Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Technische Universität München

Department of Information Technology and Electrical Engineering

Institut für Integrierte Systeme
Integrated Systems Laboratory

Hardware-Friendly Mixed-Precision Neural Networks

Master's Thesis

Saqib Javed
sjaved@student.ethz.ch

March 2021

Supervisors: Dr. Matteo Spallanzani, spmatteo@iis.ee.ethz.ch
Georg Rutishauser, georgr@iis.ee.ethz.ch

Professor: Prof. Dr. Luca Benini, lbenini@iis.ee.ethz.ch
Prof. Walter Stechele, walter.stechele@tum.de

Acknowledgements

I would like to express my sincere gratitude to both of my advisors, Dr. Matteo Spallanzani and Georg Rutishauser for all the encouragement, technical guidance, and organizational support, which ensured the successful completion of my Thesis work.

I would also like to thanks Prof. Dr. Luca Benini for giving me this opportunity to join his Integrated Systems Laboratory at ETH-Zurich and do research work under his supervision. Furthermore, I would like to acknowledge the efforts of Prof. Dr. Walter Stechele in getting me funding for this research project and for giving me this opportunity to collaborate with Prof. Dr. Luca Benini within the scope of my master thesis.

Finally, I would like to thanks my awesome parents and all my teachers for putting their trust in me and helping me with their endless efforts to make this far in my career.

Abstract

For the deployment of Convolutional Neural Networks (CNNs) on battery-powered, energy-constrained edge devices, both weights and activations in a network can be quantized to reduce the energy consumption associated with CNN inference, as low-precision integer arithmetic uses less energy to execute than operations on full-precision floating-point data. However, this quantization of weights and activations incurs a significant loss in accuracy when weights are quantized to 2-bits. Therefore, we did a thorough evaluation of mixed-precision quantization of neural networks in this work and propose Hardware-Friendly Mixed-Precision Neural Networks where this accuracy loss was reduced using mixed-precision networks but in a more hardware-friendly manner. Using INQ method as a base, we explored different weight partitioning schemes. With an unstructured quantization approach, we can achieve ∼95% quantized weights with only 2% loss in accuracy as compared to a full-precision model with one of the proposed weight partitioning methods. Moreover, we explored the fact that if we leave the first and the last layer unquantized, this drop decreases to only 1%. The drop in accuracy is majorly contributed by the quantization of the last layer. In order to make it more amenable to hardware support, we impose a filter-wise structure on the intra-layer quantization. Under this constraint, extensive evaluation of the impact of quantizing the first and the last layer, order of quantization, and the impact of 8-bit activation quantization for ternary neural networks was performed. It was observed that quantizing the activations to 8-bits does not incur a significant loss and leaving the first and the last layer unquantized improves the accuracy significantly. Moreover, quantizing the high magnitude weights first provides the best final accuracy. Using these observations, one of our proposed hardware-friendly quantization strategies achieves the Top-1 validation accuracy of 63.6 % on the ImageNet dataset using Resnet-18 architecture, where 90% weights were quantized and the remaining 10% were left in full-precision. The accuracy was increased by 1.5 % as compared to the 100% quantized network using the INQ method. Comparing our results to the other state-of-the-art CNN optimization methods, the proposed method provides a reasonable trade-off between a significant reduction of computational demand and energy required, and an acceptable degradation in the accuracy.

Declaration of Originality

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor. For a detailed version of the declaration of originality, please refer to Appendix A.

Saqib Javed,
Zurich, March 2021

Contents

List of Figures

List of Tables

List of Acronyms

lr . . . . . learning rate

ASIC . . . . . Application-Specific Integrated Circuit

BNN . . . . . Binary Neural Network

CNN . . . . . Convolutional Neural Network

DNN . . . . . Deep Neural Network

FAIR . . . . . Facebook AI Research lab (FAIR)

FPGA . . . . . Field Programmable Gate Array

GPU . . . . . Graphics Processing Unit

ILSVRC . . . . . ImageNet Large Scale Visual Recognition Challenge

INQ . . . . . Incremental Network Quantization

IoT . . . . . Internet-of-Things

QNN . . . . . Quantized Neural Network

RL . . . . . Reinforcement Learning

SGD . . . . . Stochastic Gradient Descent

STE . . . . . Straight-Through Estimator

TWN . . . . . Ternary Weights Network

Chapter 1

Introduction

With rapid progress in the machine learning domain in the past decade, Computers are capable today of doing many tasks that require humans' cognitive capabilities, for example understanding images, processing natural language, or finding complex patterns in massive datasets. This shift is producing a huge disruption in every aspect of our life and artificial intelligence or machine learning is at the center of it all.

With the availability of massive datasets like ImageNet[1], COCO[2], KITTI[3] etc. and graphics processing units (GPUs), researchers were able to train deep neural networks (DNN) to achieve impressive results for various computer vision problems like image classification, object detection, semantic segmentation, etc. One of the most prominent classes of DNNs are convolutional neural networks (CNNs), which have been extensively researched and have proven capable of solving a wide variety of problems in fields such as signal processing and computer vision. Convolutional neural networks (CNN) had been the method of choice for most computer vision tasks since Krizhevsky et al.[4] demonstrated their power to win the ILSVRC-2012 competition. State-of-the-art results have been obtained in image recognition, object detection, speech recognition tasks, etc. using these CNNs. However, it is not easy to deploy these networks on battery-powered, energy-constrained edge devices as they require a huge amount of computational resources which makes their execution very energy intensive. To tackle this problem, researchers came up with the idea of Quantized Neural Networks (QNNs) where the weights and activations in the network are quantized to low-bit precision. For example, to minimize the model size and simplify convolution operations Binary Neural Networks (BNN) have proved themselves to be a great choice. In BNNs, the network weights are quantized to binary weights. Nevertheless, the drawback of this quantization is that it leads to a significant decrease in accuracy i.e 42.2% only on ImageNet dataset[1] .

This brings us to an idea of Mixed-Precision Neural Networks where weights and activations associated to each layer are quantized to a certain(homogeneous) precision tra-

ditionally. Mixed-Precision Networks are generally a good trade-off between Quantized and Neural Networks with full-precision floating-point weights. This thesis conducts an empirical study of Mixed-Precision Model Space Exploration. Detailed experiments are conducted to study the **structural parameters**, ones which affect the final structure of the network and the computations involved in running it and the **transparent parameters**, which only impact the training process/policy and have no influence on the final network structure. In order to minimize the hardware complexity overhead to support these mixed-precision networks, we explore approaches to impose suitable structure on the quantization patterns.

1.1. Motivation

In recent times, autonomous driving is one of the popular research areas, both in academia and industry. Even though the idea of autonomous driving is out there for some time now, only in recent years a significant amount of progress in the field has been achieved, fueled by the availability of more computational resources to the researchers. Some of the large automobile manufacturers e.g Tesla, BMW, etc. already offer some driving functions with a certain level of autonomy e.g. collision prevention by automatic breaking, auto-parking, blind-spot detection, lane-change detection, etc.. These cars perceive their surroundings using different sensors and make decisions like trajectory generation, path planning, obstacle detection, etc to aid the drivers. Autonomous driving is currently revolutionizing the business model of the automotive industry and the future promises the idea of mobility being offered as a service, which will benefit not only the automotive industry but also the common person.

Normally, these autonomous cars use advanced multi-camera systems to capture high-quality multi-resolution imagery from multiple points of view. The obtained visual information from these multi-camera systems must be processed in real-time to guarantee the safety of the driver as well as the other people in the surroundings at all times. Nevertheless, the available compute power in a car is limited, so an energy-efficient solution is required, especially for processing the extensive amount of visual data, provided by the advanced camera systems of these cars.

Machine learning algorithms have become a method of choice in many computer vision tasks, fuelled by the recent advances in Deep Neural Networks(DNNs). As mentioned before Convolutional Neural Networks (CNNs) achieve excellent results in computer vision tasks like image classification, object detection, semantic segmentation, and depth estimation. However, most of these successful CNNs exhibit highly complex architectures, consisting of many layers. These layers introduce a large number of parameters and it causes large computational and storage overheads. For example, ResNet-18[5] architecture has around 11 million parameters. That is why despite this impressive progress, existing CNNs face a major problem when we try to deploy these models on embedded

systems, mobile, and Internet-of-Things(IoT) devices. For that purpose, it is very common to deploy these networks in the cloud. However, this introduces additional latencies as data transfers are required for moving the main computational load from the edge-device to remote servers. Internet connection has to be established for this data transfer and it makes things even worse. So, it is unacceptable for a safety-critical application under real-time constraints, like in an autonomous driving scenario. To tackle this problem, the prime focus is now on enhancing the efficiency of DNNs themselves in a way that the computations can be performed on the edge-devices themselves, while still adhering to the accuracy and stringent timing constraints for inference.

The convolutional layers, which are the basic element of these CNNs are the major contributors towards the execution time during inference [6]. Moreover, each layer consists of a large amount of weights which depends on its size. For instance, one of the most popular networks, AlexNet[4] by Krizhevsky et al. comprises more than 60 million weights, which is approximately equal to 240 MB of storage when using the single-precision floating-point data format. Hence there is a need to optimize the CNNs so that they can meet the restricted memory and computational resources of an embedded system without making too much compromise in the accuracy of the model.

1.2. Challenges

To speed up the execution time and reduce the energy requirements of these CNNs, we have to come up with an efficient alternative to the standard convolutional layer. This alternative should reduce the number of computations as well as the decrease of the number of weights or the bits that are required to store these weights. In the past, research has proved that DNNs have high redundancy with regard to parameterization [7] and high-bit precision weights and intermediate activation values which can be quantized. Therefore, it is possible to decrease the model size by either reducing the number of weights or by reducing the number of bits required to store these weights **without significant loss of prediction accuracy**.

For that purpose, several model compression techniques such as Quantization[8], Pruning[9] and Knowledge distillation[10] have been proposed in the past. Numerous methods proposed by the researchers for optimization of these DNNs provide good results for inference speedups on a theoretical basis. However, it is hard to take advantage of these theoretical results on a real hardware. Unstructured Pruning is one the most familiar method used for model reduction. Unstructured pruning technique,[9] pruned weights and neurons whose magnitude is below a certain threshold and demonstrated very good theoretical compression ratio and speedup results. However, just like a lot of other unstructured pruning techniques, the effects of pruning are not observed in the network architecture but just in the sparsity of weight matrices. Thus, to take advantage of the resulting

pruned or compressed model, specialized software and hardware, like FPGAs[1] or ASICs[2] are required.

To avoid these limitations and decrease the overhead in deployment of these CNNs on hardware, an optimization technique is required which results in low computation, energy, and memory demand.

1.3. Contribution

To resolve these challenges of computational resources and energy demands of these deep neural networks, we propose a **Hardware-Friendly Mixed-Precision Neural Networks**. In the course of this work, we did a thorough evaluation of mixed-precision quantization of neural networks. To be specific, we focused on "extended TWNs" with 8-bit activations as Ternary Weight Networks (TWNs) provides a reasonable accuracy at highly reduced compute effort. However, the accuracy loss is still significant and worth improving. We used the INQ method as the starting point which is an iterative-based technique where a fraction of weights is quantized at each step.

First, we start of with analyzing the effects of quantizing the activations to 8-bits precision. As INQ is an iterative quantization method, this work also demonstrates the quantitative comparison of quantizing *"weights with lower magnitude"* first against *"weights with higher magnitude"* first. We also analyzed the behavior of both quantization orders at different stages of the training.

Moreover, a lot of proposed quantization methods in the past leave the first and the last layer unquantized. Nobody ever presented any quantitative comparison. For that purpose, quantitative analysis of quantizing both, only first, only last, or none of these layers was done, which is presented in the Chapter 5.

