradation in the model accuracy. Figure 5.2 shows how the best solutions are fitted in a curve (blue line). This curve is called the Pareto frontier. This Pareto front just guarantees an optimal pruning configuration within the search space, with respect to the trade-off between optimization objectives. Low pruning rates (10%,20%) have the most members in the Pareto frontier. When the pruning rate is getting higher, lower thresholds provide more optimal solutions ((30%,0.01),(40%,0.01)). As you can see, all the models with pruning rate of 50% are dominated by better solutions.

- **Analysis:**
  Influence of Hyper-parameters: In order to examine the impact of these hyper-parameters on layer-wise pruning in ResNet-32, we will first review the structure of the model and then provide percentages of the filters pruned in each layer of each experiment in Table 5.1.

  As depicted in Figure 5.1, the ResNet-32 network architecture consist of three stages where each of them has a 5× repetition of the mentioned residual blocks. The first blocks of stage 2 and 3 have a projection shortcut (dotted line) to handle the FM size changes (for more details refer to Section 2.1.3). With a growing depth of the network, the number of filters is increasing (16,31,64), while FM size is decreasing (32,16,8).

  The pruning rates for each layer of each experiment set are shown using a heatmap in Figure 5.3. We can see that the global pruning rate (α) is just a minimum starting point, and there are layers which can be pruned beyond it or even never get pruned (like layers 12 and 23 for $\alpha > 0.1$). This shows that different layers have different sensitivity towards pruning and different selection of (α,ϵ) can help towards a degree of adaptation. Increasing ϵ can influence layers pruning differently. In spite of the fact that increasing ϵ for a fixed α improves the final compression rate, it does not

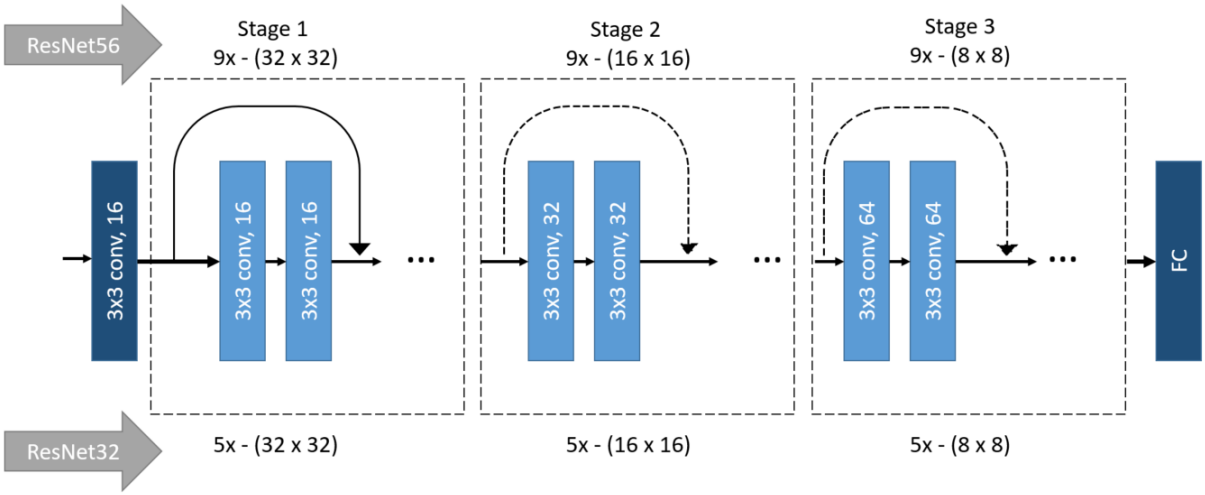


Figure 5.1: ResNet Architecture

mean that each layer is getting more pruned. As an example, for experiment set with $\alpha = 0.4$ increasing threshold from 0.02 to 0.03 does not increase pruning rates in the first stage except from layer 4. Also in stage 2 the pruning rate for layer 17 gets doubled while this stopped layer 21 from being pruned. This means that the order of the layers and their pruning do influence the next layers pruning behaviour.

For a higher pruning rate ($\alpha > 0.1$) there are common layers which can not be pruned (layers 12, 23, 26, and 28). Even increasing the ϵ may not solve this issue. For these layers lowering the pruning rate can be beneficial. For the first set of experiments (α= 0.1) with 0.01 and 0.02 tolerances, the model manages to prune 12.5% of the filters in layer 21 and 10% of filters in layers 26 and 28, although it is still unable to prune layer 23. The most sensitive layers are 12 and 23 which are located in the first block of each stage where the number of FMs changes. This point is also consistent with the results obtained in [LKD+16]. Clearly, there are also differences in pruning sensitivity between stages. Layers in the first stage (layers 2-10) can be pruned more than the other layers. Also the second stage layers (12-21) can be pruned more than the third stage. This result supports the notion that layers in later stages are more sensitive to pruning than early stages. This result is also consistent with the conclusions in [LKD+16].

Influence of Network Structure: ResNet-56 is also behaving in the same manner as ResNet-32. We can see a similar sensitivity pattern, which is partly explained by the fact that the only difference between these two models is the number of repetitions of each block inside each stage. When comparing the provided pruned models for ResNet-56 in Table 5.1 with the same counterparts of ResNet-32, we can see that ResNet56 can be pruned more. As an instance in R56-4.1 parameters the reduction is twice the reduction in R32-4.1. This is consistent with our expectation of finding more redundancy in deeper ResNet models. However, the greedy way of pruning with hyper-parameters of (0.1,0.01) does not have the same behaviour as ResNet-32. The model only gets an accuracy degaradation of 0.74% for only 1.08 compression rate. We observed that model the stopped pruning in layer 31 as it was not meeting the threshold. This shows that with greedy pruning at shallow layers the accumulated error gets high and it prevents the model from further pruning

in deeper layers. By increasing the threshold to 0.02, the model reached a 3.14 compression rate. This indicates that adapting the threshold setting for different stages in deeper ResNet models can also be beneficial.

Influence of Channel Equalization : In previous work, authors have examined the effect of channel equalization on quantization improvements [NBBW19, MFAG19]. Our analysis for this section applied CLE to the ResNet-32,56 models before applying the pruning step to see whether they correlated. The observed results are presented in Table 5.2. After applying CLE to the ResNet-56 network layers, we also observed a point that was not mentioned in previous work: there are two layers (50,51) on which all weights are small enough (10^{-32}) and applying channel equalization turns them all to zero. That gives us the idea of applying conditional CLE, which skips layers under certain conditions. Although these layers can be removed by a pruning mechanism which supports architecture change and results in a shallower network.

Most of the other layers have small weights which, after channel balancing, turn to zero. The zero weights cause some filters to have zero magnitudes. This allows for the possibility of pruning the filters without fine tuning which we refer to it as ZP. Using ZP on the channel equalized version of the ResNet-32,65 leads to compression rates of 1.46 and 2.24 respectively, without any accuracy drop. Additional pruning can be done using the resulting models as a compact base model. As recorded in Table 5.2, Res32-CLE-ZP-1.1 gets 7.2% more parameter reduction with 0.56 accuracy drop, which even acts better than most of the models on the Pareto front in Figure 5.2. Likewise, R56-CLE-1.1 gets 2.06% more parameter reduction with slightly more accuracy loss (0.14%). For the same reason, mentioned earlier, higher global pruning rate does not necessarily result in better pruning performance. As we can see in Table 5.2, models with 0.4 pruning rate end up acting worse than for the previous setting.

- **Takeaways:**
  In order to achieve the best pruning performance, hyper-parameter tuning is crucial. This tuning is also affected by the model's architecture. Different layers and depths of the network react differently during the pruning process. Additionally, models trained in different settings can behave differently. By modifying the weight distribution of the base model using CLE, we ended up with a new idea of pruning a model without tuning (ZP).

Network : MobileNet

In the next experiment, we examine the MobileNetV2 model pruning that has been trained on Cifar-10.

- **Experiment Setup:**
  For this part, the pruning of the filters is count-based. In each iteration one filter will be pruned at a time. Consequently, we have just one hyper-parameter (ϵ) which affects the pruning process. We examined a set of 3 thresholds (0.01,0.02,0.03). In addition to L1-Norm which is a magnitude-based pruning metric, we also considered a relation-based pruning criteria, GM. As explained before, only first layers of

experiment No.	Pruning rate(%)	Acc. Threshold	Acc. drop	Pruned(%)	Compression rate	FLOPs drop(%)
R32-1.1	10	0.01	0.36	29.72	1.42	31.54
R32-1.2	10	0.02	1.41	33.19	1.50	42.59
R32-2.1	20	0.01	0.63	29.47	1.42	30.48
R32-2.2	20	0.02	1.22	34.55	1.53	40.15
R32-3.1	30	0.01	0.22	22.60	1.29	27.61
R32-3.2	30	0.02	1.03	24.46	1.32	33.99
R32-3.3	30	0.03	1.29	31.27	1.46	42.49
R32-4.1	40	0.01	0.33	24.71	1.33	28.68
R32-4.2	40	0.02	0.9	29.54	1.42	36.96
R32-4.3	40	0.03	1.67	29.97	1.43	39.94
R32-5.1	50	0.01	0.34	11.89	1.13	20.40
R32-5.2	50	0.02	0.59	16.35	1.20	30.60
R32-5.3	50	0.03	1.15	26.47	1.36	37.39
R56-1.1	10	0.01	0.74	7.85	1.08	29.34
R56-1.2	10	0.02	1.38	68.20	3.14	67.39
R56-4.1	40	0.01	0.33	55.06	2.22	52.77

Table 5.1: Performance of models ResNet32-56 pruned with different set of hyper-parameters.

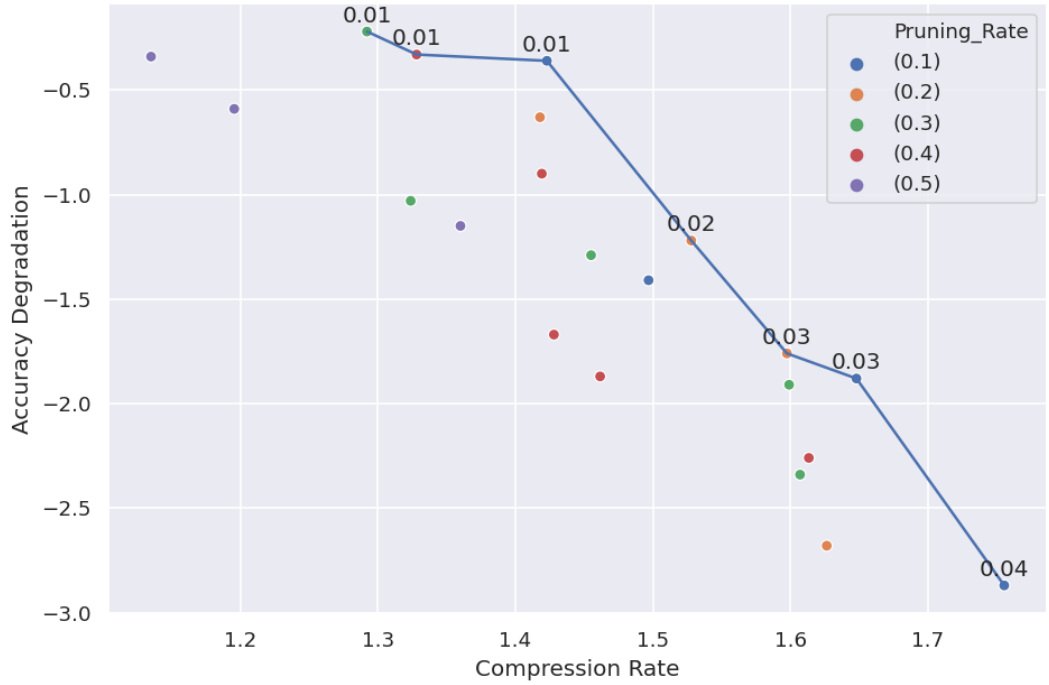


Figure 5.2: Different ResNet-32 pruned models. The blue line represents the Pareto frontier containing the best models in the search space. The numbers above each model show the accuracy degradation threshold parameter.