In the end, we move towards the goal of having a more structural approach when we do the quantization of our neural network as a very complex hardware is required for mapping unstructured mixed-precision quantized neural network. We tried two different approaches, *Channel-wise Quantization* and *Fixed Channel Proportion Quantization*. In the first method, we select the fraction of whole filters instead of weights at each iteration and quantize them. In the latter, we select the filters that will not be quantized in the beginning and do the INQ on the rest of the weights.

Sparsity analysis of the network layers after and during the quantization was also performed and is presented in the results chapter.

[1] FPGA: Field-Programmable Gate Array
[2] ASIC: Application-Specific Integrated Circuit

Chapter 2

Background

In this section, we start with giving a brief overview of deep neural networks where we mainly focus on Convolutional Neural Networks (CNNs). Moreover, a very general description of image classification is also presented without going into too many technical details. Further chapters will be focused more on the technical aspects of the problem.

2.1. Deep Neural Networks

Deep Neural Networks are one of the most powerful categories of Machine Learning algorithms. **Machine Learning** is an application of artificial intelligence that empowers computer systems to automatically learn and adapt to solve tasks without being explicitly programmed to do it by extracting information from existing data.

There are three main paradigms of machine learning, namely supervised learning, unsupervised learning, and reinforcement learning. However, we focus only on the supervised learning in this work. In **supervised learning**, the dataset comprises a lot of data points and each data point is represented by a pair of the form (x_i, y_i). Variable y is called dependent variable or target variable x is usually a m-dimensional vector called independent variables or input. The objective of supervised learning is to approximate a function or learn a model that maps input variables to the target variable.

Neural networks are currently being used in all the sub-branches of machine learning. A neural network is a computational graph composed of many small nodes called neurons. Every neuron performs a simple function on its input and feeds its output to the next one. The simplest possible neural network consists of a single neuron and is illustrated in Figure 2.1. The output Y of such a network can be expressed mathematically as given in Equation 2.1.

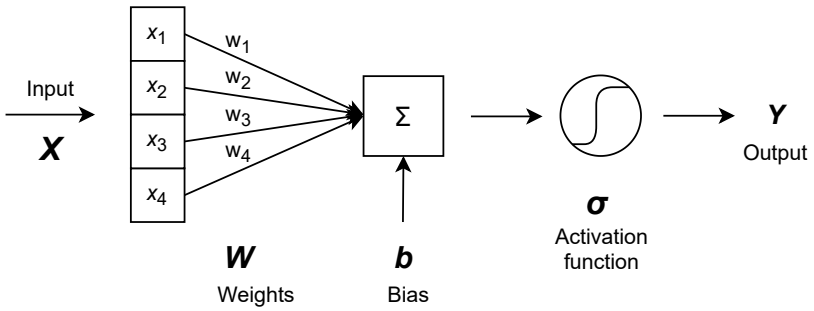

Figure 2.1.: Single Neuron Neural Network.

$$Y = \sigma(W * X + b) \tag{2.1}$$

W and b are the learnable parameters while σ is a non-linear function. By concatenating several such neurons, neural networks today are capable of modeling complex learning tasks in comparison to other algorithms that achieves the same by employing advance mathematical concepts.

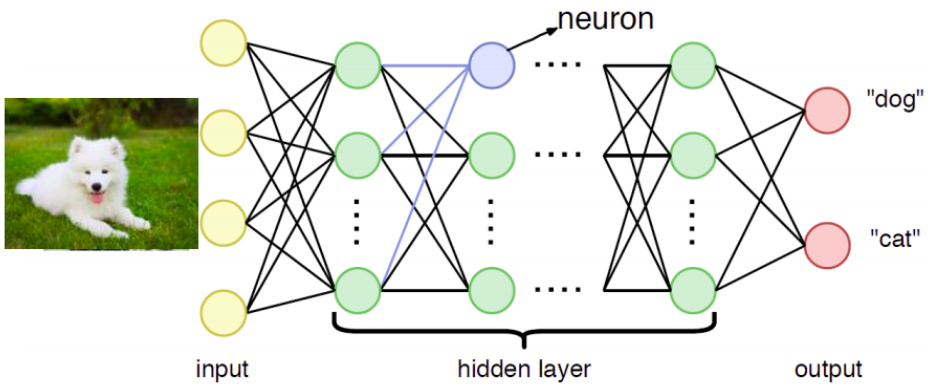

Figure 2.2.: Deep Neural Network Architecture. [11]

Deep neural networks (DNNs) are neural networks consisting of more than one layer with each layer containing hundreds of neurons. As mentioned earlier, these layers or neurons are concatenated to build DNNs. So, each layer could be viewed as a nested function of the preceding layers and can be approximated as

$$Y = \ldots \sigma(W_2 * (\sigma(W_1 * X + b_1)) + b_2). \tag{2.2}$$

The first layer of a neural network is the input layer. Normally, it is a stretched column

vector of the pixel values in the computer vision tasks. Then we have several hidden layers which consist of an enormous amount of neurons. The last layer is the output layer, which contains neurons equal to the number of classes and generates class scores for the classification task, as shown in Figure 2.2. Due to such extensive compute power at one's disposal these days, more and more deeper architectures[12] [5] are being leveraged to perform better.

2.1.1. Convolutional Neural Networks

As the input needs to be stretched into a column vector to be fed into the fully connected neural networks, we lose most of the contextual information and this makes a huge difference when we are dealing with image data. Moreover, as shown in Figure 2.2, neurons in each layer of fully connected neural networks are connected to all the neurons in the previous layer and it introduces a lot of variables to be optimized during training. This brings us to a special type of deep neural network i.e convolutional neural network (CNN), which was inspired by the visual cortex of cats[13].

In the mid-twentieth century, Hubel and Wiesel discovered two distinct types of cells in the visual cortex of cats, namely simple cells and complex cells. The **simple cells** (violet) prefer spots (dashed ovals in Figure 2.3) in the image, where they can respond most strongly to bars of a specific orientation. On the other hand, **complex** cells (green) takes input from many simple cells and therefore have more spatially invariant responses. These operations are replicated in a convolutional neural network as shown in Figure 2.3.

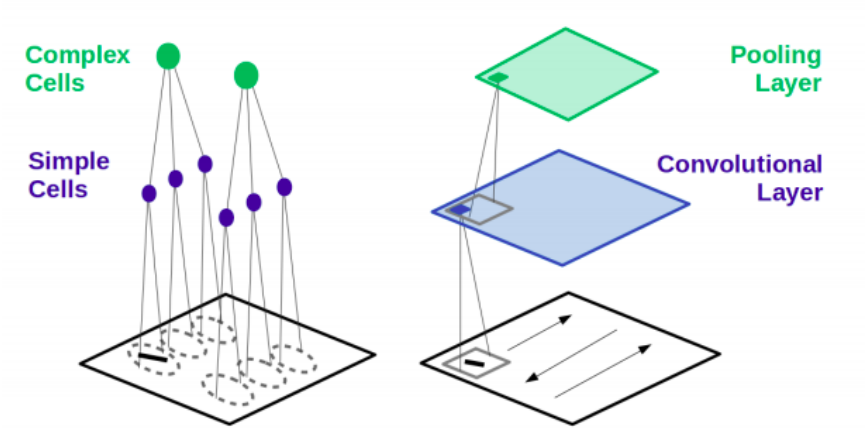

Figure 2.3.: Visual System and Convolutional Neural Network. [13]

Just like the visual cortex, CNNs have a layered structure consisting of neurons that only look at a local context to compute their outputs and keep the original shape of the

input data intact. Each neuron in a layer is sensitive or responsive to a certain region in input space which is also known as the receptive field. This feature enables CNNs to find spatial correlations in compact areas of an image. Overlapping receptive fields of multiple neurons extend the network's ability to detect features to the whole image. Due to this fact, CNNs perform very well when it comes to image processing tasks.

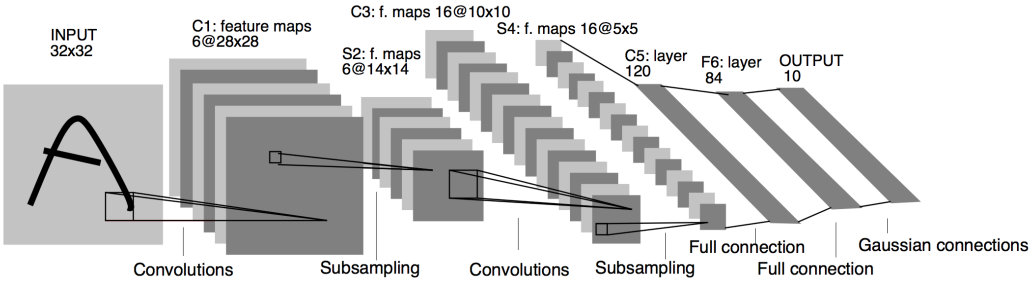

Figure 2.4.: Architecture of first modern-day CNN called LeNet-5.[14]

In 1989, Yann LeCun[14] invented one of the first modern-day CNNs and named it LeNet-5. Recognition of handwritten and machine-printed digits was performed using this network. Figure 2.4 illustrates the network architecture of LeNet. The two main components of a basic CNN, the convolutional and pooling layer are briefly discussed below.

2.1.2. Convolutional Layer

Generally, the weights in a convolution filter are arranged in quadratic structures, called kernels. The spatial dimensions of these kernels, height and width are small and the depth is identical to the depth of the preceding layer's output. As shown in Figure 2.5, these kernels are convolved with the input image or the output of the previous layer, and a dot product is performed on each spatial location to produce a single output in the activation map. Therefore, each output in the activation map or output feature map is only connected to a small local segment of the previous layer or input image. The height and width of the kernel determines the size of this local slice and is known as the receptive field of the neuron. Moreover, there are multiple filter stacks as shown in Figure 2.5 and each one of them produces a single output feature map or output channel. As across all the spatial locations same kernel weights are shared, this leads to a significantly less number of weight variables in CNNs as compared to fully connected neural networks.

Similar to fully connected neural networks, the non-linear activation functions are also applied to CNNs. After performing the convolution operation (dot product), usually "ReLU" non-linearity is applied. It implies that each kernel looks for certain features and gets activated only on finding those features. The first few convolutional layers in a CNN look for low-level features like edges and blobs and as we go deeper through the

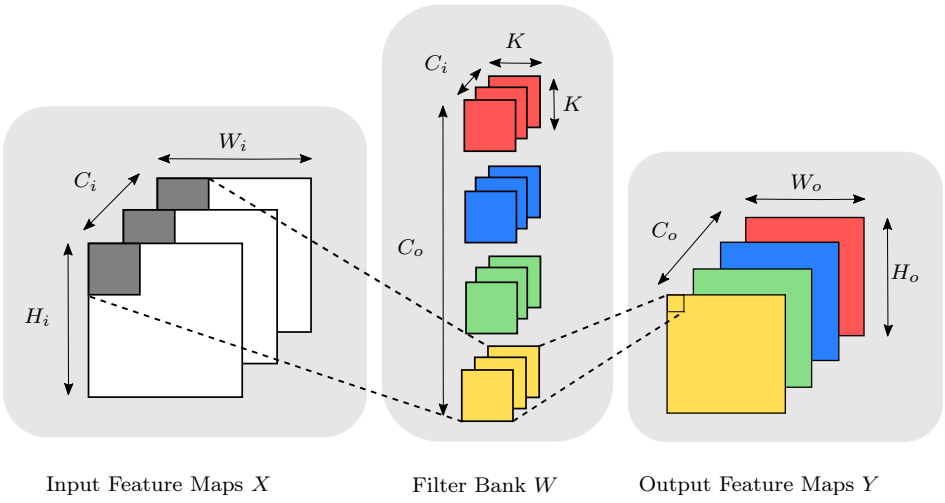

Figure 2.5.: Convolution Operation. [15]

network, the deeper layers use the low-level features extracted by initial layers and look for high-level features like shapes of objects.