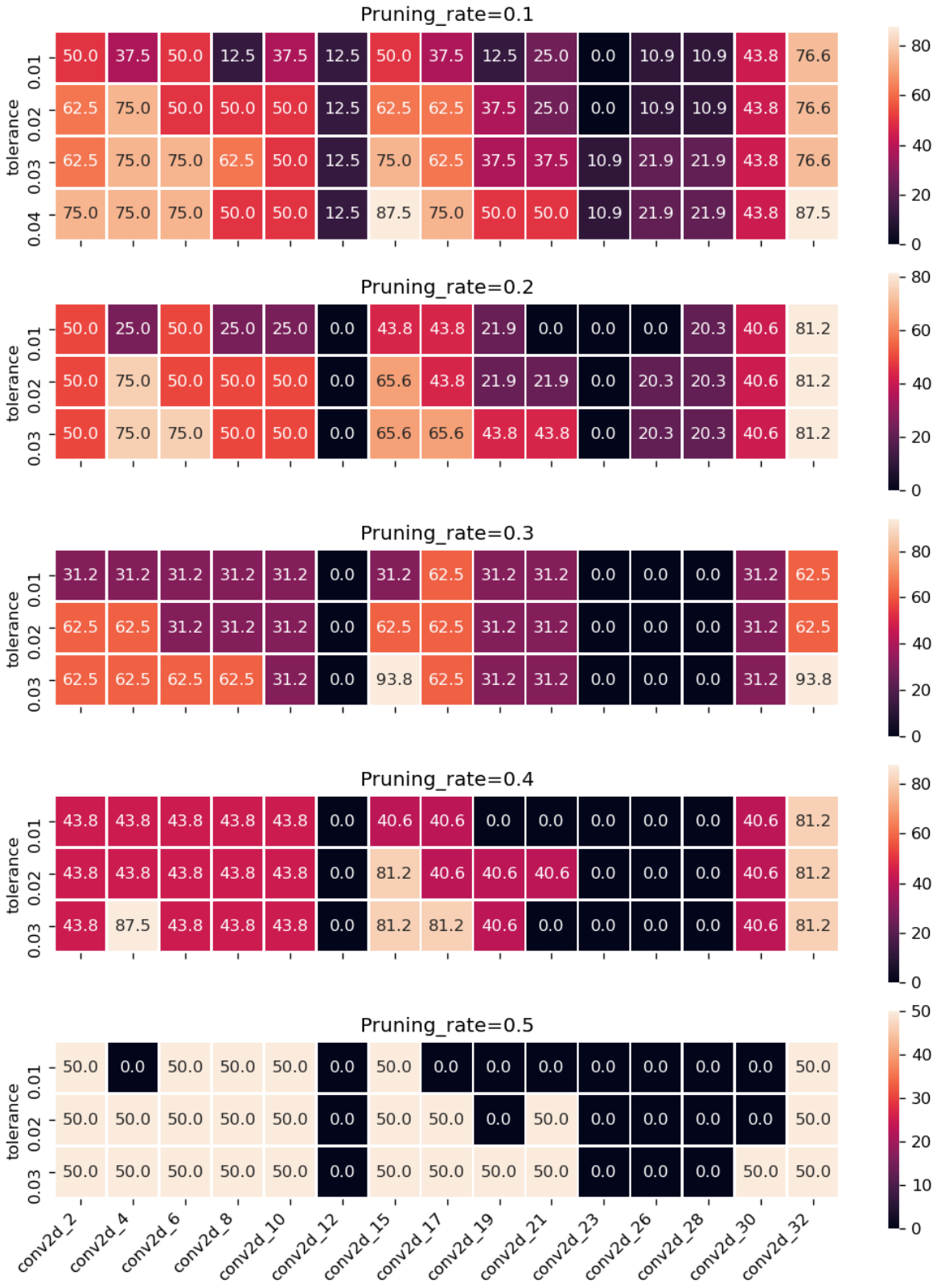


Figure 5.3: Heatmap of pruning rates per layer in different experiments of ResNet32 on Cifar-10. Each row is a run of experiment with mentioned pruning rate and accuracy degradation tolerance. Each column represents each prunable layer of the network. The annotation in each cell is the pruning ratio of the corresponding layer.

Experiment No.	Pruning rate(%)	Acc. Threshold	Acc. drop	Pruned(%)	Compression rate	FLOPs drop(%)
R32-CLE	-	-	0	31.51	1.46	23.94
R32-CLE-1.1	10	0.01	0.56	38.76	1.63	38.92
R32-CLE-4.1	40	0.01	0.53	34.67	1.53	33.18
R56-CLE	-	-	0	55.39	2.24	47
R56-CLE-1.1	10	0.01	0.88	68.11	3.14	66.02
R56-CLE-4.1	40	0.01	1.37	65.22	2.88	61.90

Table 5.2: Effect of CLE on ResNet-32,56 models pruning

Pruning Metric	Naming	Acc. threshold	Acc. drop	Pruned(%)	Comp. rate	FLOPs drop(%)
L1-Norm	N1	0.01	0.34	12.32	1.14	9.86
	N2	0.02	0.90	13.75	1.16	12.33
	N3	0.03	1.11	17.46	1.21	15.50
Geometric Median	G1	0.01	0.16	8.59	1.09	8.82
	G2	0.02	0.31	11.02	1.12	11.61
	G3	0.03	0.86	15.34	1.18	16.08
	G4	0.04	1.42	17.06	1.21	18.03

Table 5.3: Performance of MobileNetV2 pruned with different accuracy degradation thresholds and pruning metrics.

depthwise separable convolutions are pruned during the process. After each layer's pruning, there is a fine-tuning step of one epoch using the whole validation dataset. The optimizer employed was SGD with a learning rate of 10^{-3}. Similar to our previous experiment, we folded the BN layers at the beginning, before starting the pruning process.

- **Expected Results:**
  We expect to see more parameter reduction for higher thresholds. Also we expect to see performance differences between the same experiments with different pruning metrics.

- **Results:**
  As a matter of simplicity, we use abbreviations from Table 5.3 to rename the models. The alphabet G represents GM pruning metric, while the letter N stands for L1-Norm measure. The number following the alphabets indicates threshold percentage. The results in Table 5.3 show that models pruned with L1-Norm measure have a higher compression rate than the same models pruned with the GM measure. It means GM based pruning should be applied at higher thresholds (ϵ) to get the same compression rate as L1-Norm based pruning. As can be seen, the G2 model provides almost the same accuracy as the N1 model with significantly less parameter pruning (-1.3%), while reducing more FLOPs (+1.75%). Also, we can see that G3 with less compression rate than N3 has slightly more FLOPs reduction. Comparing G4 with N3, with almost the same compression rate, we observe that GM metric provides more speedup (+2.53 FLOP reduction) at the cost of 0.31% more accuracy loss.

  Considering these results as a search space for finding an optimized pruned model

with the best compression rate and less accuracy degradation, we get the Figure 5.4. The models on the blue curve (Pareto frontier) not dominated by other solutions are the best performing ones. G1 has the lowest compression rate with the lowest accuracy loss, and N3 has the highest parameter reduction with the most accuracy degradation at the end of the frontier. As previously discussed, we can see that N2 is dominated by G3, and G4 is dominated by N3. We should mention that the Pareto frontier would look different if the objective were changed from compression rate to speed-up.

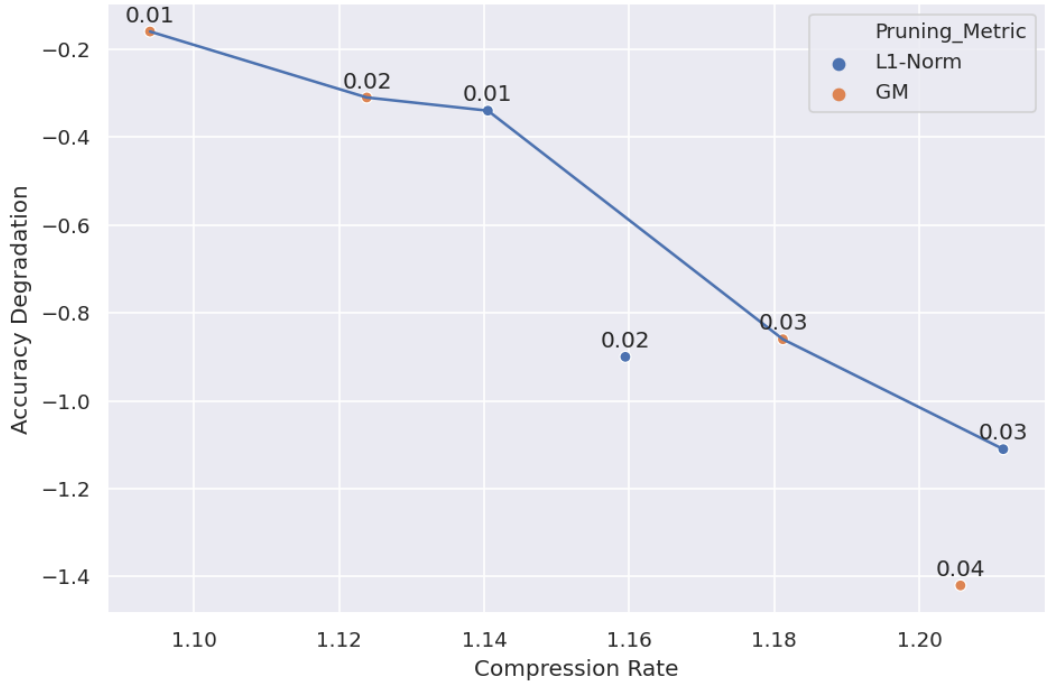


Figure 5.4: Different MobileNetV2 pruned models. The blue line represents the Pareto frontier containing the best models in the search space. The numbers above each model show the accuracy degradation threshold parameter.

- **Analysis:**
  Influence of network structure : To understand the influcene of network structure, we present the layer-wise pruning rate for MobileNet using a heatmap in Figure 5.6. Since we are applying a greedy pruning for each layer, increasing the threshold always helps in leveraging each layers pruning rate. However, there are layers which are more robust towards pruning (layers 1,6, and 10). Layers 6 and 10 are the last layers of two stages before FM size changes. Layer 15 in both metrics has the highest pruning rate. The second most pruned layer with L1-Norm metric is layer 9 while for GM metric it is layer 2. We could not find any meaningful relationship between depth of the network and the layers final pruning rate.

  Influence of pruning metric : Figure 5.6 compares models G2 and N1, which all show the same degradation of accuracy. To be fair, we also included the tuned model's accuracy after each layer pruning. It shows that the pruning rate of G2 is

higher than that of N1 for all layers until layer 14. This can explain why FLOPs reduction in G2 is greater than N1, since shallow layers have fewer parameters. It is also evident that G2 in shallow layers (smaller than layer 7) behaves better than N1 and N2. Taking G2 and N2 side by side, we can see that at layer 13, they perform the same, but at the top of that layer G2 loses the most accuracy. All these results suggest that a different pruning metric for different layers can be beneficial based on each layer's data distribution. In this experiment, shallow layers can benefit from GM-based pruning, while L1-Norm measure can lead to greater parameter reduction in deeper layers. This adaptability of the pruning measure is compatible with ideas discussed by other authors [HDL+20]. In such work authors proposed a framework to choose an appropriate pruning metric for each layer automatically. Nevertheless, they only examined it for ResNet models in a one-shot pruning scheme. More exploration is needed on this matter for MobileNet models and our pruning scheme.

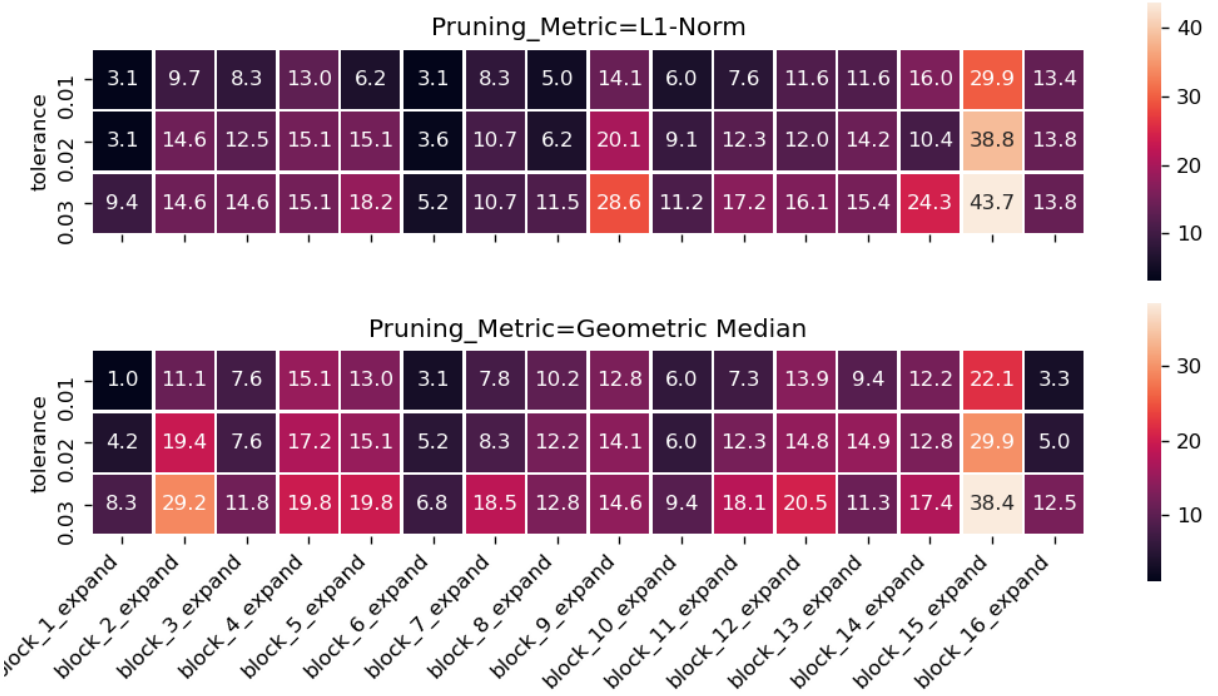


Figure 5.5: Heatmap of pruning rates per layer in different MobileNetV2 experiments on Cifar-10. Each row is a run of experiment with mentioned pruning metric and accuracy threshold. Each column represents each prunable layer of the network. The annotation in each cell is the pruning rate of the corresponding layer.

Influence of channel equalization : For this analysis, we apply CLE before the pruning step to further investigate a correlation between them. Results for two thresholds (0.01,0.02) with two pruning metrics are presented in Table 5.4. The weight ranges in this model are greater than previous ResNet models and consequently we do not observe the same behavior of pruning zero magnitude filters. Except from N1-CLE model, the remaining models gain higher compression rate and FLOP reduction with slight accuracy degradation or even better performances.

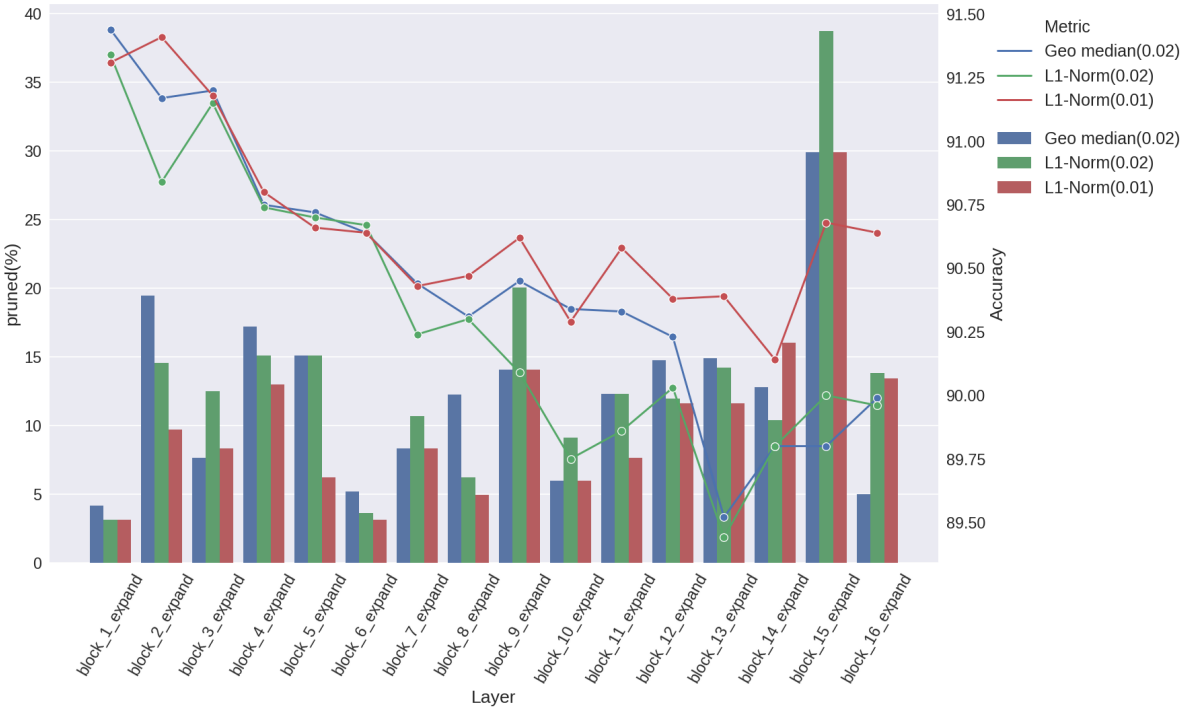


Figure 5.6: MobileNetV2 layer wise pruning visualization. Bar charts depict the pruning rate for each layer. The line chart indicates the model's intermediate accuracy.

Pruning Metric	Naming	Acc threshold	Acc drop	Pruned(%)	Comp. rate	FLOPs drop(%)
L1-Norm	N1 - CLE	0.01	0.25	11.28	1.13	11.75
	N2 - CLE	0.02	0.80	15.63	1.18	15.67
Geometric Median	G1 - CLE	0.01	0.32	13.95	1.16	11.98
	G2 - CLE	0.02	0.41	16.77	1.20	15.57

Table 5.4: MobileNetV2 pruning performance under channel equalization.

This change in the data distribution, decreases the layers' sensitivity towards pruning. For example, G2-CLE with 5.75% more parameter reduction and 0.1% more accuracy degradation than G2 acts better than G3, which is less compact with more accuracy loss. Comparing GM models with their counterparts in Table 5.3, we observe that CLE helps deeper layers to have more capacity for pruning which they suffered from previously. As a result, GM-based pruned models outperform N1-based ones.

- **Takeaways:**
  Taking advantage of zero magnitude based filter pruning without fine-tuning (ZP) depends on the tensors' weight distribution. For example, MobileNet-V2 does not offer this opportunity, whereas ResNets does. Layer-wise pruning parameter customization can also lead to improved pruned model performance, which is in line with findings in related work[HDL+20]. With MobileNet-V2, CLE changes the weight distribution so that pruning objectives (both FLOPs and parameters reduction) are in favour, while accuracy changes are negligible.

5.1.2 Case Study 2: Large Dataset (ImageNet)

ResNet50

- **Experiment Setup:**
  In this evaluation, we prune ResNet-50 without folding the BN layers. This choice is made, as we observed during the same experiment the model with BN layers folded is getting over-fitted during pruning, and its accuracy is lower than that of a model pruned without folding the BN layers. In order to simplify the pruning process, only a single hyper-parameter α is applied, without considering the accuracy degradation threshold. In our evaluations, we use two pruning metrics, L1-Norm and GM criteria. The model is tuned after pruning each layer just for one epoch, and due to time constraints, we use a sub-training set of 200K samples (200samples per class) instead of the entire training set. We apply SGD optimizer with learning rate of 10^{-4}. We fine tune the final pruned models with the whole training set for a few epochs to compensate for the accuracy loss. Using the same optimizer, we tune them with learning rates (0.01,0.001,0.0001) and epochs varying between 1-3 for each of them.

- **Expected Results:**
  During pruning, we expect to observe a sensitivity pattern related to the network architecture. In addition, hyper-parameter tuning and pruning metrics are expected to affect the pruned model performance.

- **Results:**
  The results are presented in Table 5.5. The models are given alphabetical names. As depicted in Figure 2.11, ResNet-50 has four stages that contain bottleneck blocks. Both models A and B are pruned based on L1-norm metric at a pruning rate of 0.3 and 0.4, respectively. Both models C and D are pruned in a mixed way. In both, only the last two stages are pruned as the majority of the parameters are located there. Sensitive layers with higher intermediate accuracy degradation are pruned with $\alpha = 0.3$ and for the rest, the pruning rate remains 0.4. Model C is pruned using L1-norm metric and model D is pruned based on GM criterion.

  We can see that model B achieves the highest compression rate and FLOPs reduction because in all layers 40% of the filters are pruned. However, it comes with the cost of 2.32% accuracy loss. The next models C and D, with the same compression rate (1.59), take the next places, followed by model A (1.50). However, FLOPs reduction in C and D is less than the model A because we do not prune the first two stages. We can see that models with optimized way of pruning have better performance than model A. Additionally, model D with GM pruning metric is 0.34% more accurate than L1-Norm counterpart.

- **Analysis:**
  Influence of network structure : In Figure 5.7a, we report the degradation of accuracy by layer after pruning and fine tuning in models Res50-P-A and Res50-P-B. ResNet-50 network has four stages (indicated by the resX prefix in the heatmap), and within each stage there are different counts of bottleneck blocks, as outlined in Section 2.1.3. Based on our pruning strategy, the first two layers (branch2a and branch2b in the heatmap) of each block are pruned. Coloured layer names indicate

Model	Acc. drop	Pruned(%)	Comp. rate	FLOPs drop(%)	Metric	Description
Res50-P-A	1.47	33.34	1.50	36.64	L1-Norm	All stages - 30%
Res50-P-B	2.32	42.80	1.75	46.74	L1-Norm	All stages - 40%
Res50-P-C	1.44	37.10	1.59	24.41	L1-Norm	stages 3,4 - mixed
Res50-P-D	1.10	37.10	1.59	24.41	GM	stages 3,4 - mixed

Table 5.5: Performance of ResNet-50 pruned with different pruning configurations.

which layers have the highest degradation inside each stage. As we expected, degradation with the same pattern is increasing when pruning rate is increased. The results show that in both models, the top layer in the first stage exhibits the highest sensitivity to pruning. Also, the last stage (res5) has the highest decay all its layers. We see that almost the first and last blocks of each stage are very sensitive to pruning. As a result, we decided to examine the next set of experiments with mixed pruning rate and different pruning measures.

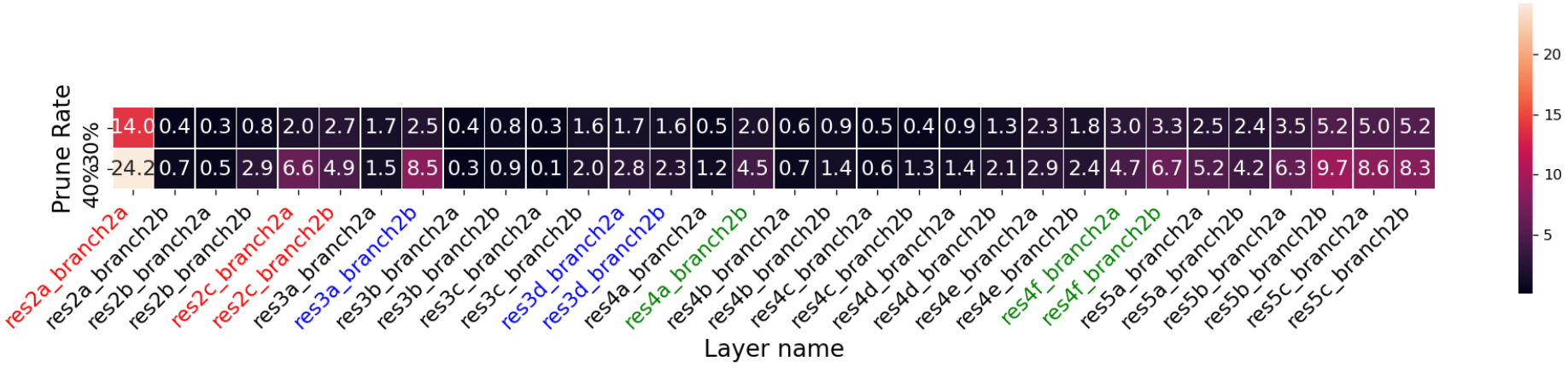
Influence of pruning metric : In the last two stages of ResNet-50, we applied pruning with 30% pruning rate for the layers that were deemed as sensitive, and as for the final stage which has the highest degradation, we considered only the last block. The remaining layers are pruned with $\alpha = 0.4$. Figure 5.7b represents the accuracy of models Res50-P-C and Res50-P-D after each layer is pruned and tuned. As we can see, for a few blocks at stage 3 (res4) the L1-Norm metric performs slightly better than that of GM, while in the last stage, GM outperforms the L1-Norm. In the end, Res50-P-C (L1-Norm) gained 65.87% accuracy and Res50-P-D (GM) gained 66.09%. With a few training epochs, the top-1 accuracy reaches 73.55% for the first model and 73.89% for the second one, respectively. GM metric suggests a 0.34% improvement over L1-Norm. It can be concluded that like MobileNet-V2(Cifar-10) mixed pruning could be useful, despite the fact that this difference is not very significant. In this experiment, layers in the last stage with higher degradation perform better with GM pruning, which results in improved performance at the end.

Influence of channel equalization : For this model, we are not able to examine the effect of channel equalization on its pruning performance. As mentioned earlier, the prerequisite step for CLE is BN layers folding, and ResNet-50 model with BN folded layers performs poorly during pruning.

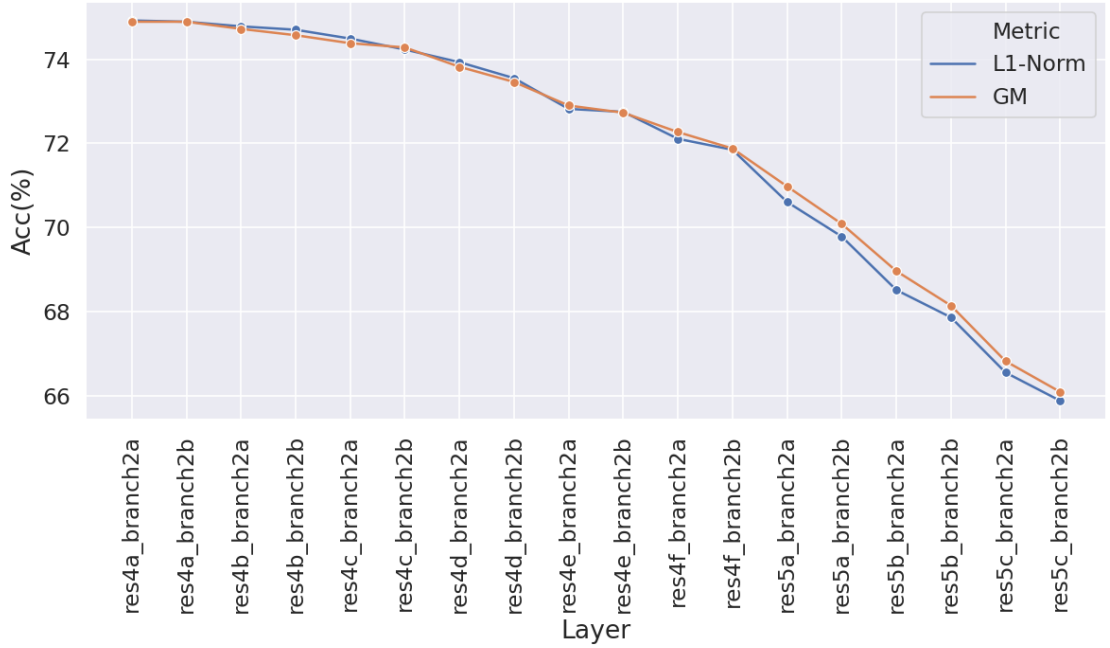
- **Takeaways:**
  It is important to tune hyper-parameters to provide a compressed model with sufficient performance. In addition, the depth and structure of the network play a role in this process as well. The inclusion of pruning metrics in layer-wise pruning has its own influence. However, we observed a minor difference between models with different pruning metrics in the ResNet-50 model.