2.1.3. Pooling Layer

A pooling layer is another building block of a CNN. As mentioned before, CNN kernels work as feature extractors, so it is important to select the most dominant features from each spatial location. For that purpose, the pooling layers are used in CNNs after every few convolutional layers. The pooling layer also reduces the dimension of the feature maps and as a result, reduces the number of computations required in the following layers. Max pooling with kernel and stride each equal to 2 is one of the most common pooling operations used in CNNs.

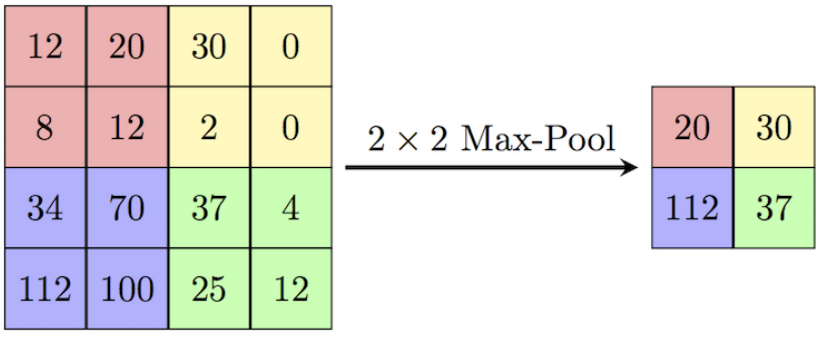

Figure 2.6.: Max Pooling with 2 x 2 pooling kernel and stride S = 2 [16]

As shown in Figure 2.6, it selects the maximum value from each unique non-overlapping 2x2 grid and down-samples the spatial-size of each input activation map by half without the need for any additional parameters. The downside is that we lose some spatial information, which is why the pooling layer is omitted in some CNN designs. Some of the CNN architectures that we used for experimentation are shown in the next chapter.

2.2. Image Classification

Image classification is one of the core problems in Computer Vision in which the input image is assigned one label from a fixed set of categories. The aim is to predict a distribution over labels to indicate our confidence and the label with the most probability is assigned to the image.

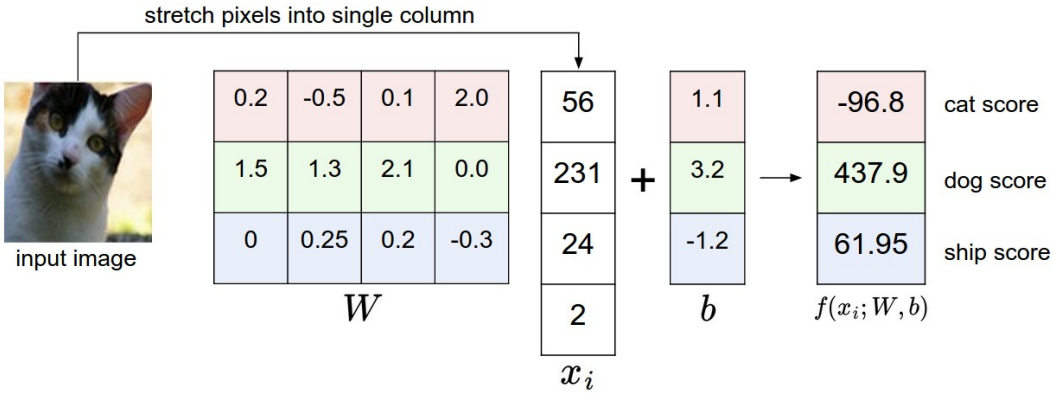

Figure 2.7.: Linear Classification [17]

Consider an input image with W width, H height and C channels respectively, the aim is to approximate a model that maps $M = H * W * C$ dimensional tensor of input/image pixel values x_i to label y_i where $y_i \in 1 \dots K$ is the label space of the classification problem. Just like linear regression, linear classification calculates the score of each class y_i using the linear combination of the input variables, which in the case of image classification are simply the pixel values.

$$y_i = Wx_i + b \tag{2.3}$$

In equation 2.3, b is the bias vector, W is KxM transformation matrix and y_i is the K-dimensional score vector for i^{th} image. As illustrated in Figure 2.7, the output of the linear classifier is simply the scores for each class in the label space. Generally, a softmax function is used as the next layer to map these scores to normalized class probabilities.

Chapter 3

Related Work

In the past decade or so, researchers came up with a lot of methods or proposals to make CNNs more efficient. The prime focus was to reduce the computational, energy, and memory demand of a model while keeping the performance as high as possible. As we are trying to optimize CNN such that it utilizes fewer resources, it is reasonable to start with a compact model as a baseline. Some of such compact state-of-the-art CNNs are presented in this section. Architectures of AlexNet[4] and ResNet[5] are briefly discussed as these CNNs were also used in experiments for the thesis work. Moreover, a brief overview of the Straight-Through Estimator (STE) is also presented that we used for the purpose of training activation quantized network. In the last section, Incremental Network Quantization (INQ)[18], which is the basis of this thesis work is briefly explained. This is the main weight quantization method that was primarily analyzed in this research.

3.1. AlexNet

In 2012, a deeper and wider CNN model was proposed by Alex Krizhevesky and others in their paper "**ImageNet Classification with Deep Convolutional Neural Networks**" as compared to LeNet. The proposed network won the most difficult ImageNet challenge for visual object recognition called the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 [19].

Until today, it is considered as one of the most significant breakthroughs in the field of machine learning and computer vision and is the point in history where interest in deep learning increased rapidly. It was also very interesting as they use multiple GPUs for training the network and reduced the computation time. This helped for future research purposes in the deep learning domain. The proposed architecture which was called "AlexNet" by the authors is illustrated in Figure 3.1.

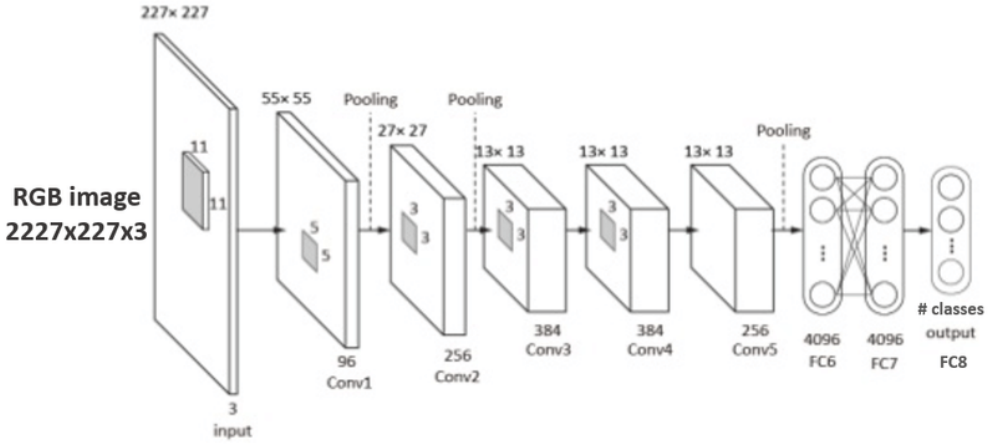

Figure 3.1.: AlexNet Architecture [20][4]

3.2. ResNet

In 2015, He et al.[5] evaluated deep residual nets with a depth of up to 152 layers on the ImageNet dataset. Training of these deep neural networks with high prediction performance was a ground-breaking success. The ResNet-152 model with 152 convolutional layers proposed by He et al. was winner of ILSVRC 2015 [19] challenge. Deep residual nets[5] also won the 1st places on the tasks of COCO detection, and COCO segmentation in COCO 2015 competition.

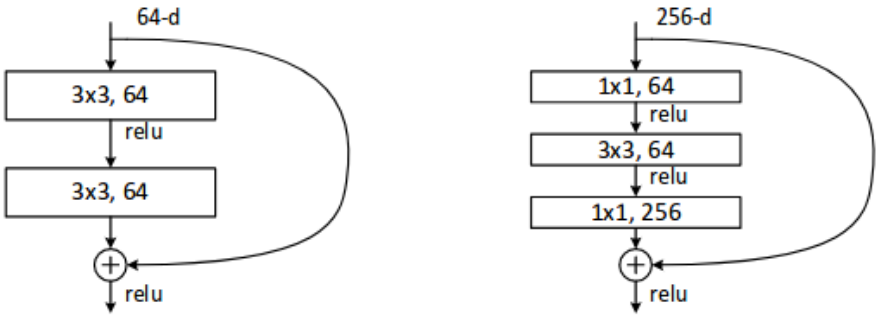

Figure 3.2.: Residual Blocks; Left: a building block (on 56×56 feature maps) for ResNet34. Right: a "bottleneck" building block for ResNet-50/101/152. [17]

The main idea was to incorporate residual learning in the network architecture. In order to achieve this goal, they came up with the design of residual blocks which incorporates

skip connections or sometimes called shortcut connections. Residual blocks that became the basic modules of these residual networks are shown in Figure 3.2. ResNet models consist of these stacked residual units where each residual unit is a small neural network with a skip connection.

However, the proposed network "ResNet-152" that won the ILSVRC 2015 challenge performed exceptionally well but it is not suitable at all for embedded applications due to its large size. However, this residual learning can be beneficial even with smaller models. For that purpose, He et al.[5] also proposed deep residual models ResNet-18 and ResNet-34, which have only 18 and 34 layers respectively. Both the models require fewer resources and also achieve competitive results on the ImageNet dataset. ResNet-18 has 8 residual blocks as illustrated in Figure 3.3.

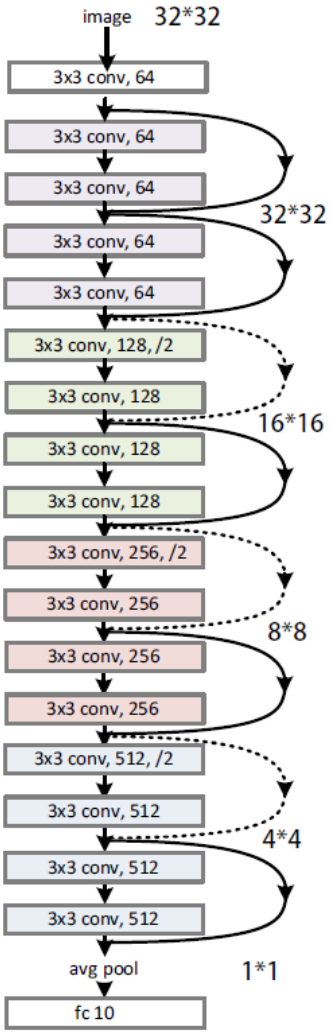

Figure 3.3.: ResNet-18 Architecture [5][21]

While presenting the results for ResNet[5] architecture on ImageNet[1] dataset, the authors used the ten-crop testing method. **Ten-crop testing**[22] is a standard procedure that takes four crops from each corner, as well as one from the center, and then performs a horizontal flip on each to pass 10 crops from a single test image through the network. On the other hand, all our experiments were done using single crop testing.

3.3. Straight-Through Estimator

The major hurdle in the quantization aware training is that quantization layers are not differentiable. The training involves minimizing a piecewise constant function and it suffers from gradient vanishing problem almost everywhere in the network. Hence, the standard back-propagation or chain rule cannot be used directly for the training.

STE is useful here as using a straight-through estimator (STE)[23] in the backward pass only, the "gradient" through the amended chain rule can be made non-trivial. Generally, we can get a very good derivative approximation using STE for quantization aware training.

Essentially, it handles the quantization function as if it were an identity function in the clipping range $[\alpha, \beta]$ and constant function outside the clipping range. That is why the resulting derivatives are 0 outside the clipping range and 1 inside it. We use STE with 8-bit quantization in all our experiments for the quantization of activations. The STE function during forward and backward while training can be illustrated using Figure 3.4.