MobileNet

- **Experiment Setup:**
  In this evaluation, the same reason as mentioned in ResNet-50 pruning, we prune



(a) Heatmap of accuracy degradation per layer in different experiments of ResNet50 on ImageNet. Each row is a run of experiment with mentioned pruning rate. Each column represents each prunable layer of the network. The annotation in each cell is the accuracy degradation after each layer pruning.



(b) Layer wise intermediate accuracy of pruned models Res50-P-C and Res50-P-D in Table 5.5.

Figure 5.7: ResNet50 pruning visualization

Model	Acc drop	Pruned(%)	Comp. rate	FLOPs drop(%)	Metric	Description
Mob-P-A	1.59	13	1.15	11.51	L1-Norm	Layers 11-16 , 30%
Mob-P-B	1.30	11.04	1.12	12.48	L1-Norm	Layers 7-16 , mixed
Mob-P-C	2.02	13	1.15	11.51	GM	Layers 11-16 , 30%
Mob-P-D	1.41	11.04	1.12	12.48	GM	Layers 7-16 , mixed

Table 5.6: Performance of MobileNetV2 pruned with different pruning configurations on ImageNet dataset.

the model without folding the BN layers. Pruning rate is the only hyper-parameter applied and pruning is assessed using the L1-Norm and GM metrics. During fine-tuning, the model is trained for one epoch using a subset of training images containing 200 samples of each class (200K samples in total). As an optimizer, we used SGD with a learning rate of 10^{-4}. Finally, we use the entire training set to fine tune the model for few epochs to improve the accuracy. Using the same optimizer, the learning rates for this step are (0.01,0.001,0.0001).

- **Expected Results:**
  We expect to observe less compression rate in MobileNetV2 pruning when compared to ResNet-50 according to the pruning strategy mentioned in Section 3.2.1. Additionally, two different metric sets might result in different pruning processes.

- **Results:**
  The result of experiments is presented in Table 5.6. We use an alphabetical naming for the experiments. In models Mob-P-A and Mob-P-C, 30% of the filters in expand layers from 11 till 16 are pruned away using metrics L1-Norm and GM respectively. In other set of experiments, we pruned the expand layers(7-16) in a mixed pruning way. 10% of the filter in sensitive layers and 30% of the filters in remained layers get pruned. These models are noted with mixed word in the description column in the Table 5.6.

- **Analysis:**
  Influence of network structure : A breakdown of the accuracy degradation of each layer for Models Mob-P-A and Mob-P-C is presented in Figure 5.8a. As seen in the heatmap, the last layers of each stage cause greater accuracy degradation during pruning than the rest. Labels of these layers have different coloring in the figure. Furthermore, we observe that as models get deeper, the accuracy degeneration gets greater. For the next set of experiments, we pruned the sensitive layers only at a rate of 10%.

  Influence of pruning metric : Figure 5.8a shows that the immediate layers' accuracy reduction resulted from Mob-P-C with GM based pruning is more than what happens with the L1-norm based pruning. However, after fine-tuning both models, they get nearly the same accuracy. This indicates that different pruning metrics may react differently to the fine-tuning hyper-parameters. Intermediate accuracy after pruning and tuning of each layer is shown in Figure 5.8b. According to this figure, the Mob-P-D model pruned based on GM performs slightly better than Mob-P-B(L1-Norm) except from the last two layers. Because of this performance

	Top-1 Acc [%]		
	ResNet32	ResNet56	MobileNet
Float32 Baseline	92.53	92.30	90.60
WQ8 - max abs	92.40	92.26	90.74
WQ8 - AQ8 (max abs)	91.81	91.10	87.67
WQ8 - AQ8 (min MSE)	92.12	92.10	89.11
WQ8 - AQ8(min propQE)	**92.44**	**92.17**	**89.32**
Best result vs baseline	-0.09	-0.13	-1.28

Table 5.7: Quantization results of the models based on Cifar-10 dataset

drop in the last two layers, Mob-P-D has 0.11% less accuracy than Mob-P-B, which is negligible.

Influence of Channel Equalization : This model cannot assess the effect of CLE on layer pruning for the same reason as ResNet-50.

- **Takeaways:**
  Training dataset magnitude and pruning strategy can highly influence pruning process and final model's performance. Our pruning strategy for already compact MobileNetV2 model is resulting in less compression rate with more accuracy degradation. Although careful layer-wise hyper-parameter tuning and appropriate pruning metric selection can improve it.

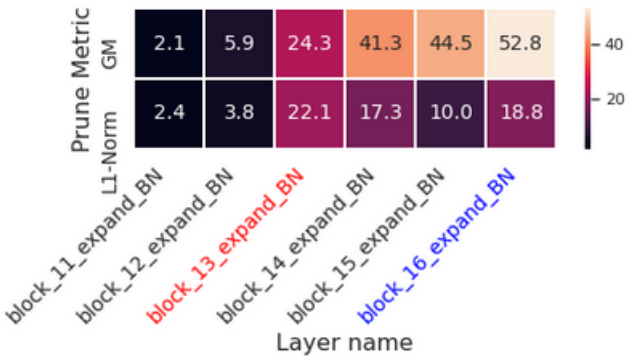
5.2 RQ1: Performance of Quantized model

In this section, we compare the methods for calculating quantization step sizes explained in 4.3.2 which are: maximum absolute value (max abs), minimal mean squared error (min MSE), and minimal propagated quantization error(min PropQE). We evaluate performance of them on the same benchmark networks and datasets mentioned in previous section. The calibration dataset is a random subset of training images, and results are reported on the validation dataset. For weight quantization we apply max abs to find the step size and for activations step sizes we all three methods are applied. As mentioned previously, we keep the biases at float32 precision.

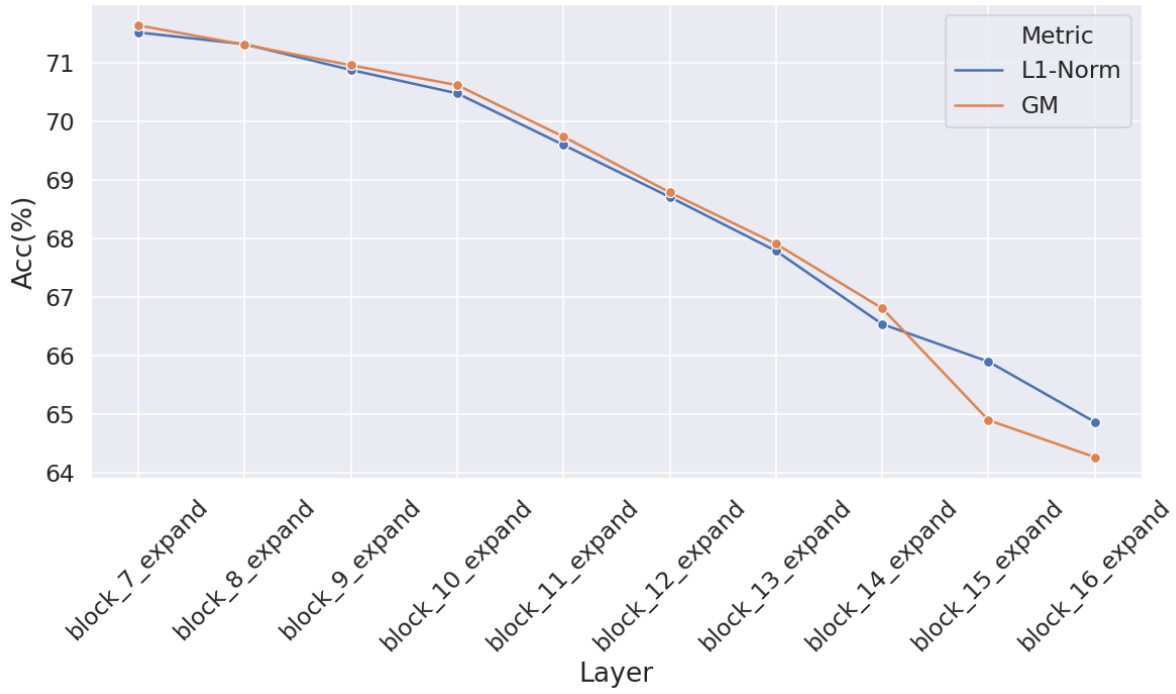
5.2.1 Case Study 1: Small Dataset (CIFAR-10)

- **Experiment Setup:**
  In this section, we apply the weight and activation quantization on ResNet-32,56 and MobileNet-V2 models. The sample dataset size used for finding the appropriate step size is 64, and we use the same calibration set for all experiments. Choosing another set sizes does not lead to noticeable differences.

- **Expected Results:**
  For ResNet models we are expecting minor accuracy degradation in the quantized models while performance degradation might be higher in MobileNet-V2 because



(a) Heatmap of accuracy degradation per layer in different experiments of MobileNetV2 on ImageNet. Each row is a run of experiment with the mentioned pruning metric. Each column represents a prunable layer of the network. The annotation in each cell is the accuracy degradation after each layer pruning.



(b) Layer wise intermediate accuracy of pruned models Mob-P-B and Mob-P-D in Table 5.6

Figure 5.8: MobileNetV2 pruning visualization

of small kernels [FAG19]. In addition, we expect to see better performance for optimized activation quantization methods.

- **Results:**
  In Table 5.7, the third row reports the baseline model accuracy. The fourth row shows the performance result of the weight quantized models. The step size of each layer is calculated using the maximum absolute value of the weight tensor (see Equation 4.4). We see no noticeable accuracy drop in any of the models. ResNet-32 and 56 both have accuracy drops of 0.13 and 0.04 respectively. MobileNetV2 improves its performance by 0.14% with weight quantization. In the next row, activations from previous models' output are quantized by their maximum absolute value. For ResNet-32,56, the accuracy drops are 0.72 and 1.20, while for MobileNet, it is 2.93. As we expected, performance degradation in MobileNetV2 is higher. We can see performance improvements by applying optimized quantization. Applying activation quantization using minimizing the MSE decreases the accuracy degradation to 0.41, 0.2, and 1.49 for ResNet-32,56 and MobileNetV2. When using propagated quantization errors for finding step sizes, mentioned values in the same order decrease to 0.09, 0.13, and 1.28. These results represent the best performance for each optimized model in the table.

- **Analysis:**
  In this part, we investigate the effect of methods CLE and IBC on quantization for two models, ResNet-56 and MobileNet. The step sizes for WQ are found using max abs and for AQ min MSE is applied. As mentioned in the previous section, we apply a conditional CLE on ResNet-56 to avoid turning all weights of few layers into zero. The performance improvement due to applying each method in ResNet-56 is not as notable as expected. It proves that the output channels in this model are not exposed to imbalance ranges, and the MAS of pre-activations is negligible (we get 0.05% improvement after applying IBC). In the end, we achieve 0.13% accuracy boost using both of these methods. Before we apply channel quantization on MobileNetV2, we need to replace ReLU6 activation with ReLU. Consequently, the model's accuracy went up to 91.03%. Based on the weight change data plotted, the first depthwise separable layer (Figure 5.9) has the highest imbalance values per filter. Consequently, the resulting quantization noise after CLE was reduced. It is interesting to note that we see no improvement after applying IBC. It indicates that small kernels may not necessarily be a cause of activation shift. Based on these results, accuracy degradation due to activation quantization (1.49% for the min MSE) must be investigated layer by layer for another noise source or problematic layer, which can be explored in future work.

		CLE	IBC	CLE+IBC
Model	WQ8	WQ8-AQ8	WQ8-AQ8	WQ8-AQ8
ResNet56	92.31	92.08	92.15	92.23
MobileNet	91.08	89.26	89.15	89.19

Table 5.8: Top-1 accuracy of quantized ResNet-56 and MobileNetV2 using CLE and IBC methods.