3.4. Incremental Network Quantization

As discussed in the previous chapters, an enormous amount of computational power and energy is required to train and deploy CNNs. Reduction in model complexity has become a necessity for CNN's practical application in embedded systems, mobile, and Internet-of-Things (IoT) devices since they do not have extensive power/computational resources on board. For that purpose, Zhou et al. came up with the idea of Incremental network quantization (INQ)[18] to have CNNs with low-precision weights. INQ is an innovative method published at the International Conference on Learning Representations (ICLR) 2017 and using this algorithm, we can convert any pre-trained full-precision CNN model into a version with low-precision weights by constraining them to either zero or power of twos. In contrast to the other methods, the authors stated that this method did not suffer from a significant amount of accuracy loss. Detailed analysis of INQ was done in this thesis work and the results are presented in the results section.

INQ introduces three interdependent operations, namely weight partition, group-wise quantization, and re-training. First of all, the weights in each layer of a pre-trained CNN

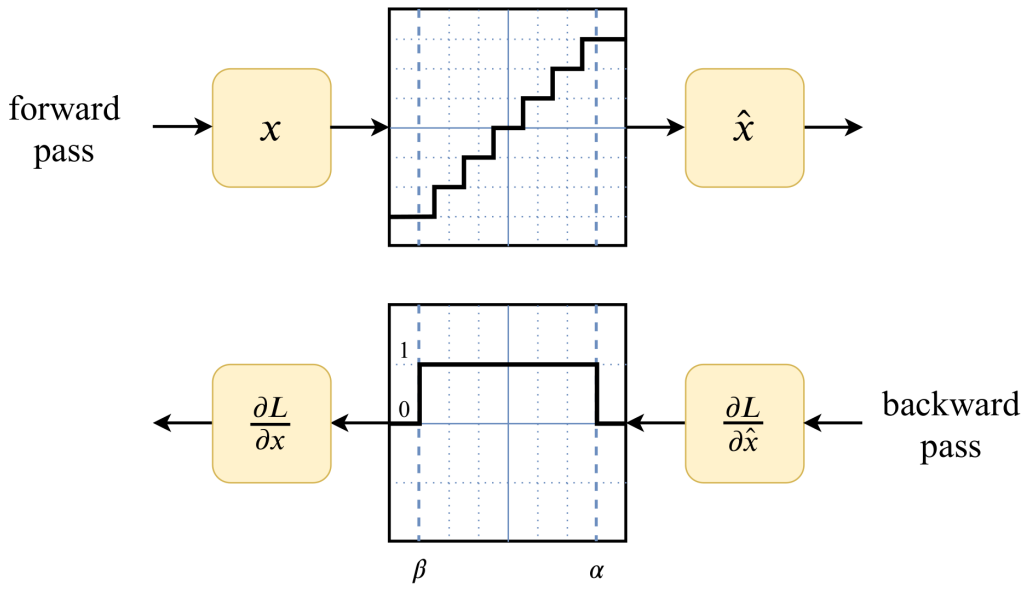

Figure 3.4.: Straight-Through Estimator (STE) [24]

model are divided into two distinct groups. The first group of weights is responsible for forming a low-precision base as these weights are quantized to a certain bit-precision. The other group of weights is re-trained and tries to compensate for the accuracy loss from the quantization. These three operations are performed iteratively on the latest re-trained group until we have all the weights converted into low-precision ones.

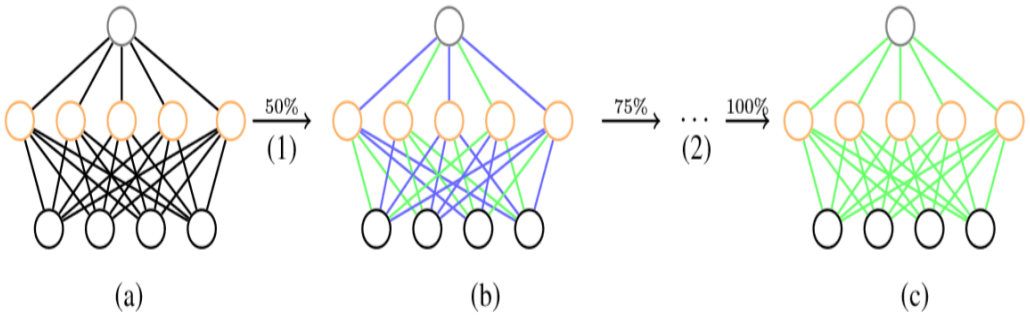

Figure 3.5.: Incremental Network Quantization (INQ) method [18]

In Figure 3.5, an overview of the INQ method is shown. Full precision pre-trained model is depicted in (a) and it is used as a reference. (b) shows the updated model after three proposed operations: weight partition, group-wise quantization (green connections), and re-training (blue connections). With all the quantized weights, the low-precision model is

shown in (c). Moreover, operation (1) illustrates a single iteration of (b), and operation (2) delineates the iterations of operation (1) on the previously re-trained weight partition until all the non-zero weights are constrained to either zero or powers of two.

Before INQ, many network quantization methods were proposed like BinaryConnect [25], BinaryNet [26], XNOR-Net [27] and TWN [28]. However, they all suffered from significant accuracy loss on deep CNNs, especially when being tested on the ImageNet large-scale classification dataset[1]. A common factor in all these methods is that they used a global policy while quantizing the weights. All these approaches convert weights simultaneously into low-precision, which incurs accuracy loss. On the other hand, accuracy loss becomes even worse if we try to train low-precision CNNs from scratch instead of using a pre-trained model as a starting point.

Another big advantage of using INQ is that it works independently of the network architecture and tries to achieve a lossless low-precision network from a pre-trained full-precision network. To illustrate the incremental strategy of INQ, Figure 3.6 takes a weight matrix and depicts the working of INQ in steps.

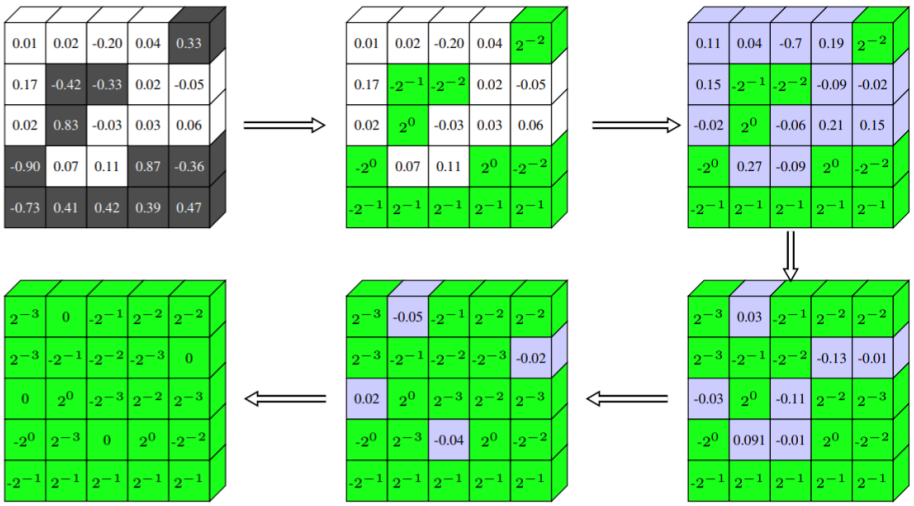

Figure 3.6.: Incremental Network Quantization (INQ) example [18]

The resulting outputs from the 1st iteration of the three basic operations of INQ are shown in the first row of the Figure 3.6. The cube on the top left depicts the first basic operation i.e weight partition and it creates two distinct groups. The middle image in the first row shows the quantization of the weights in the first weight group (green blocks), and the re-training operation on the second weight group (violet blocks) is illustrated in the top right cube. The second row shows the results from the following iterations of the INQ. The fraction of weights quantized at each iteration undergoes from 50%→75%→87.5%→100%.

The whole procedure of INQ is summarized in a form of an algorithm and presented in the next chapter along with modified algorithms that we also tested.

Chapter 4

Methodology

This chapter focus on the methodology that we used to convert high-precision networks into low-precision or mixed-precision networks. First of all, algorithm[1] for INQ strategy is briefly explained. Then the modified algorithms that we used for our experimentation to create mixed-precision networks are presented, namely Fixed Channel Proportion and Channel-wise Quantization. In the end, a brief explanation of our modified network architectures is provided.

To define the algorithm, consider a network with n number of layers. Suppose, for the l^{th} layer of the network, we have two weight partitions where $\mathbf{A}_l^{(1)}$ represents the first group of weights that will be quantized, and $\mathbf{A}_l^{(2)}$ depicts the other group of weights that requires to be re-trained. In the form of an equation, it can be defined as:

$$\mathbf{A}_l^{(1)} \cup \mathbf{A}_l^{(2)} = \{\mathbf{W}_l(i,j)\}, \quad and \quad \mathbf{A}_l^{(1)} \cap \mathbf{A}_l^{(2)} = \phi \tag{4.1}$$

To differentiate between two weight partitions, we define a binary matrix $\mathbf{T}_l$ which acts as a mask. $\mathbf{T}_l(i,j) = 0$ means $\mathbf{W}_l(i,j) \in \mathbf{A}_l^{(1)}$, and $\mathbf{T}_l(i,j) = 1$ means $\mathbf{W}_l(i,j) \in \mathbf{A}_l^{(2)}$. Generation of this binary matrix $\mathbf{T}_l$ is not conclusively defined in the INQ method and Zhou et al. explored random and large magnitude first approaches using this matrix. In addition to that, we also explored low magnitude first approach in this work.

Coming towards the training of the network's l^th layer, we can convert its weights to be either powers of two or zero by using the basic optimization problem defined in equation

[1]The whole algorithm is described in a similar fashion as presented in the original INQ paper[18].

4.2

$$\begin{aligned} \min_{\mathbf{W}_l} \quad & E(\mathbf{W}_l) \ = L(\mathbf{W}_l) + \lambda R(\mathbf{W}_l) \\ \text{s.t.} \quad & \mathbf{W}_l(i,j) \ \in \ \mathbf{P}_l, \ \text{if} \quad \mathbf{T}_l(i,j) \ = \ 0, \ 1 \leq l \leq L \end{aligned} \tag{4.2}$$

Where $L(\mathbf{W}_l)$ represents the network loss, $R(\mathbf{W}_l)$ denotes the regularization, λ is a positive coefficient, and the constraint term indicates that each quantized weight $\mathbf{W}_l(i,j)$ should be chosen from the set $\mathbf{P}_l$ consisting of a specific number of the values, which are powers of two or zero. The binary matrix $\mathbf{T}_l$ acts as a mask that forces zero updates to the weights that have been quantized.

Using these defined representations, we will now define few algorithms that were used in the course of this thesis work.

4.1. Incremental Network Quantization

The whole precedure of INQ is summarized in the Algorithm 1.[2]

Algorithm 1: Incremental network quantization for lossless CNNs with low-precision weights

input : X: the training data,
$\{\mathbf{W}_l : 1 \ \leq \ l \ \leq \ L\}$: the pre-trained full-precision CNN model,
$\{\sigma_1, \sigma_2, \cdots, \sigma_N\}$: the fraction of weights quantized at iterative steps

output: : $\{\widehat{\mathbf{W}_l} : 1 \ \leq \ l \ \leq \ L\}$:
the final low-precision model with the weights constrained to be either powers of two or zero

Initialize $\mathbf{A}_l^{(1)} \leftarrow \phi, \ \mathbf{A}_l^{(2)} \leftarrow \{\mathbf{W}_l(i,j)\}, \ \mathbf{T}_l \leftarrow 1, \ \text{for } 1 \ \leq \ l \ \leq \ L$
for $n = 1, 2, ..., N$ **do**
  Reset the base learning rate and the learning policy;
  for $l = 1, 2, ..., L$ **do**
    According to σ_n, create weight partitions and update $\mathbf{A}_l{}^{(1)}, \ \mathbf{A}_l{}^{(2)}$ and $\mathbf{T}_l$;
    Based on $\mathbf{A}_l^{(1)}, \ \text{determine } \mathbf{P}_l$
    Quantize the weights in $\mathbf{A}_l^{(1)}$
    Calculate feed-forward loss, and update weights in $\{\mathbf{A}_l^{(2)} : 1 \leq l \leq L\}$;
  end
end

[2] Algorithm is copied from the INQ paper[18].

4.2. Fixed Channel Proportion Quantization

In this method, we modified the INQ algorithm to have a more structural approach while quantizing weights in the network. This helps when we map mixed-precision networks to hardware. Otherwise, more complex hardware is required to run unstructured mixed-precision networks. To tackle this problem, we fix a proportion of channels using a parameter γ in the start and only quantize the weights in those channels while leaving others unquantized.

Suppose, for the l^{th} layer of the network, we have two channel-wise partitions where $\mathbf{C}_l^{(1)}$ represents the first group of channels that will be quantized, and $\mathbf{C}_l^{(2)}$ depicts the other group of channels that will stay in full precision. In the form of an equation, it could be defined as:

$$\mathbf{C}_l^{(1)} \cup \mathbf{C}_l^{(2)} = \{\mathbf{WB}_l\}, \quad and \quad \mathbf{C}_l^{(1)} \cap \mathbf{C}_l^{(2)} = \phi \quad and \quad |\mathbf{C}_l^{(1)}| = \gamma \times |\mathbf{WB}_l| \tag{4.3}$$

Where $\{\mathbf{WB}_l\}$ represents all the channels of the layers. However, once we create this channel partition at the beginning, only the weights in $\mathbf{C}_l^{(1)}$ partition will be assigned to $\mathbf{W}_l(i,j)$. The other weights in the channel partition $\mathbf{C}_l^{(2)}$ will be trained in full precision throughout the training phase. For the quantization of the channel partition $\mathbf{C}_l^{(1)}$, we use the same INQ algorithm. To make it even more clear, the proposed method is explained in the Algorithm 2

Algorithm 2: Fixed Channel Proportion Quantization

input : X: the training data, γ: channels proportion
$\{\mathbf{W}_l : 1 \leq l \leq L\}$: the pre-trained full-precision CNN model,
$\{\sigma_1, \sigma_2, \cdots, \sigma_N\}$: the fraction of weights quantized at iterative steps

output: : $\{\widehat{\mathbf{W}}_l : 1 \leq l \leq L\}$:
the final low-precision model with the weights constrained to be either powers of two or zero

Perform layer-wise channel partition depending on the γ;
Initialize $\mathbf{A}_l^{(1)} \leftarrow \phi$, $\mathbf{A}_l^{(2)} \leftarrow \{\mathbf{W}_l(i,j)\} \leftarrow \mathbf{C}_l^{(1)}$, $\mathbf{T}_l \leftarrow 1$, for $1 \leq l \leq L$
for $n = 1, 2, ..., N$ **do**
    Reset the base learning rate and the learning policy;
    for $l = 1, 2, ..., L$ **do**
        According to σ_n, create weight partitions and update $\mathbf{A}_l{}^{(1)}$, $\mathbf{A}_l{}^{(2)}$ and $\mathbf{T}_l$;
        Based on $\mathbf{A}_l^{(1)}$, determine $\mathbf{P}_l$
        Quantize the weights in $\mathbf{A}_l^{(1)}$
        Calculate feed-forward loss, and update weights in $\{\mathbf{A}_l^{(2)} : 1 \leq l \leq L\}$;
    end
end

This whole process can be further explained in a pictorial fashion as illustrated in Figure 4.1. Two successive iterations are shown in this figure. (a) shows the weights (white blocks) before any quantization. (b) shows the channels (blue) which are fixed and will stay in full-precision throughout the training process. Furthermore, the other channels are quantized using INQ method. Green blocks represents the quantized weights in these channels. (c) shows the the next iteration and (d) shows the output after final iteration where all the weights are quantized in the fixed channels.

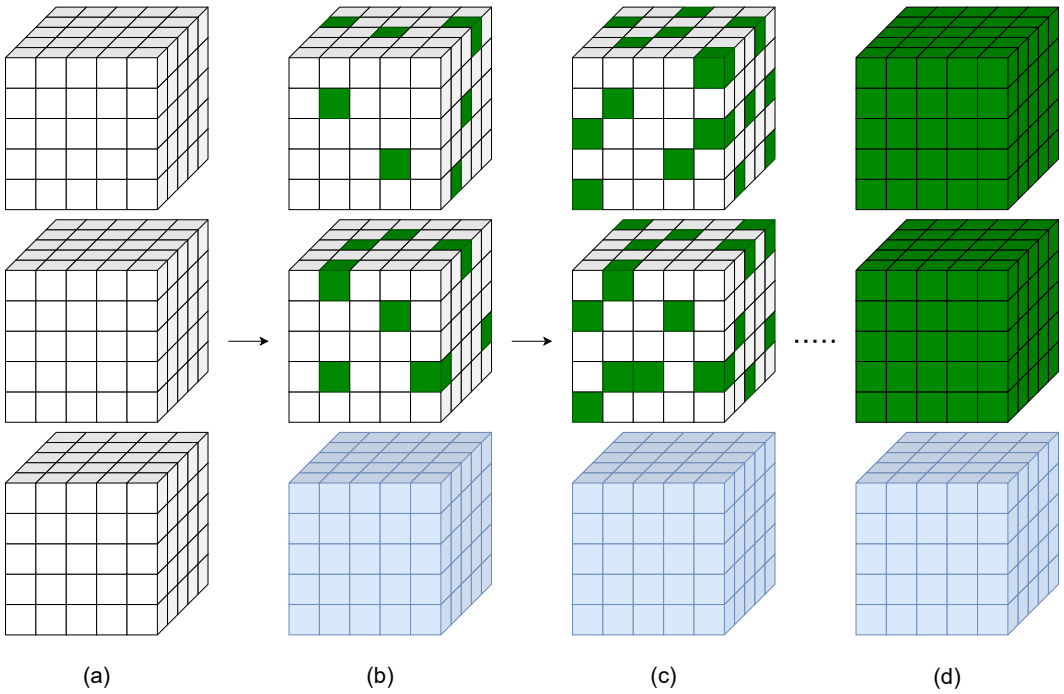

Figure 4.1.: Fixed Channel Proportion Quantization

4.3. Channel-wise Quantization

In this quantization strategy, we make filter partitions instead of weight partitions and quantize whole filters at each iteration. This helps in doing quantization in a structural manner as well. The proposed quantization strategy is mentioned in the Algorithm 3. To keep it simple, you can assume that $\mathbf{A}_l^{(1)}$ and $\mathbf{A}_l^{(2)}$ represents the filter partition instead of weight partition and $\mathbf{W}_l$ represents the whole filter bank for the l^{th} layer.

Algorithm 3: Channel-wise Quantization

input : X: the training data,
$\{\mathbf{W}_l : 1 \leq l \leq L\}$: the pre-trained full-precision CNN model,
$\{\sigma_1, \sigma_2, \cdots, \sigma_N\}$: the fraction of weights quantized at iterative steps

output: : $\{\widehat{\mathbf{W}}_l : 1 \leq l \leq L\}$:
the final low-precision model with the weights constrained to be either powers of two or zero

Initialize $\mathbf{A}_l^{(1)} \leftarrow \phi$, $\mathbf{A}_l^{(2)} \leftarrow \mathbf{W}_l$, $\mathbf{T}_l \leftarrow 1$, for $1 \leq l \leq L$
for $n = 1, 2, ..., N$ **do**
    Reset the base learning rate and the learning policy;
    for $l = 1, 2, ..., L$ **do**
        According to σ_n, create filter partitions and update $\mathbf{A}_l{}^{(1)}$, $\mathbf{A}_l{}^{(2)}$ and $\mathbf{T}_l$;
        Based on $\mathbf{A}_l^{(1)}$, determine $\mathbf{P}_l$
        Quantize the weights in $\mathbf{A}_l^{(1)}$
        Calculate feed-forward loss, and update weights in $\{\mathbf{A}_l^{(2)} : 1 \leq l \leq L\}$;
    end
end

Just like previous approach, this process is also illustrated in Figure 4.2 and two successive iterations are shown. (a) shows the weights (white blocks) before any quantization. (b) shows that the whole filters (green) are quantized at each iteration instead of random weights inside these filters. (c) shows the the next iteration.

4.4. Modified Network Architecture

This section focuses on the Architectures of Neural Network that were used and modified. As explained earlier, to achieving compression and enhanced performance, both weight and activations should be quantized. The INQ compression method only quantizes weights to low-bit precision. However, efficiency can be further enhanced by quantizing floating-point activations to low-precision fixed point-numbers. In this regard, STE layers were incorporated in the network architecture and the final architectures for both ALexNet and ResNet are presented in this chapter.

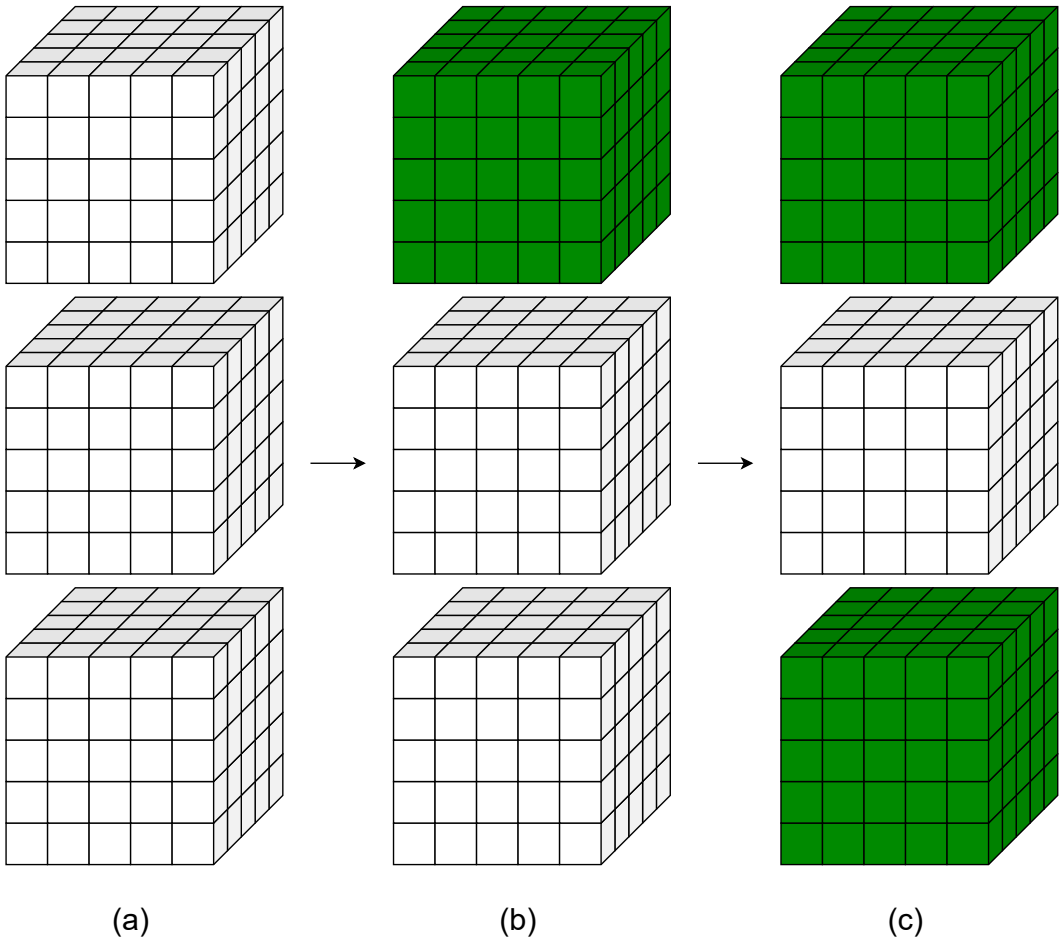

Figure 4.2.: Channel-wise Quantization

Figure 4.4 illustrates the incorporation of STE layers in AlexNet Architecture. For simplicity, max-pool layers are not shown. On the other hand, modified ResNet-18 architecture is shown in Figure 4.3.