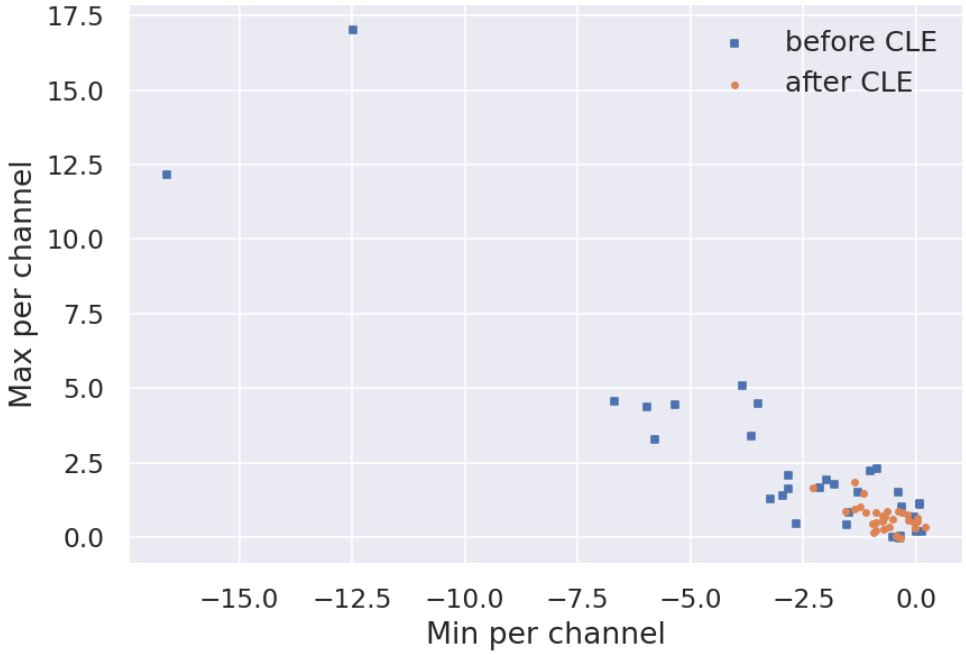


Figure 5.9: Output channel ranges of the first depthwise separable layer in MobileNetV2 trained on Cifar-10 before and after applying CLE.

	Top-1 Acc [%]	
	ResNet50	MobileNetV2
Float32 Baseline	74.99	71.72
WQ8 - max abs	74.75	7.3
WQ8 - AQ8 (max abs)	69.19	4.96
WQ8 - AQ8 (min MSE)	73.81	4.37
WQ8 - AQ8 (min propQE)	**73.94**	3.03
Best result vs baseline	-1.05	-66.76

Table 5.9: Quantization results of the ResNet50 and MobileNetV2 models based on ImageNet dataset

- **Takeaways:**
  Optimised methods for finding the step size for uniform quantization are providing the best performance. ResNet models are more robust against quantization than MobileNetV2. In some cases, a layer-by-layer analysis of activation quantization is needed in order to identify the potential sources of noise.

5.2.2 Case Study 2: Large Dataset (ImageNet)

		CLE	IBC	CLE+IBC
Model	WQ8	WQ8-AQ8	WQ8-AQ8	WQ8-AQ8
ResNet50	74.82	73.86	73.63	73.75

Table 5.10: Top-1 accuracy of the ResNet50 after applying cross layer equalization and bias correction.

	Float32 (ReLU): 64.49		
	CLE	CLE+IBC	IBC
WQ8 - max abs	59.40	63.53	63.27
WQ8 - AQ8 (max abs)	0.01	0.1	33.61
WQ8 - AQ8 (min MSE)	37.69	42.90	54.40
WQ8 - AQ8 (min propQE)	**43.90**	**56.76**	**60.08**
Best result vs baseline(ReLU6)	-27.82	-14.96	-11.64
Best result vs baseline(ReLU)	-20.59	-7.73	-

Table 5.11: Quantization results of the models based on ImageNet dataset

- **Experiment Setup:**
  We evaluate ResNet-50 and MobileNetV2 in this part by quantizing their weights and activations. The calibration dataset contains 100 samples, as recommended in previous [VSGA19].

- **Expected Results:**
  Compared to the ResNet models trained on Cifar-10, we anticipate a greater degradation in accuracy for the ResNet-50 model. Regarding MobileNet-V2, we anticipate the accuracy of the model to significantly degrade after WQ, so the model's performance will be impaired.

- **Results:**
  Results are shown in Table 5.9. After applying WQ to ResNet-50, we just observe 0.24% accuracy loss. We experienced 5.8% accuracy loss after applying AQ using max abs method. This degradation can be improved to 1.18%(min MSE) and 1.05%(min propQE) by optimizing how step sizes are found. In contrast, MobileNet-V2 behaves totally different, than we expected. By applying WQ, the model's accuracy drops from 71.72% to 7.3%, which further decreases after applying AQs. By the end, the best case that we get is a model with 66.76% accuracy degradation.

- **Analysis:**
  The results of using the CLE and IBC methods to minimize quantization noise in the MobileNet-V2 are shown in Table 5.11. As mentioned previously, the prerequisite of applying CLE is linear activation. When ReLU6 is replaced with ReLU we experience 7.23% accuracy degradation, which is very different from what previous work [NBBW19] claims. However, we were unable to find a reason for this decrease, so we followed up using this baseline (first and second columns in Table 5.11). As stated in the literature review, this model exhibits imbalanced channel ranges. With the help of the CLE method, the weight quantized network accuracy gets improved to 59.40% and by applying the extra IBC step (second column in Table 5.11), it reaches the baseline accuracy with 0.96% loss. Hence, MAS in this model is substantial due to small kernels. We can see that AQ using max abs fails for both cases and the best accuracy is achieved by min propQE (56.76%) after applying both CLE and IBC methods. Looking at the third column in Table 5.11, it is evident that IBC optimization individually can improve the accuracy of the weight quantized model to 63.27% which is 8.45% less than ReLU6 baseline. The best result after AQ is achieved by min propQE which is the best result among all for ReLU6 baselines with an 11.64% accuracy loss.

We also examined the influence of optimization methods on ResNet50 with the results presented in Table 5.10. Activation quantization step sizes are calculated using the minimum MSE method. Because the model is not suffering from MAS or imbalanced channel ranges, quantization optimization techniques and their combination are not generating any improvements.

- **Takeaways:**
  Optimised methods for finding the step size for uniform quantization are providing the best performance. MobileNetV2 model is really sensitive towards the quantization and quantization optimization techniques are necessary to decrease the noise and compensate the performance loss.

5.3 Summary

In this chapter, we evaluated the performance of each compression technique individually for different models trained on two datasets. In the first section, we analyzed the importance of pruning hyper-parameters and pruning metric choice on the accuracy of the final compressed models. Models with different architecture, depth, and training dataset behave differently during pruning. As an instance, MobileNetV2 models have less parameter redundancy than ResNet models. ResNet-56(Cifar-10) is more prunable than ResNet-50(ImageNet) with less accuracy degradation. These observations are compatible with the related literature. In our observations, we found that changing the weight distribution of a layer can be in favour of its pruning. As an instance, with the help of CLE and pruning without tuning we managed to prune 55% of the ResNet56 parameters. Also, it boosted the compression rate in the MobileNetV2(Cidar-10) model with even better accuracy.

In the second part, we examined weight and activation quantization methods and the effect of post-training quantization optimization techniques such as channel equalization and bias correction on them. Models based on their architecture and building blocks as well as training dataset show different performances. As an instance, MobileNetV2(Cifar10) unlike its similar model trained on ImageNet is not suffering from any activation shift caused by weights quantization. In all experiments, the optimized method of step size finding specially min PropQE provides better accuracy.

6 Evaluation and Results - Part2

The following chapter summarizes and discusses the results of the experiments we carried out to answer to the second question posed in (Section 3.1). The following sections comprise this chapter:

1. In Section 6.1, we discuss the performance of the combined compressed models using our proposed framework.
2. We summarize the basic findings of this study in Section 6.2, as well as the most important observations.

6.1 RQ2: Combined Compression

The purpose of this section is to examine the effect of post-training quantization on a pruned model for further optimization. Similarly to previous experiments, these evaluations are also divided according to the complexity of the dataset utilized. We examine them in two case studies of small and large data sets. For each dataset, we evaluate ResNet and MobileNetV2 models.

6.1.1 Case Study 1: Small Dataset (CIFAR-10)

In this part, we will present the results for three models. Resnet-32,56 and MobileNetV2 in two separate sections.

ResNet

- **Experiment Setup:**
  Using the pruned models from Subsection 5.2.1, we conduct the experiment. We use the linear quantization for quantizing both weights and activations. Weights are quantized using maximum abs, while activations are quantized using min MSE and min PropQE methods. We choose the candidates from the previous section with a pruning rate of 40% and an accuracy degradation threshold of 0.01. Pruning is based on L1-Norm magnitude.
- **Expected Results:**
  By taking a look at performances of the specified models in each step separately (pruning: Table 5.1, quantization:Table 5.7), we can see both models have almost the same accuracy, although with different compression rates. We do not expect to see a huge difference between their accuracy after joint-way compression as well.

- **Results:**
  The results of the combined optimization for the mentioned models is presented in Table 6.1. We can see that for both models, WQ has negligible effect on the pruned model performance. The difference between results of the two AQ methods is also small and the two models almost have the same accuracy degradation after applying each of the methods. ResNet-32 and ResNet-56 with compression rates of 1.33 and 2.22, can be quantized to 8-bit with a degradation of 0.65 and 0.69 in their accuracy.

	Top-1 Acc [%]			
	ResNet32	Acc Δ	ResNet56	Acc Δ
Original Model	92.53		92.30	
Pruned - baseline	92.20		91.97	
WQ8 - max abs	92.15	0.05	91.99	-0.02
WQ8 - AQ8 (min MSE)	91.65	0.79	91.25	0.72
WQ8 - AQ8(min PropQE)	**91.79**	0.65	**91.28**	0.69

Table 6.1: Combined compression of ResNet32 and 56 based on Cifar-10 dataset. Original models are pruned by $\alpha = 0.4$ and $\epsilon = 0.01$. Acc Δ represents the accuracy degradation of the baseline model.

- **Analysis:**
  After taking a closer look at the intermediate results of ResNet32 combined compression, we observed that there is an opportunity for more pruning without the need of fine-tuning. First, after quantizing the original model we recognized that magnitude of some filters turned to zero (second row of the Table 6.2 represent those filter counts). This shows that a pruning step without fine-tuning can compress more the quantized model. On the other hand, same thing happens after quantizing our already pruned model. In the pruned version of the model(fourth row of Table 6.2), layers which are sensitive to pruning (refer to Subsection 5.1.1) get a chance to be pruned. For this network, layers conv2d-[12,19,21,23,26,28] are the ones which based on the pruning hyper-parameters never get pruned. Weight quantization can provide a compression opportunity for these layers. Performance

	Conv2d_4	Conv2d_6	Conv2d_10	Conv2d_12	Conv2d_15	Conv2d_17	Conv2d_19	Conv2d_21	Conv2d_23	Conv2d_26	Conv2d_28	Conv2d_30	Conv2d_32
Main model filter cnt	16	16	16	32	32	32	32	32	64	64	64	64	64
WQ	2	6	4	2	11	12	5	6	6	10	12	27	54
Pruned model filter cnt	9	9	9	32	19	19	32	32	64	64	64	38	12
Pruned+WQ	-	-	-	2	-	-	5	6	6	10	12	1	2

Table 6.2: Comparison of the filter counts with zero magnitude in ResNet32. The columns represent the name of the layers which can be pruned. WQ means the weight quantized version of the original model, Pruned+WQ means the quantized version of the already pruned model.

details of these new models are presented in Table 6.3. In this table ZP means pruning of the filters with zero magnitude. As you can see, Quantized+ZP version of ResNet32 just with small accuracy degradation(0.09) can provide 31.76% parameter reduction independent of data-based pruning. This model is more compact than its pruned+quantized counterpart without the effort of the pruning step. However, considering the speed-up objective, it acts worse while most of the zero pruning is happening in deep layers. Zero weight pruning of an already pruned and quantized model helps to gain an extra 8.85% reduction of the parameters without any accuracy loss. ResNet56 has almost the same behavior as ResNet32. Zero pruned quantized model provides as compact model as the quantized+pruned version with less accuracy degradation (0.14 vs 1.02). ZP also helps the model to increase the pruned parameters by 10.3%. We can see that quantized ResNet56 has more potential of producing zero length filters in comparison to ResNet32. It can be assumed that deeper ResNet models on Cifar-10 will have the same pattern.

Model Name	Acc	Compression rate	Pruned (%)	FLOPs ↓(%)
ResNet32	92.53			
+Quant.+ZP	92.44	1.47	31.76	24.04
+Pruned+Quant.	91.79	1.33	24.71	28.68
+Pruned+Quant.+ZP	91.79	1.51	33.56	34.46
ResNet56	92.30			
+Quant.+ZP	92.16	2.24	55.39	47
+Pruned+Quant.	91.28	2.22	55.06	52.77
+Pruned+Quant.+ZP	91.30	2.89	65.36	59.15

Table 6.3: Effect of zero magnitude filters pruning

- **Extra study:**
  For an extra study, we investigated the effect of pruning on quantizing the weights of the network to lower bit. Similar to [CYD+20], we calculate sensitivity of each layer towards quantization using the Kullback–Leibler (KL) divergence between full-precision model and the quantized one. The result is plotted in Figure 6.1. Blue and red lines present the sensitivity of the model when the values are quantized to 8 bit and 4 bit respectively. Green and purple ones are the sensitivity of pruned version of those models when quantized in the same bit order. As we can see almost all layers have less sensitivity when they are first pruned and then quantized. The sensitivity gap between quantized main model and quantized pruned model for different layers are different which means different layers can benefit form pruned base model differently. It might be useful to prune some problematic layers before applying quantization. Although the idea is unlikely to make a noticeable impact on models based on small datasets, it merits further study for more complex tasks and datasets.