Apart from that, we also incorportated this functionality where we can also synchronize different STE layers in the network if needed. It means that they can have same clipping range and we have a uniformity across different activation functions in the network. If we do not synchronize them, our quantization ranges might differ in different parts of the network.

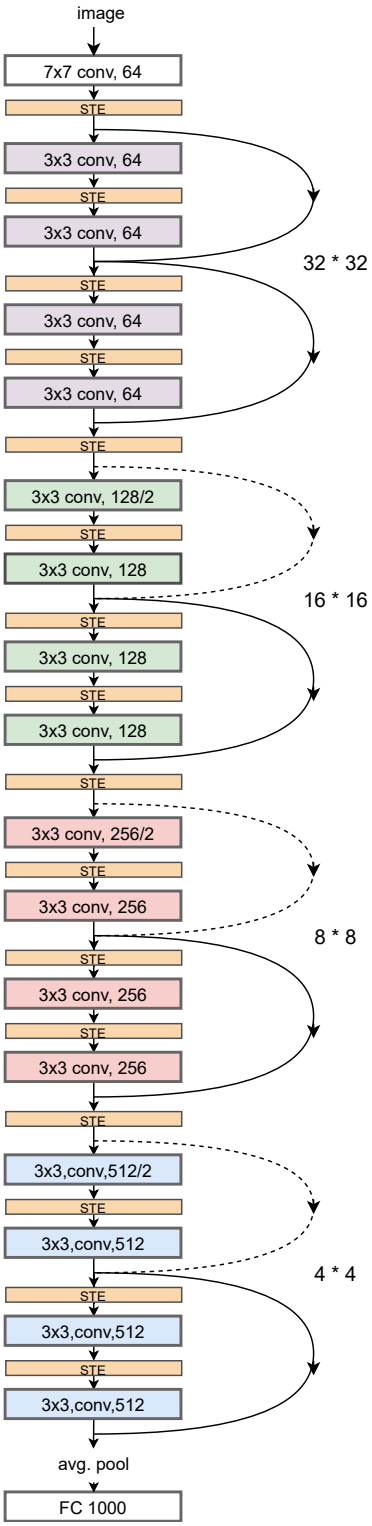

Figure 4.3.: Modified ResNet-18 Architecture

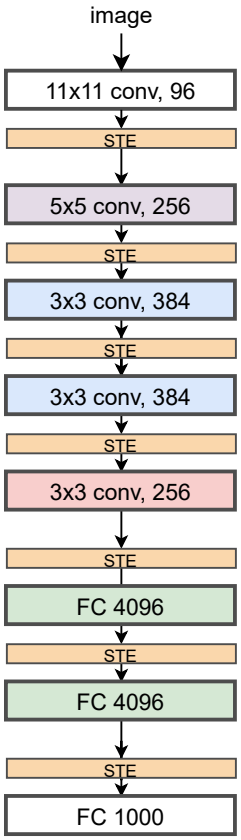

Figure 4.4.: Modified AlexNet Architecture

4.5. Partitioning Schemes

Two partition strategies for the weight quantization or the quantization of the whole filter were explored, namely *"low magnitude"* and *"high magnitude"*. The idea for selecting either weights or filters on the basis of these strategies is exactly the same. Vanilla INQ method uses a pruning-inspired strategy, which is what we call *"high magnitude"* for our experimentation. It considers that the weights with a larger magnitude are relatively more important than the smaller ones to form a low-precision base for the original CNN model. So, it always chooses the weights with the largest absolute values for the partition which will be quantized. Unlike random partition strategy, where weights are randomly chosen for each partition, *"low magnitude"* partition strategy selects the weight with the lowest absolute values for the partition which will be quantized.

As far as filter partitions are concerned for channel-wise quantization, the same two strategies were used as explained before. The absolute values of all the weights in a filter are added and then the filters are assigned to any partition based on the magnitude of the sum. In the case of the *"low-magnitude"* strategy for the channel-wise quantization, filters with the lowest absolute sum of the weights will be assigned to the partition which will be quantized, and vice versa.

Chapter 5

Results

This chapter presents and analyses all the results of the experiments conducted during the whole thesis work. The first section describes the general overview of all the experiments. In the next section, a brief description of our framework and used datasets is given. In the next part, baseline models were created and compared to the reported results of Zhou et al.'s work[18]. Moreover, a detailed analysis of quantization of activations were also performed. Along with that we also draw comparisons of the different partition strategies for the INQ method. Comparison of different structural quantization approaches that we used to achieve mixed-precision networks was also performed. Extensive testing is done on ResNet-18 architecture for an image classification task. It also presents the detailed analysis of quantizing the first and the last layers of the model as well. In the end, a brief sparsity analysis is also presented.

5.1. Overview

In general, we explored the intra-layer mixed-precision approaches based on INQ method. To start off with our experimentation, we first created the baseline using INQ method where we did 2-bits weight and 8-bits activation quantization. To quantify the effect of quantization of activations in a neural network, we ran some experiments without quantizing the activations. In this analysis, we found out that 8-bit quantization of activations does not have any significant effect on the accuracy of the network.

In the next phase, we explored different weight partitioning strategies for unstructured quantization using INQ method. We found out that *"high magnitude"* works best if we want to fully quantize the network as the accuracy loss rises progressively and settles at the end. The final accuracy is much better as compared to *"low magnitude"* based strategy. However, we observed that *"low magnitude"* based strategy outperforms the

"high magnitude" strategy if we partially quantize the network. To be specific, it can be used to quantize 90-95% of the weights with almost negligible loss in the accuracy. Nevertheless, the accuracy loss explodes at the very end and becomes even worse than *"high magnitude"* strategy if we fully quantize the network. Apart from that, we wanted to explore the impact of different partitioning schemes on the sparsity throughout the quantization process. We found out that network sparsity increases heavily in the beginning for *"low magnitude"* based strategy. So, if we take two different networks which are 90% quantized, the one quantized with the *"low magnitude"* strategy will have almost 10% more sparsity as compared to the one quantized with the *"high magnitude"* strategy. However, the sparsity becomes approximately same for both strategies if we quantize the network 100%.

As unstructured partially quantized network is harder to exploit in hardware, so we moved towards the more structured approaches. So, we evaluated our two proposed structured quantization methods for intra-layer mixed-precision weights based on INQ. So, only some channels are computed in full-precision while most of them are in 2-bits precision. Here we also explore two approaches. In the first one, we fix the fraction of channels that will stay in full-precision and perform the INQ on the rest of the channels. We call it *"Fixed Channel Proportion Quantization"*. In the second, instead of quantizing individual weights at each step, we quantize the whole filters to give it a more structured manner. We call this *"Channel-wise Quantization"*. Both the strategies were explained in detail in Section 4.2 and 4.3. They were evaluated using both partitioning schemes as explained earlier in Section 4.5, and we observed that *"Fixed Channel Proportion Quantization"* is better when we use "high magnitude" strategy for weight or filter partitioning and vice versa. If we quantize only 90% of the network using *"Fixed Channel Proportion Quantization"*, we can gain 1.5% more accuracy than fully quantized network using unstructured quantization based on INQ. Moreover, the proposed approach is easier to map on hardware.

In the next phase, we explored the impact of quantizing the first and the last layers of the network. We used only *"low magnitude"* weight partitioning scheme and evaluated this impact using unstructured quantization based on INQ. We found out that quantization of last layer has a much bigger impact on loss in accuracy as compared to the first one. In fact quantization of only the first layer does not have that significant effect on accuracy loss of the network.

In the end, we perform sparsity analysis in different layers of the network. We found out that initial layers of the network has much more sparasity after quantization and it keeps on decreasing as we go deeper in the network.

5.2. Experimental Setup

Although all the experiments conducted and presented in this chapter are unique in nature, there are certain elements that were common between all the experiments to make it a fair comparison.

5.2.1. Frameworks

Currently, one of the most popular deep learning frameworks is Pytorch, which is an open-source machine learning library primarily developed by Facebook's AI Research lab (FAIR)[29]. Therefore, all the code base was developed in Python and Pytorch. In addition to the realization of the hardware-friendly mixed-precision method itself, several networks were implemented and trained, including the ResNet-18 architecture, which served as a baseline for the experiments and adaptations for modified quantization strategies.

As far as hardware platforms for training the networks is concerned, we used two servers. One of which contains two Intel Xeon CPUs and four Nvidia GeForce GTX-1080 [30] GPUs with 12 GB of memory each and the other one contains two Intel Xeon CPUs and four Nvidia GeForce RTX-2080 [31] GPUs with 12 GB of memory each. These were the majorly used computational resources. However, training DNNs do not only need high computational and memory resources but an enormous amount of data as well. So, the datasets used for training and validation of these models are explained in the next section.

5.2.2. Benchmarking Datasets

When it comes to deep neural networks, high-quality and large datasets are needed to train robust machine learning models. However, training machine learning model on these large datasets take a long time. In this regard, it makes sense to start with a relatively smaller dataset for prototyping and experimenting with neural networks.

So, we chose the CIFAR-10 dataset [32] initially for evaluating models as it saves a lot of time. This dataset contains 60,000 color images, assigned to 10 different classes, ranging from various animals to means of transport. Once we were confident with our approach, we did an evaluation on the ImageNet dataset, which consists of 1,3 million color images, categorized into 1000 different classes. It is considered one of the most important benchmarking datasets for visual object classification tasks.

5.3. Baseline

In this section, we only focus on the image classification task and the ImageNet dataset served for benchmarking purposes. Extensive experimentation was done using ResNet-18 architecture, which is one of the famous network architectures. However, we did some initial experimentation on CIFAR-10 to assess the whole approach and save us some time. To make the time even shorter for our training, we chose AlexNet architecture.

AlexNet and CIFAR-10

After creating the AlexNet baseline for the CIFAR-10 dataset, 8 and 2 bits weight quantization was performed to test our developed framework. As far as hyperparameters are concerned, we chose vanilla stochastic gradient descent (SGD) as an optimizer and momentum as 0.9. The initial learning rate *(lr)* was set to 0.01 and we divide the *lr* by 10 when the validation error rate stopped improving with the current learning rate. Moreover, the chosen batch size was 128 and we trained the network for 90 epochs to create the full-precision baseline. These hyperparameters are also presented in tabular form in Appendix in Table B.1.