- **Takeaways:**
  Some layers in ResNet models have small weight ranges which after quantization leads to filters with zero L1-Norm magnitude. As a result, there is no need for pruning with a fine-tuning step and extra compactness can be provided by our proposed ZP step. This is the same attribute as after applying CLE to those layers

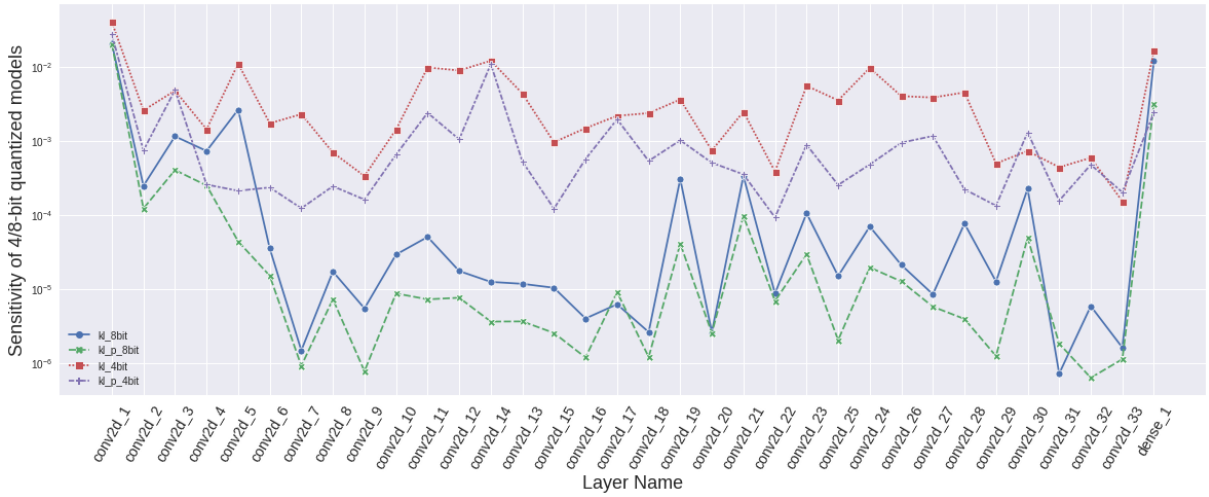


Figure 6.1: Sensitivity of ResNet32 when quantized to 4/8-bit weight precision; measured for both main(blue and red lines) and pruned models (green and purple lines).

(refer to analysis part of Section 5.1.1). A pruning step before lower bit quantization decreases models accuracy sensitivity and consequently might increase model performance, and it is worth investigating further.

MobileNet

- **Experiment Setup:**
  In this experiment, we apply quantization to previously pruned models in Section 5.1.1. Our weights and activations are quantized using linear quantization. The quantization step size is determined by applying max abs to weights, and min MSE and min PropQE to activations.

- **Expected Results:**
  Our expectation is that the pruning stage will have the greatest impact on the performance of compact models. On top of that, we expect to see different quantization performances for different pruned models.

- **Results:**
  We visualize the results of the optimized model performance in Figure 6.2. For model naming, we use the same abbreviations as in Table 5.3. The second group of bar charts presents the only accuracy degradation for pruned models. Third and fourth groups present the joint-way's model accuracy degradation in comparison to the quantized models(Table 5.7). The last two groups are absolute relative errors between a compact version and its only pruned version.

  By quantizing the pruned models, we do not experience the same behavior as ResNet networks. The weight distribution in MobileNet model is different from the ResNet, and it is not resulting in filters with zero magnitude. Consequently, we cannot benefit from ZP.

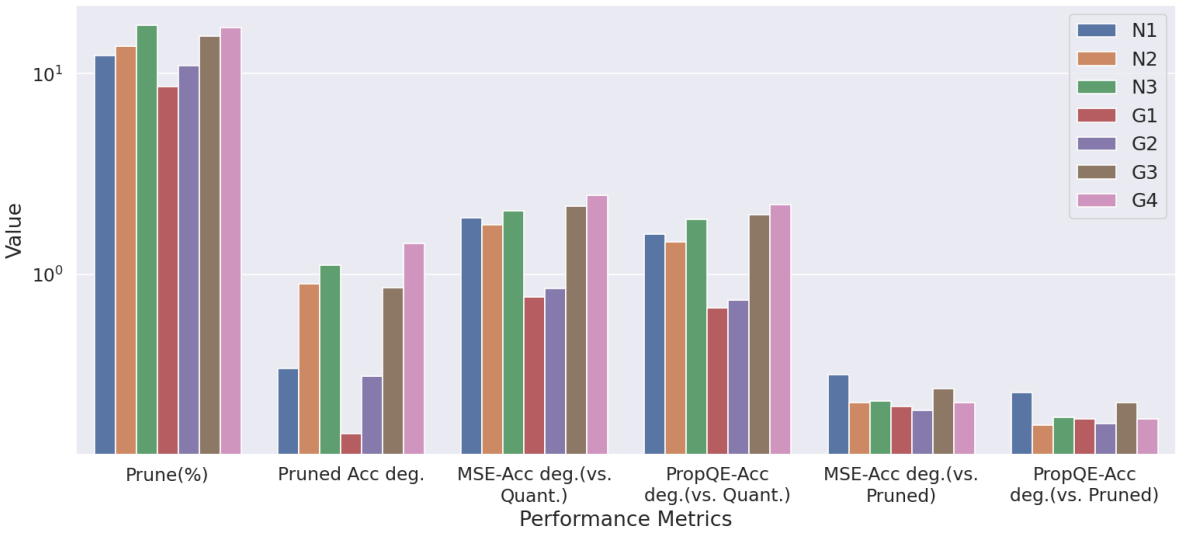


Figure 6.2: Visualization of MobileNet(Cifar-10) models optimized using pruning and quantization. Y-axis is in log scale and Errors are presented in absolute format.

Comparing accuracy degradation of optimized models vs. its only quantized version in Figure 6.2, we can see that L1-norm based optimized models on average have higher error rate than GM based optimized models. Although quantizing G3 model with $\epsilon = 0.03$ is causing almost the same error as quantized N3 (min MSE: 3.68 vs. 3.56 and min PropQE: 3.27 vs. 3.15). Also, for both metrics, using min PropQE for activation quantization, results in a slightly better model than applying min MSE. Taking a look at the relative errors, surprisingly we noticed that quantization error in N1 is higher than the rest (MSE: 3.40,PropQE: 2.79). The next most affected model is G3 with 3.14 and 2.68 reduction in accuracy of the model quantized with min MSE and PropQE respectively.

- **Analysis:**
  Influence of pruning rate : The performance of joint way compressed models with L1-Norm pruning metric behaves differently than only pruned versions in the models N1, N2, and N3. Combined version of N1 with less pruning rate and error leads to more degradation than N2. The difference can be explained by relative errors in the last two groups of Figure 6.2; clearly, the data distribution in N1 is causing more quantization errors. With higher pruning error, compression rate, and minor quantization performance degradation than N2, N3 takes third place. Based on the GM metric, with combined compression of the models, we notice that the error difference between (G1,G2) and (G3,G4) is getting smaller. This is explained by introducing different quantization errors to the network resulting from different pruned base models. In the quantization step, G3 has the highest relative error, and G1 is acting marginally worse than G2. In spite of this, the pruning performance is dominant and models after quantization retain the same order of performance as pruned ones.

  Influence of pruning metric : We learned from pruning experiments that the pruning error of G2 and N1 are almost the same, but the parameter reduction for G2 is 1.3% greater. The best accuracy column in Table 6.4 shows that quantized G2

					Error (vs. Quantized)		Relative error (vs. Pruned)	
model	CLE	Metric	Comp. rate	Best Acc	Min MSE	PropQE	Min MSE	PropQE
N1	-	L1-Norm	1.14	87.74	1.92	1.58	0.32	0.26
N2	-	L1-Norm	1.16	87.87	1.77	1.45	0.23	0.18
G1	-	GM	1.09	88.64	0.77	0.68	0.22	0.19
G2	-	GM	1.12	88.58	0.85	0.74	0.21	**0.18**
N1-CLE	✓	L1-Norm	1.13	88.85	0.47	0.35	**0.16**	0.15
N2-CLE	✓	L1-Norm	1.18	87.91	1.5	1.2	0.19	0.18
G1-CLE	✓	GM	1.16	87.97	1.44	1.14	0.25	0.24
G2-CLE	✓	GM	1.20	88.56	1.20	0.55	0.21	0.17

Table 6.4: Performance details of MobileNetV2 (Cifar-10) joint-way compressed models. Error values are in absolute format.

has a better accuracy than N1 (+0.84%). Quantization performance of G2 (relative error columns) can be attributed to its superiority. Same pattern is observed for G3 and N2 models with the same pruning accuracy loss. By comparing the G4 and N3 models with compression rates of 1.20 and 1.21 respectively, G4 has 0.31% greater accuracy loss. After applying quantization, we observe they have almost the same relative error. As a result, the final step which determines whether N3 is superior to G4 is pruning.

Influence of channel equalization : The second part of the Table 6.4 presents the results of CLE applied pruned models that are quantized. Taking a look at GM models, since G2-CLE compression rate is superior to G1-CLE and it suffers from fewer quantization noise, it has an accurate final result more comparable to that of G1-CLE (+0.59%). Comparing these results with G1 and G2 in the same table, we see that G2-CLE leads to the highest compression rate with negligible accuracy loss, compared to G1 with 1.09 compression rate. Finally, G2-CLE with a compression rate of 1.16 has the lowest final accuracy. Comparing the relative error columns, this order is explainable.

- **Takeaways:**
  In MobileNetV2 we observed that when the accuracy difference caused by pruning is higher than the difference caused by quantization, then pruning makes the final influence on final joint-way optimized model performance. In some cases pruning hyper-parameters, pruning metrics or pre-quantization techniques provide better data distribution for quantization which compensates the pruning difference.

6.1.2 Case Study 2: Large Dataset (ImageNet)

ResNet

- **Experiment Setup:**
  In this experiment, we apply quantization on already pruned ResNet-50 models in

Section 5.1.2. We apply the linear quantization, step sizes for weights are determined using max abs and for activations min PropQE and min MSE methods are applied.

- **Expected Results:**
  Quantizing the pruned models should result in the same pattern as quantizing the original model. This means that we expect pre-quantization techniques to not alter the behavior of the model. Furthermore, we expect to see the pruning step have more influence on the performance of the final model.

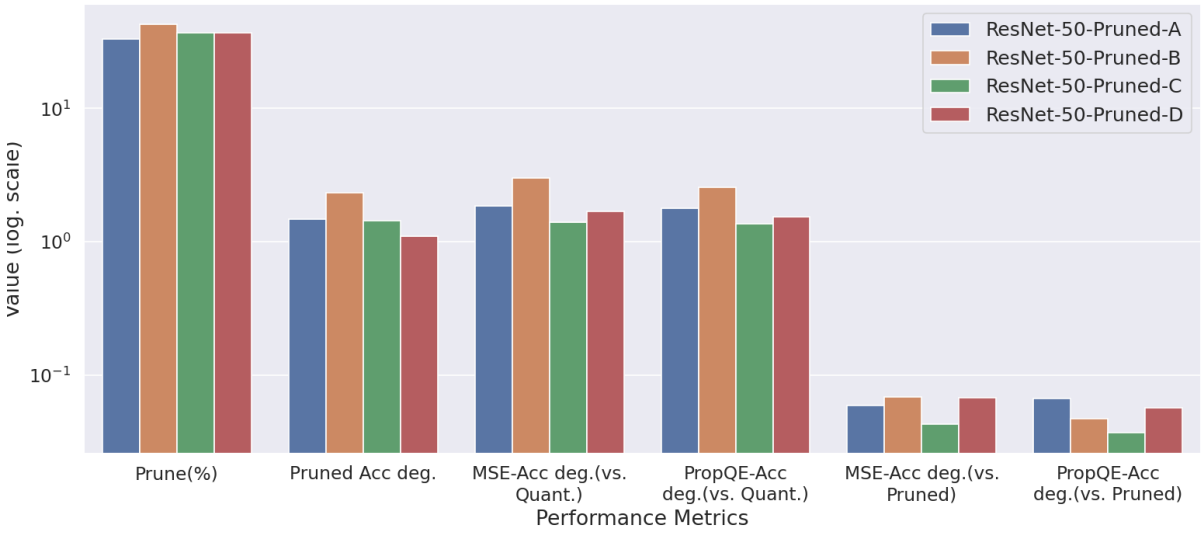


Figure 6.3: Performance visualization of ResNet-50 models optimized using pruning and quantization. Errors are presented in absolute format.

- **Results:**
  Figure 6.3 presents the evaluation results. The first and second groups show the pruning percentage for the models and their corresponding accuracy degradation in absolute format, according to Table 5.5. These next two groups represent the absolute accuracy error after applying quantization to pruned models in comparison to the quantized base model (numbers in Table 5.9). In the last two groups, we compare the absolute relative error of the combined compressed models to their only pruned models. For better understanding, numbers are presented on a log scale. As a general overview, pruning method is mainly responsible for final accuracy. The quantization error reveals that the performance difference between the two methods is marginal, which is the same behavior as only a quantized model. Quantized model B with a 40% pruning rate has the highest error rate, while quantized model C has the lowest accuracy degradation.

- **Analysis:**
  Influence of pruning rate : Considering compressed models A,B, and C with the same pruning metric but different pruning rates in Figure 6.3, we can see that the final error is following the same errors pattern as only pruned version. In other word, the dominant error is the pruning error for this model. Taking a look at relative errors in the last two groups, interestingly we observe that models A and B react differently. Considering min MSE method, Model A acts better than B while

Model	CLE	Metric	Comp. rate	Best Acc	Error (vs. Quantized) Min MSE	Error (vs. Quantized) Min PropQE	Relative Error (vs. Pruned) Min MSE	Relative Error (vs. Pruned) Min PropQE
Res50-P-A	-	L1-Norm	1.50	72.15	1.86	1.79	0.059	0.051
Res50-P-C	-	L1-Norm	1.59	**72.58**	1.41	**1.36**	0.043	**0.036**
Res50-P-D	-	GM	1.59	72.39	1.69	1.55	0.067	0.057
Res50-P-A	✓	L1-Norm	1.50	72.25	1.85	1.69	0.058	**0.047**
Res50-P-C	✓	L1-Norm	1.59	72.42	1.55	1.52	0.058	0.052
Res50-P-D	✓	GM	1.59	**72.61**	1.66	**1.33**	0.066	0.051

Table 6.5: Performance details of ResNet-50 joint-way compressed models. Absolute values of the errors are reported.

in min PropQE this order differs. This might occur because of a noise presence in the experiments and it is not affecting the final result.

Influence of pruning metric : Models C and D are pruned with the same strategy but different pruning metrics. In only pruned version model D with GM measure shows better performance than L1-Norm counterpart. However, after applying quantization, we observe higher error rate in model D (last groups in Figure 6.3 or last two columns in Table 6.5). This means that data distribution provided by L1-Norm is much more quantization-friendly rather than the GM metric. Also, this indicates that in some cases quantization error might dominate the pruning error.

Influence of channel equalization : The last three rows in Table 6.5 show the combined compression results with considering the extra CLE step after the pruning step. As we expected, this model has balanced channel ranges and CLE is not providing noticeable improvements. It decreases the quantization error in models A and D. In the end model D, with a 0.22% accuracy improvement performs better than the others.

- **Takeaways:**
  The performance of the pruned models can strongly affect the performance of final joint-way compressed models, which means models with higher accuracy after pruning will perform better even after quantization. Quantization performance can be the final influence on pruning accuracy, if the differences between pruned models are small. Assuming the main model does not have channel imbalances, it is likely that the pruned version also possesses the same property.

MobileNet

- **Experiment Setup:**
  In this section of the evaluations, we quantize weights and activations of the already pruned MobileNetV2 models in Table 5.6. Based on the quantized model performance discussed in Subsection 5.2.2, we apply IBC method before applying AQ step. We do not apply CLE while in best case the quantized model accuracy is less than the IBC provided ones. We quantize the weight ranges using max abs

Model	Metric	WQ8-IBC Acc	Comp. rate	Best Acc	Error (vs. Quantized) Min MSE	Error (vs. Quantized) Min PropQE	Relative Error (vs. Pruned) Min MSE	Relative Error (vs. Pruned) Min PropQE
Mob-P-A	L1-Norm	59.62	1.15	56.29	-2.44	-3.78	-0.61	-0.46
Mob-P-B	L1-Norm	61.02	1.12	58.28	-0.82	-1.8	-0.57	-0.41
Mob-P-C	GM	**62.73**	1.15	**59.51**	**+0.46**	**-0.57**	**-0.48**	**-0.33**
Mob-P-D	GM	57.90	1.12	56.08	-1.87	-4	-0.60	-0.48

Table 6.6: Performance details of MobileNetV2 (ImageNet) joint-way compressed models

method and for activation ranges two methods of min propQE and min MSE have been applied.

- **Expected Results:**
  This model's quantization is more challenging, so we expect to see more of a difference between the compressed models' performance because of their varied weight distributions.

- **Results:**
  Results are presented in Table 6.6. The Error columns show the accuracy difference between quantized pruned models and their only quantized version. Relative error columns present the difference between final compressed models and their only pruned version in a relative format while the base pruned models are different. We observe that Mob-P-C model has the final best accuracy among all of the models (59.51%) and less compressed-quantized Mob-P-D with 3.43% accuracy degradation stand in the last position. Considering relative error columns, we observe that best accuracy for each experiment belongs to the AQ using min PropQE.

- **Analysis:**
  Influence of pruning rate : Comparing models Mob-P-A and Mob-P-B in Table 5.6, we know that model A accuracy is 0.29% less than model B. Results in Table 6.6 shows that this difference is increasing to 1.99% after quantizing the compact models which is mostly originated in WQ regarding the accuracy values in WQ8-IBC column. On the other hand, in models pruned based on GM we see different behaviour. Accuracy of pruned version of Mob-P-C is 0.61% less than Mob-P-D and after quantization step model C performance dominates model D by 4.83%. Same as previous L1-Norm models, this degradation is caused by WQ. The pruning setting in model C provides a better data distribution for quantization rather than model D.

  Influence of pruning metric : In this part, we compare models in Table 6.6 with the same compression rate but different pruning metrics. The pruned version of model A with L1-Norm pruning metric and model C pruned with GM have almost the same accuracy. Although model C results in better data distribution for quantization and at the end its performance is 3.25% better than model A. Comparing models B and D with 1.12 compression rate and with marginal performance superiority of B over D, after quantization, accuracy of model B is 2.2% higher than D.

Comparing the results with their only quantized version, we observe more improvements in MSE errors rather than PropQE however at the end PropQE provides the best accuracy. Interestingly, we can see that Model C reaches higher accuracy than its only quantized version(min MSE: +0.46). The Next model with less accuracy difference is B. In these models, superiority of the quantization performance mostly caused by pruning metric covers the pruning degradation.

- **Takeaways:**
  For models with special network architectures and building modules, such as MobileNetV2, the quantization step is more sensitive. Final weight distributions in pruned models with different pruning configuration but nearly the same performance have a major effect on quantization performance. To get a more specific view, we recommend layer-wise investigation to measure each layer's quantization impact on final model accuracy. This way we will see how each pruning metric affects quantization and which pruning metric brings the model more gain.

6.2 Summary

In this chapter, we apply our proposed design to have a more compact model. We evaluate the performance of the quantized version of already pruned models from the previous chapter. According to our framework, in some applicable cases the post-training techniques (CLE and IBC) are also examined. The following are important observations:

- In models with small weight ranges, after quantization, a percentage of weights are quantized to zero and, consequently, part of the output channels are zero. This allows us to apply pruning even after quantization while it does not need fine-tuning step. We named it as ZP.
- Pruning performance in this joint-compression scenario is the main performance factor in models that are robust towards quantization or have a good quantization method.
- In different pruned models with similar performance, it is the performance of the quantization step that determines the final accuracy degradation, while the baseline weight distributions differ. For instance, two MobileNetV2 models pruned with the same compression rate and accuracy but different pruning metrics, the model pruned with GM criteria and quantized with min MSE acts even better than its only quantized version(+0.46) and 3% better than its L1-Norm counterpart.
- In terms of final optimization performance, we cannot make a concrete rule based on the observed accuracy, compression rate, and pruning metric. The layer-wise analysis, however, can be highly recommended to better understand the impact of each layer like its pruning rate and pruning metric on quantization noise of its weights and its corresponding feature maps and also its effect on final accuracy of the model.
- From our experiments, we also find out that a sparse version of the model may inherit the baseline model's characteristics. For example, MobileNetV2(ImageNet)

pruned version also suffers from MAS after WQ, and pruned ResNet-50 also provides balanced weight ranges.

7 Conclusion and Future Work

The conclusions that we drew from this thesis are described in this chapter. Our study may be extended in the future with the help of some proposed practices.
This chapter has been organized as follows:

- Section 7.1, outlines the main conclusions of our thesis work.
- In Section 7.2, we propose some modifications and extra features which can be incorporated in future work.

7.1 Conclusion

Our thesis attempts to compress CNN models in a combined way. For this purpose, two well investigated and straight-forward approaches of pruning and quantization are selected. Notably, we aim to compare the performance of compressed state-of-the-art models regarding the different configurations for pruning and quantization stages. Our focus is in the direction of generic post training optimization. Accordingly, we designed a two stage framework, for which the first step is structured filter pruning and the next one is 8-bit layer-wise quantization of weights and activations. To propose generic quantization solutions and handle its robustness towards noise, we also included optimization steps such as CLE and IBC in the framework. During the study, we expanded our research questions to one specific issue to know, to what extent, configurations of each step can improve final optimized model performance. First, we investigated behavior of each stage separately. During the experiments, we observed that in models with small enough weight ranges, after applying CLE or after quantization step, the magnitude of the some filters turn to zero which enabled us, with our ZP method (Zero Pruning without fine-tuning), to prune them. This type of pruning can be applied after quantization. We also observed that changing weight ranges using CLE may also be beneficial for the pruning process as well. This method was previously applied for channel equalization for quantization performance improvement. Regarding activation quantization, we found that the optimized method for finding step sizes always performs better than the baseline max abs one. Also, min PropQE always has superiority to min MSE.

We observed that the optimal solution for pruning is not necessarily providing the appropriate model for quantization.

Although, for models which are robust towards quantization, or have been properly quantized, the performance of the base pruned model has the most influence on the final performance. In models like MobileNetV2 in which the quantization noise is rooted in its building blocks and architecture, the quantization performance, has the highest impact on the end compression. Also in models with the same compression rate and accuracy, but

different pruning metrics, the baseline weight distribution and quantization performance make the final influence.

Putting all these together, we cannot come up with general learned policies, but observe that a good understanding of the model can be used to decide on the most impactful optimization. The pruning hyper-parameters and more importantly pruning metric are highly effective in quantization of each layer, its corresponding activations, and final accuracy of the model. Consequently, as a conclusion from our study, we suggest a layer-wise pruning that chooses the best configuration for the quantization should be followed. This can be named as *quantization-aware pruning* and next steps can be its automation, similar to the idea of [HDL+20] which learns the pruning metrics for each layer.

7.2 Future Work

Our thesis attempts to combine the two well known compression techniques of pruning and quantization. We investigate the effect of these techniques in classification tasks. Due to time constraints, we made limited choices in hyper-parameter tuning section and baseline NN architecture selections. Our design is modular and following suggestions can be developed to have a more generalized solution for different CNNs following different use-cases:

- *Computer Vision Tasks.* We have investigated the influence of feature extractor optimization on image classification tasks. This methodology can be extended to examine the optimization effect in other computer vision tasks such as object detection and semantic segmentation.
- *Data Free.* Our design can be extended to be data independent. In real world, there are use cases which we do not have access to the training data and a solution relying on data and fine-tuning is not applicable anymore [NBBW19]. There are few proposals that already study this direction, such as data-free quantization [CYD+20], data-free pruning [TLJ+21, YMA+20] and the combination of both [ZXS+20, HJFR20].
- *Mixed-bit Precision* In our design, we only consider 8-bit quantization because of simplicity and hardware support. To further extend the investigations, the effect of pruning on mixed-bit precision quantization can be evaluated. Using pruning can change the data distribution of a layer in a way in which it needs less bits for data representation.

Bibliography

[AMAZ17] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.

[BOFG20] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020.

[BTOK91] Wolfgang Balzer, Masanobu Takahashi, Jun Ohta, and Kazuo Kyuma. Weight quantization in boltzmann machines. *Neural Networks*, 4(3):405–409, 1991.

[BWD+20] Chongke Bi, Jiamin Wang, Yulin Duan, Baofeng Fu, Jia-Rong Kang, and Yun Shi. Mobilenet based apple leaf diseases identification. *Mobile Networks and Applications*, pages 1–9, 2020.

[CBD15] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *arXiv preprint arXiv:1511.00363*, 2015.

[CCEKL20] Yoojin Choi, Jihwan Choi, Mostafa El-Khamy, and Jungwon Lee. Data-free network quantization with adversarial knowledge distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 710–711, 2020.

[CHS+16] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[CMGS20] Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, pages 1–43, 2020.

[CRK16] Chen Chen, Yuzhuo Ren, and C-C Jay Kuo. Global-attributes assisted outdoor scene geometric labeling. In *Big Visual Data Analysis*, pages 93–120. Springer, 2016.

[CWZZ17] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[CYD+20] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13169–13178, 2020.

[CZH18] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[DDG+19] Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan. Approximated oracle filter pruning for destructive cnn width optimization. In *International Conference on Machine Learning*, pages 1607–1616. PMLR, 2019.

[DDGH19] Xiaohan Ding, Guiguang Ding, Yuchen Guo, and Jungong Han. Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4943–4953, 2019.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[DJL+20] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[DLH+20] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

[DlPGK20] Cecilia De la Parra, Andre Guntoro, and Akash Kumar. Proxsim: Gpu-based simulation framework for cross-layer approximate dnn optimization. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1193–1198. IEEE, 2020.

[DRRT+18] Edmar RS De Rezende, Guilherme CS Ruppert, Antonio Theophilo, Eric K Tokuda, and Tiago Carvalho. Exposing computer generated images by using deep convolutional neural networks. *Signal Processing: Image Communication*, 66:113–126, 2018.

[Elg20] Mohamed Elgendy. *Deep Learning for Vision Systems*. Manning Publications, 2020.

[ES21] Prem Enkvetchakul and Olarik Surinta. Effective dataaugmentation and training techniques for improvingdeep learning in plant leaf disease recognition. 2021.

[FAG19] Alexander Finkelstein, Uri Almog, and Mark Grobman. Fighting quantization bias with bias. *arXiv preprint arXiv:1906.03193*, 2019.