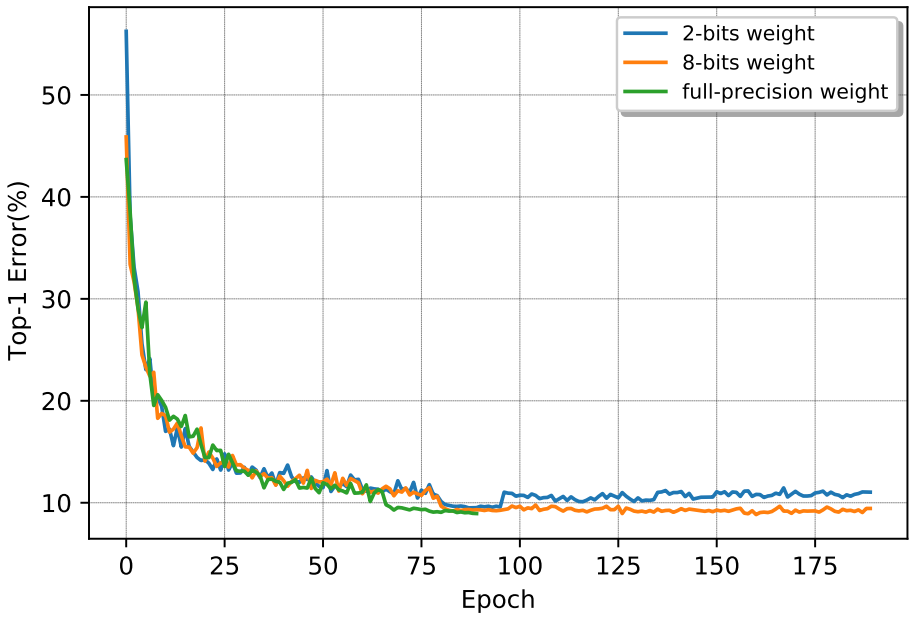

Figure 5.1.: Training on CIFAR-10 dataset

For quantization of the weights, we used the *"high magnitude"* strategy for weight partitioning which is explained in Section 4.5. Moreover, the accumulated portions of quan-

tized weights at iterative steps were set as {0.2, 0.4, 0.6, 0.7, 0.8, 0.9, 1} for both 8 bits and 2 bits ternary models. Number of epochs after each quantization steps were set as {24,15,15,15,10,10,5} for both 8 bits and 2 bits respectively. Moreover, the *lr* is increased back to initial learning after the quantization and when the validation error rate stopped improving with the current *lr*, we reduce it by a factor of 10. This heuristic was used in all the experiments presented in this chapter. For these initial experiments on CIFAR-10 dataset, the top-1 error(%) on validation set during the whole training is illustrated in Figure 5.1 and the results are shown in Table 5.1.

Table 5.1.: Benchmarking results on CIFAR-10

Network	**Weights**	**Activations**	**Top-1 error**
AlexNet	floating point	floating point	8.94%
AlexNet	8 bits	8 bits	9.44%
AlexNet	2 bits	8 bits	11.04%

In addition to that, all the experiments and results mentioned in this chapter also quantize the activations to 8 bits unless mentioned. However, the first and the last layer of the network are not quantized for the initial experiments. Later in the next section, we also observed the effects of quantizing only last, only first, or both laters on the performance of the model. It can be observed that there is only ∼1% drop in the accuracy for the CIFAR-10 dataset after quantizing weights and activation to 2 and 8 bits respectively.

ResNet-18 and ImageNet

For the second set of experiments, we created the baseline models for the ResNet-18 model on the ImageNet dataset. As ResNet-18 is a much bigger network as compared to AlexNet, we did not train the full-precision model, instead, we used the trained weights from Facebook's official repository[33]. For this set of experiments, the batch size was fixed to 256. However, the *lr*, optimizer and momentum were similar to the experiments done on CIFAR-10 dataset. All the hyperparamters for these set of experiments are presented in tabular form in Table B.2 in the Appendix. The results of these experiments are presented in Table 5.2.

Table 5.2.: Benchmarking results on ImageNet

Network	**Weights**	**Activations**	**Top-1 error**
ResNet-18	full-precision	full-precision	31.38%
ResNet-18	8 bits	8 bits	30.61%
ResNet-18	2 bits	8 bits	37.74%
ResNet-18 - INQ[1]	2 bits	full-precision	33.98%

For 8 bits weight quantization, we set the accumulated portions of quantized weights at iterative steps as {0.2, 0.4, 0.6, 0.7, 0.8, 0.9, 1} while for 2 bits ternary models, we set it as {0.2, 0.4, 0.6, 0.7, 0.8, 0.9, 0.9125,0.925,0.9375,0.95,0.9625,0.975,0.9875,1}. We used a very conservative schedule with very small steps at the end to see if we can have any reduction in the error. There is another reason for doing these small steps at the very end, which is explained in section 5.5. However, it is quite evident from Figure 5.2 that 8-bits weights and activations incur an almost negligible loss in the accuracy when we leave the first and the last layer unquantized.

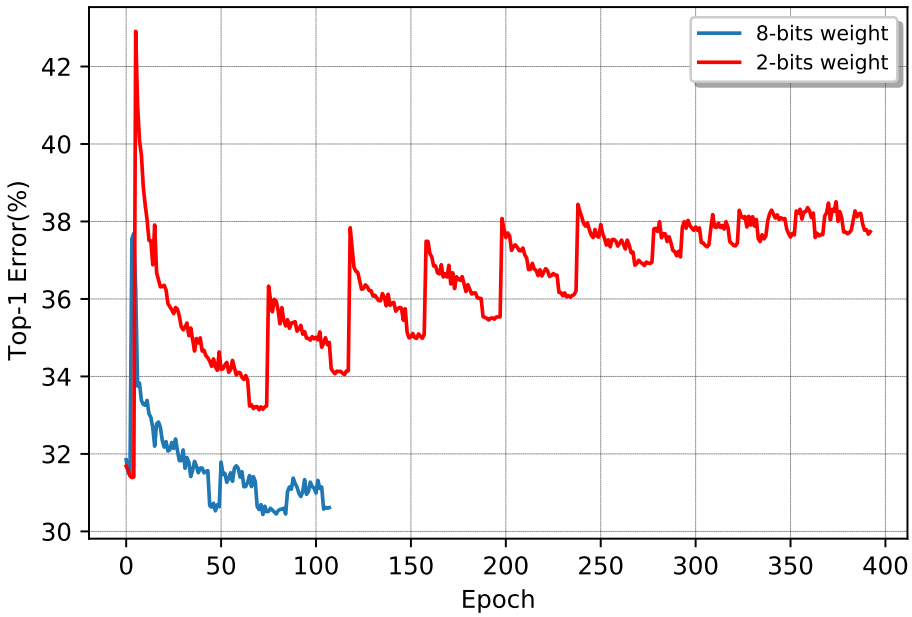

Figure 5.2.: Training on ImageNet dataset

The training on the validation set is illustrated in Figure 5.2, where we quantize the weights to 2 bits and 8 bits respectively. You can see that we trained the ternary weights network for a much longer time as it takes more time to converge after each quantization iteration. Moreover, ternary weight quantization has a much bigger impact when we evaluate the network on ImageNet dataset as compared to CIFAR-10 dataset. We get a significant decrease in the accuracy when we quantize the network for ImageNet dataset.

We observed that our results has a significant difference from Zhou et al's work[18] after doing 8 bit quantization of activations. Table 5.2 shows the difference between our

[1]These are the numbers that were presented in the Zhou et al.'s work[18]. Moreover, first and the last layers of the network are quantized as well.

baseline and the reported results by Zhou et al.[18]. In this regard, the next step we performed was to quantize the network without quantizing activations.

Fixed Hyperparameters

For all the experiments presented in the next sections, we did 2-bits weight quantization and the hyper-parameters mentioned in the previous section were used. Moreover, we quantized the activations to 8-bits fixed-point numbers unless it is explicitly mentioned. We also fixed the quantization schedule which is shown in Figure 5.3. All the fixed hyperparameters are given in a tabular form in the Appendix in Table B.3. It can be seen that it takes ∼3-4 times more epochs as it takes to train a ResNet-18 netowrk from scratch.

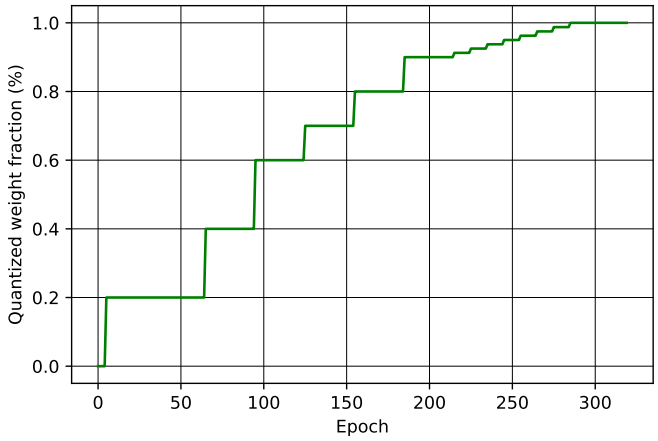

Figure 5.3.: Quantization Schedule

5.4. Unquantized Activations

To observe the impact of 8-bit activation quantization on the accuracy, we performed some experiments without quantizing the activations in the network. The results of these experiments are illustrated in Table 5.3. To be specific, the accuracy achieved after 2-bits weight and 8-bits activation quantization without quantizing the first and the last layer was approximately 4% lower than the accuracy reported by Zhou et al.[18].

It is quite evident from Table 5.3 we were not able to reproduce the results reported by Zhou et al [18]. However, we noticed that 8-bits quantization of activations have

[2]F: First, L: Last

Table 5.3.: Impact of quantizing the activations

Quantization	Layers	Partitioning	Activations	Top -1 error (%)		
				0.90	**0.95**	**1.0**
Unstructured	All	low mag.	full-prec.	33.46	33.84	64.96
Unstructured	All	high mag.	full-prec.	43.11	44.78	44.99
Unstructured	All except F&L	low mag.	full-prec.	32.07	32.47	48.06
Unstructured	All except F&L	high mag.	full-prec.	37.24	37.8	37.84
Unstructured	All	low mag.	8-bits	33.57	34.18	64.76
Unstructured	All	high mag.	8-bits	43.6	45.24	45.67
Unstructured	All except F&L[2]	low mag.	8-bits	31.99	32.47	49.93
Unstructured	All except F&L	high mag.	8-bits	36.95	37.63	37.74

negligible effect on the accuracy. Here we evaluated two different approaches for weight partitioning as well which are explained in the Section 4.5.

5.5. Partition Strategy

Top-1 error was monitored throughout the training for unstructured 2-bits weight and 8-bits activation quantization using vanila INQ method for both the partitioning strategies as explained in section 4.5. The observed plot is shown in Figure 5.4. One important observation made from these plots was that *"high magnitude"* based strategy produce a greater loss in the accuracy in the initial quantization steps as compared to *"low magnitude"* based strategy. The latter works really well until the very end and incurs minimum accuracy loss.

It is quite evident from Figure 5.4 that there is a huge spike at the very end when we quantize the remaining 3-4 percent of the network weights using *"low magnitude"* strategy. It can be seen that the network was never able to recover from this as all the weights were already quantized. We explored this fact and analyzed the weight distributions at several stages of training. An example layer from the network was selected and these weight distributions were plotted for both partitioning strategies using Tensorboard[34] and are depicted in Figure 5.5 and 5.6. Moreover, it was observed that all the layers have similar weight distibutions at each iteration. Each line on these charts represents a percentile in the distribution over the data. For example, the top line shows how the maximum value has changed over time, and the line in the middle shows how the median has changed. Reading from bottom to top, the lines have the following meaning: [minimum, 7%, 16%, 31%, 50%, 69%, 84%, 93%, maximum]. These percentiles can also be interpreted as standard deviation boundaries on a normal distribution i.e [maximum, μ + 1.5σ, μ + σ, μ + 0.5σ, μ, μ - 0.5σ, μ - σ, μ - 1.5σ, minimum]. Therefore, the colored

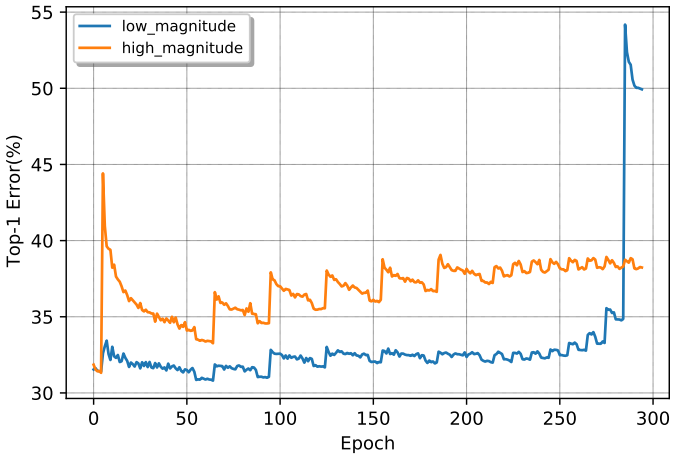

Figure 5.4.: Partitioning strategies

regions, read from inside to outside, have widths $[\sigma, 2\sigma, 3\sigma]$ respectively. On the X-axis, we have time and on the Y-axis we have the magnitude.