[FC18] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[FCC90] Emile Fiesler, Amar Choudry, and H John Caulfield. Weight discretization paradigm for optical neural networks. In *Optical interconnections and networks*, volume 1281, pages 164–173. International Society for Optics and Photonics, 1990.

[Gai18] Akash S Gaikwad. *Pruning convolution neural network (SqueezeNet) for efficient hardware deployment*. PhD thesis, 2018.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[GG20] Shachar Gluska and Mark Grobman. Exploring neural networks quantization via layer-wise quantization analysis. *arXiv preprint arXiv:2012.08420*, 2020.

[GKD+21] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[GKP+19] Julia Gusak, Maksym Kholiavchenko, Evgeny Ponomarev, Larisa Markeeva, Philip Blagoveschensky, Andrzej Cichocki, and Ivan Oseledets. Automated multi-stage compression of neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[GMG16] Philipp Gysel, Mohammad Motamedi, and Soheil Ghiasi. Hardware-oriented approximation of convolutional neural networks. *arXiv preprint arXiv:1604.03168*, 2016.

[GPMG18] Philipp Gysel, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *IEEE transactions on neural networks and learning systems*, 29(11):5784–5789, 2018.

[GPNV16] Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

[GSL19] Hui Guan, Xipeng Shen, and Seung-Hwan Lim. Wootz: A compiler-based framework for fast cnn pruning via composability. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 717–730, 2019.

[Guo18] Yunhui Guo. A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*, 2018.

[GYMT20] Jianping Gou, Baosheng Yu, Stephen John Maybank, and Dacheng Tao. Knowledge distillation: A survey. *arXiv preprint arXiv:2006.05525*, 2020.

[HCS+16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Proceedings of the 30th international conference on neural information processing systems*, pages 4114–4122. Citeseer, 2016.

[HCZ+19] Song Han, Han Cai, Ligeng Zhu, Ji Lin, Kuan Wang, Zhijian Liu, and Yujun Lin. Design automation for efficient deep learning computing. *arXiv preprint arXiv:1904.10616*, 2019.

[HDL+20] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2009–2018, 2020.

[HJFR20] Maxwell Horton, Yanzi Jin, Ali Farhadi, and Mohammad Rastegari. Layer-wise data-free cnn compression. *arXiv preprint arXiv:2011.09058*, 2020.

[HKD+18] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018.

[HLL+18] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[HLW+19] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019.

[HLZY19] Yang He, Ping Liu, Linchao Zhu, and Yi Yang. Meta filter pruning to accelerate deep convolutional neural networks. *arXiv preprint arXiv:1904.03961*, 2019.

[HMD15] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[HPTD15] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28:1135–1143, 2015.

[HPTT16] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[HPZ+21] Peng Hu, Xi Peng, Hongyuan Zhu, Mohamed M Sabry Aly, and Jie Lin. Opq: Compressing deep neural networks with one-shot pruning-quantization. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21), Vancouver, VN, Canada*, pages 2–9, 2021.

[HSK+12] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[HZC+17] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[HZRS16a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[HZRS16b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[IHM+16] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[JKC+18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[JSL+19] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4350–4359, 2019.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[KK20] Sungrae Kim and Hyun Kim. Mixture of deterministic and stochastic quantization schemes for lightweight cnn. In *2020 International SoC Design Conference (ISOCC)*, pages 314–315. IEEE, 2020.

[KL80] Virginia Klema and Alan Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on automatic control*, 25(2):164–176, 1980.

[KMGB21] Uday Kulkarni, SM Meena, Sunil V Gurlahosur, and Gopal Bhogar. Quantization friendly mobilenet (qf-mobilenet) architecture for vision based applications on embedded platforms. *Neural Networks*, 136:28–39, 2021.

[KPY+15] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[Kri18] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[KSZQ20] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516, 2020.

[KYH+18] Doyun Kim, Han Young Yim, Sanghyuck Ha, Changgwun Lee, and Inyup Kang. Convolutional neural network quantization using generalized gamma distribution. *arXiv preprint arXiv:1810.13329*, 2018.

[KYWW19] KC Kamal, Zhendong Yin, Mingyang Wu, and Zhilu Wu. Depthwise separable convolution architectures for plant disease classification. *Computers and Electronics in Agriculture*, 165:104948, 2019.

[LAT18] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBE15] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[LCO18] Manolis Loukadakis, José Cano, and Michael O'Boyle. Accelerating deep neural networks on low power heterogeneous architectures. 2018.

[LCY13] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[LGWS21] Tailin Liang, John Glossner, Lei Wang, and Shaobo Shi. Pruning and quantization for deep neural network acceleration: A survey. *arXiv preprint arXiv:2101.09671*, 2021.

[LKD+16] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[LLS+17] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

[LLZ+19] Yuchao Li, Shaohui Lin, Baochang Zhang, Jianzhuang Liu, David Doermann, Yongjian Wu, Feiyue Huang, and Rongrong Ji. Exploiting kernel sparsity and entropy for interpretable cnn compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2800–2809, 2019.

[LMX+20] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates. In *AAAI*, pages 4876–4883, 2020.

[LMZ+19] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3296–3305, 2019.

[LS18] Chong Li and CJ Shi. Constrained optimization based low-rank approximation of deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 732–747, 2018.

[LSV+17] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

[LSZ+18] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[LTA16] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International conference on machine learning*, pages 2849–2858. PMLR, 2016.

[LW17] Jian-Hao Luo and Jianxin Wu. An entropy-based pruning method for cnn compression. *arXiv preprint arXiv:1706.05791*, 2017.

[LW19] Congcong Liu and Huaming Wu. Channel pruning based on mean gradient for accelerating convolutional neural networks. *Signal Processing*, 156:84–91, 2019.

[LW20] Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *Pattern Recognition*, page 107461, 2020.

[LWL17] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[LZH+16] Xiaqing Li, Guangyan Zhang, H Howie Huang, Zhufan Wang, and Weimin Zheng. Performance analysis of gpu-based convolutional neural networks. In *2016 45th International conference on parallel processing (ICPP)*, pages 67–76. IEEE, 2016.

[LZN+18] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018.

[MEA+18] Jeffrey L McKinstry, Steven K Esser, Rathinakumar Appuswamy, Deepika Bablani, John V Arthur, Izzet B Yildiz, and Dharmendra S Modha. Discovering low-precision networks close to full-precision networks for efficient embedded inference. *arXiv preprint arXiv:1809.04191*, 2018.

[MFAG19] Eldad Meller, Alexander Finkelstein, Uri Almog, and Mark Grobman. Same, same but different: Recovering neural network quantization error through weight factorization. In *International Conference on Machine Learning*, pages 4486–4495. PMLR, 2019.

[MFL+20] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5191–5198, 2020.

[MLM16] Daisuke Miyashita, Edward H Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016.

[MLY+21] Xiaolong Ma, Sheng Lin, Shaokai Ye, Zhezhi He, Linfeng Zhang, Geng Yuan, Sia Huat Tan, Zhengang Li, Deliang Fan, Xuehai Qian, et al. Non-structured dnn weight pruning–is it beneficial in any platform? *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[MTK+16] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[NBBW19] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1325–1334, 2019.

[NPOV15] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks. *arXiv preprint arXiv:1509.06569*, 2015.

[OCC+19] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep learning vs. traditional computer vision. In *Science and Information Conference*, pages 128–144. Springer, 2019.

[ON15] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[PBP+18] Subhankar Pal, Jonathan Beaumont, Dong-Hyeon Park, Aporva Amarnath, Siying Feng, Chaitali Chakrabarti, Hun-Seok Kim, David Blaauw, Trevor Mudge, and Ronald Dreslinski. Outerspace: An outer product based sparse matrix multiplication accelerator. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 724–736. IEEE, 2018.

[PBR18] Ratko Pilipović, Patricio Bulić, and Vladimir Risojević. Compression of convolutional neural networks: A short survey. In *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–6. IEEE, 2018.

[RBK+14] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[RORF16] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[SCYE17] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[SFZ+18] Tao Sheng, Chen Feng, Shaojie Zhuo, Xiaopeng Zhang, Liang Shen, and Mickey Aleksic. A quantization-friendly separable convolution for mobilenets. In *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, pages 14–18. IEEE, 2018.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[SHZ+18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[SLW16] Fangxuan Sun, Jun Lin, and Zhongfeng Wang. Intra-layer nonuniform quantization of convolutional neural network. In *2016 8th International Conference on Wireless Communications & Signal Processing (WCSP)*, pages 1–5. IEEE, 2016.

[SVRN20] Pravendra Singh, Vinay Kumar Verma, Piyush Rai, and Vinay Namboodiri. Leveraging filter correlations for deep model compression. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 835–844, 2020.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[TLCC20] Cheng-Hao Tu, Jia-Hong Lee, Yi-Ming Chan, and Chu-Song Chen. Pruning depthwise separable convolutions for mobilenet compression. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[TLJ+21] Jialiang Tang, Mingjin Liu, Ning Jiang, Huan Cai, Wenxin Yu, and Jinjia Zhou. Data-free network pruning for model compression. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[TM18] Frederick Tung and Greg Mori. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7873–7882, 2018.

[UGK+16] Gregor Urban, Krzysztof J Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matt Richardson. Do deep convolutional nets really need to be deep and convolutional? *arXiv preprint arXiv:1603.05691*, 2016.

[VSGA19] Sebastian Vogel, Jannik Springer, Andre Guntoro, and Gerd Ascheid. Self-supervised quantization of pre-trained neural networks for multiplierless acceleration. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1094–1099. IEEE, 2019.

[VSM11] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. 2011.

[WJZ+20] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*, 2020.

[WLL+19] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019.

[WNK+18] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8817–8826, 2018.

[WWC+20] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. Apq: Joint search for network architecture, pruning and quantization policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2078–2087, 2020.

[WYH+19] Yanzhi Wang, Shaokai Ye, Zhezhi He, Xiaolong Ma, Linfeng Zhang, Sheng Lin, Geng Yuan, Sia Huat Tan, Zhengang Li, Deliang Fan, et al. Non-structured dnn weight pruning considered harmful. *arXiv preprint arXiv:1907.02124*, 2019.

[WZH+20] Xiaodong Wang, Zhedong Zheng, Yang He, Fei Yan, Zhiqiang Zeng, and Yi Yang. Progressive local filter pruning for image retrieval acceleration. *arXiv preprint arXiv:2001.08878*, 2020.

[WZW+19] Huan Wang, Qiming Zhang, Yuehai Wang, Lu Yu, and Haoji Hu. Structured pruning for efficient convnets via incremental regularization. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[WZZ+18] Dong Wang, Lei Zhou, Xueni Zhang, Xiao Bai, and Jun Zhou. Exploring linear relationship in feature map subspace for convnets compression. *arXiv preprint arXiv:1803.05729*, 2018.

[XHCZ20] Sheng Xu, Anran Huang, Lei Chen, and Baochang Zhang. Convolutional neural network pruning: A survey. In *2020 39th Chinese Control Conference (CCC)*, pages 7458–7463. IEEE, 2020.

[XWL+19] Qian Xiang, Xiaodan Wang, Rui Li, Guoling Zhang, Jie Lai, and Qingshuang Hu. Fruit image classification based on mobilenetv2 with transfer learning technique. In *Proceedings of the 3rd International Conference on Computer Science and Application Engineering*, pages 1–7, 2019.

[YCS17] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5687–5695, 2017.

[YGZL20] Haichuan Yang, Shupeng Gui, Yuhao Zhu, and Ji Liu. Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2178–2188, 2020.

[YHC+18] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018.

[YMA+20] Hongxu Yin, Pavlo Molchanov, Jose M Alvarez, Zhizhong Li, Arun Mallya, Derek Hoiem, Niraj K Jha, and Jan Kautz. Dreaming to distill: Data-free knowledge transfer via deepinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8715–8724, 2020.

[YSP+20] Fei Yuan, Linjun Shou, Jian Pei, Wutao Lin, Ming Gong, Yan Fu, and Daxin Jiang. Reinforced multi-teacher selection for knowledge distillation. *arXiv preprint arXiv:2012.06048*, 2020.

[YSX+19] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. Quantization networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7308–7316, 2019.

[ZG17] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[ZL16] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[ZMDCF18] Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard. Adaptive quantization for deep neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[ZSD+20] Haoran Zhao, Xin Sun, Junyu Dong, Changrui Chen, and Zihe Dong. Highlight every step: Knowledge distillation via collaborative teaching. *IEEE Transactions on Cybernetics*, 2020.

[ZTZ+18] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. *arXiv preprint arXiv:1810.11809*, 2018.

[ZW19] Chen-Lin Zhang and Jianxin Wu. Improving cnn linear layers with power mean non-linearity. *Pattern Recognition*, 89:12–21, 2019.

[ZWHD20] Zhekai Zhang, Hanrui Wang, Song Han, and William J Dally. Sparch: Efficient architecture for sparse matrix multiplication. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 261–274. IEEE, 2020.

[ZWN+16] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[ZXS+20] Chen Zhu, Zheng Xu, Ali Shafahi, Manli Shu, Amin Ghiasi, and Tom Goldstein. Towards accurate quantization and pruning via data-free knowledge transfer. *arXiv preprint arXiv:2010.07334*, 2020.

[ZYYH18] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 365–382, 2018.