These weight distributions made a lot of clarity in understanding the huge spike in accuracy loss at the very end of *"low magnitude"* based strategy. You can see that there is a minimal change in the weight distribution in the initial steps. So, negligible accuracy loss is observed. However, when the large values are clamped to -1 and 1 at the very last iteration, there is a huge quantization error and accuracy loss as well. Moreover, all the weights are in 2 bits after this iteration, so it is difficult to recover network accuracy afterward. On the other hand, we quantize weights with higher magnitude first in the *"high magnitude"*, so the network can still recover from the accuracy loss as there are still some full-weights in the network. That is why the weight distribution in *"high magnitude"* based strategy changes significantly in the beginning and negligibly in the last iterations.

Table 5.4 summarizes these experiments and report top-1 error at 90, 95 and 100 percent quantized weights in the network.

To sum it up, we have a negligible loss in accuracy if we quantize 90% weights in the network. Moreover, we can quantize 95% weights with only ∼1% accuracy reduction. We tried to see how much quantization we can achieve without a significant loss in accuracy and observed that ∼98% weight quantization can be achieved with only ∼1.8% reduction in accuracy.

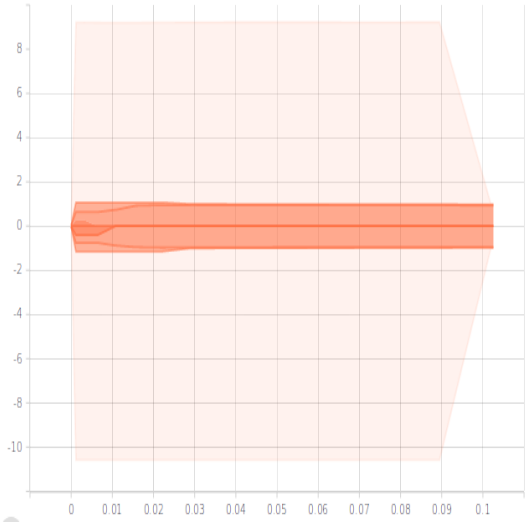

Figure 5.5.: *"low magnitude"* strategy

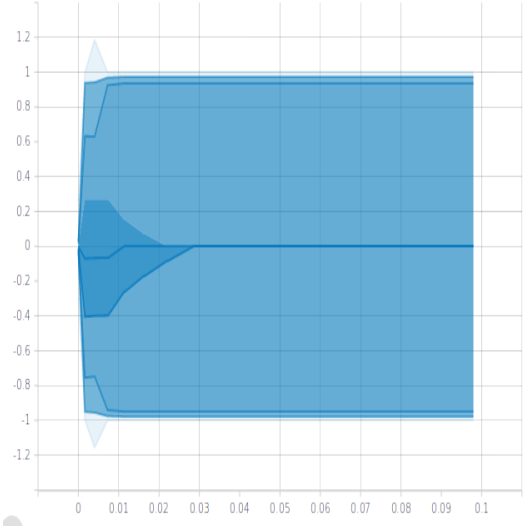

Figure 5.6.: *"high magnitude"* strategy

Table 5.4.: Comparison of two different strategies for weight partition.

Network	**Partition-strategy**	**Bit-width (W/A)**	**Top-1 error (%)**		
			0.9	0.95	1
ResNet-18	low magnitude	2/8	31.99	32.47	49.93
ResNet-18	high magnitude	2/8	36.95	37.63	37.74

5.6. Mixed-precision Quantization

5.6.1. Unstructured

We saw before that if we use *"low magnitude"* weight partitioning strategy, we can achieve 95% quantized weights without a significant loss in accuracy for unstructured quantization. However, we need a very complex hardware to exploit this method. Therefore, a more structural approach is needed which is easier to map on to a hardware.

5.6.2. Structured

In this section, we will demonstrate the results of our proposed structured quantization methods. These approaches are explained in section 4.2 ann 4.3. Moreover, we quantized weights and activations to 2 and 8 bits respectively. So, it is not explicitly mentioned in the tables.

Channel-wise Quantization

In the channel-wise quantization, we quantize whole filters based on the absolute sums of the weights in each filter. The proposed strategy was compared with unstructured quantization. As the training of each model takes a very long time, so only the models with all the layers quantized and the one where the first and the last layers were left in full-precision, were trained. The results are illustrated in Table 5.5. Experiments with quantizing the first and the last layers of the network using channel-wise quantization were not performed as they did not provide very promising results in the first place.

Table 5.5.: Structural vs Unstructural quantization

Quantization	**Layers**	**Partitioning**	**Top -1 error (%)**		
			0.90	**0.95**	**1.0**
Unstructured	All	low magnitude	33.57	34.18	64.76
Unstructured	All	high magnitude	43.6	45.24	45.67
Unstructured	All except F&L	low magnitude	31.99	32.47	49.93
Unstructured	All except F&L	high magnitude	36.95	37.63	37.74
Channel-wise	All except F&L	low magnitude	38.56	39.74	41.03
Channel-wise	All except F&L	high magnitude	39.98	39.74	40.78
Fixed Channel	All	low magnitude	34.47	34.88	54.73
Fixed Channel	All	high magnitude	43.07	43.67	43.37
Fixed Channel	All except F&L	low magnitude	31.82	32.1	42.12
Fixed Channel	All except F&L	high magnitude	36.28	37.3	36.37

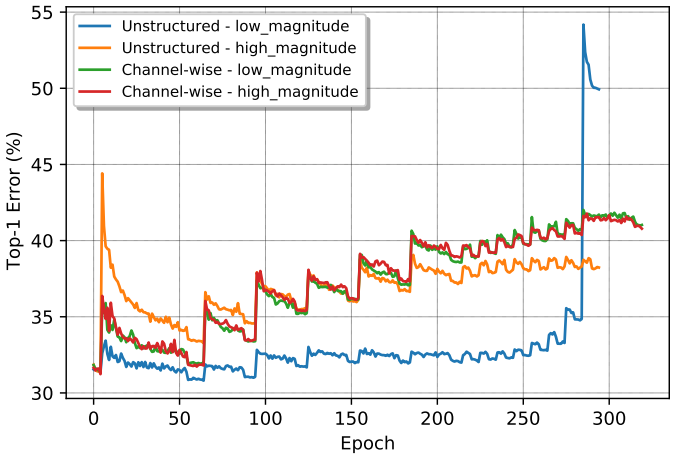

Figure 5.7.: Channel-wise vs Unstructured Quantization

In addition to that, Figure 5.7 shows the whole training process for some of the experiments listed in Table 5.5, where we observe how accuracy is affected when we quantize the network using channel-wise quantization and unstructured quantization. As whole filters are quantized in channel-wise quantization and granularity becomes more coarse, we did not achieve a very good final accuracy. Also, the weights quantized at each step are a mixture of low magnitude and high magnitude weights regardless of weight partitioning strategy, therefore we do not see a big difference in the training plots of *"low magnitude"* and *"high magnitude"* based channel-wise quantization. The final accuracy after 100% quantization is slightly better for *"high magnitude"* based strategy and *"low magnitude"* based strategy works slightly better for partial quantization. However, there is no significant differences.

Fixed Channel Proportion Quantization

In this method, we fix some randomly picked channels in the beginning and they are trained in full precision throughout the training. On the other channels, the INQ strategy is performed.

The results for this method are also illustrated in Table 5.5 and compared with unstructured and channel-wise quantization. For all three different quantization strategies, the training plots for the experiments where we do not quantize the first and the last layers are illustrated in Figure 5.8. We can see that the top-1 accuracy for *"Fixed Channel Proportion"* quantization throughout the training, follows almost the same pattern as unstructured quantization. However, the final accuracy is better as we still have 10% weights in full-precision.

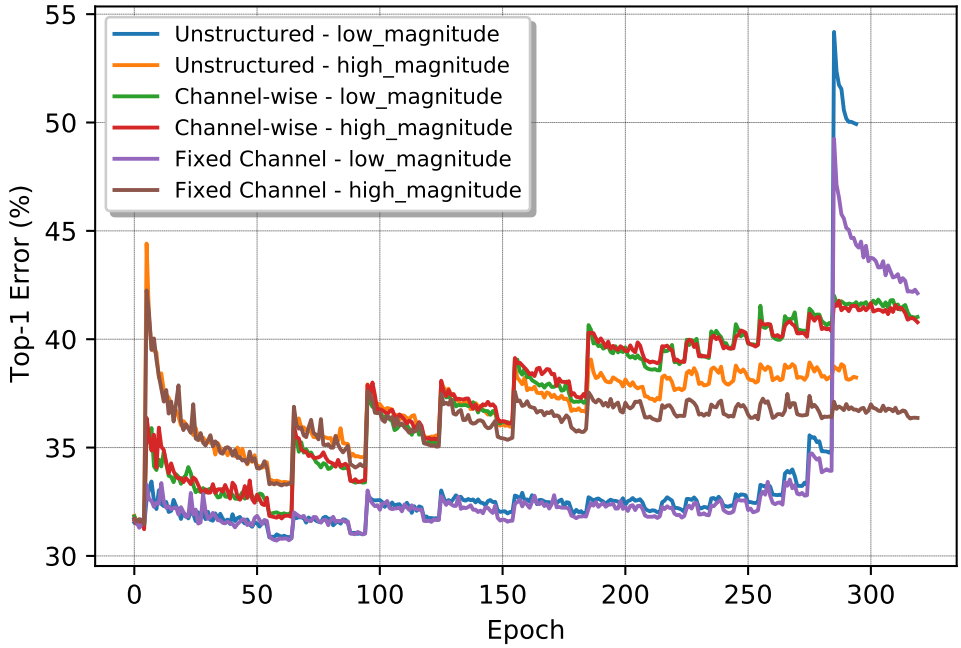

Figure 5.8.: Unstructured vs Structured Quantization

Moreover, *"low magnitude"* weight partitioning strategy works best in this case as well for partial quantization. However, the final accuracy loss still explodes in the very end. The remaining full-precision weights manage to recover it somehow but still the accuracy is worse than the final accuracy achieved by *"high magnitude"* weight partitioning strategy. **Nevertheless, we got an improvement of ∼1.5% accuracy from the ternary weight baseline model if we use *"high magnitude"* weight partitioning strategy.**

5.7. Quantizing First and Last Layer

After the structured quantization strategies, research was done on quantizing the first and the last layers of any given neural network. In most of the research methods proposed in the past for network quantization, the first and the last layers were left untouched as it incurs a huge loss in accuracy. However, nobody has ever presented any quantitative analysis of quantizing the first and the layers of the network. So, a detailed analysis was performed in this regard in the course of this thesis, and some conclusions were drawn. All the hyperparameters were kept same as in the previous experiments.

The Table 5.6 summarizes the results of this comparison. Just like previous experiments,

Table 5.6.: Comparison of quantizing the first(F) and the last(L) layer

Network	Layers Quantized	partitioning	Top - 1 error (%)		
			0.90	0.95	1.0
ResNet-18	All	low magnitude	33.57	34.18	64.76
ResNet-18	All except F	low magnitude	35.04	35.73	62.83
ResNet-18	All except L	low magnitude	32.22	32.55	51.77
ResNet-18	All except F & L	low magnitude	31.99	32.47	49.93
ResNet-18	All	high magnitude	43.6	45.24	45.67
ResNet-18	All except F&L	high magnitude	36.95	37.63	37.74

top - 1 error is reported at different quantization fractions. To make it clear, we quantize all the network layers along with the specified layer.

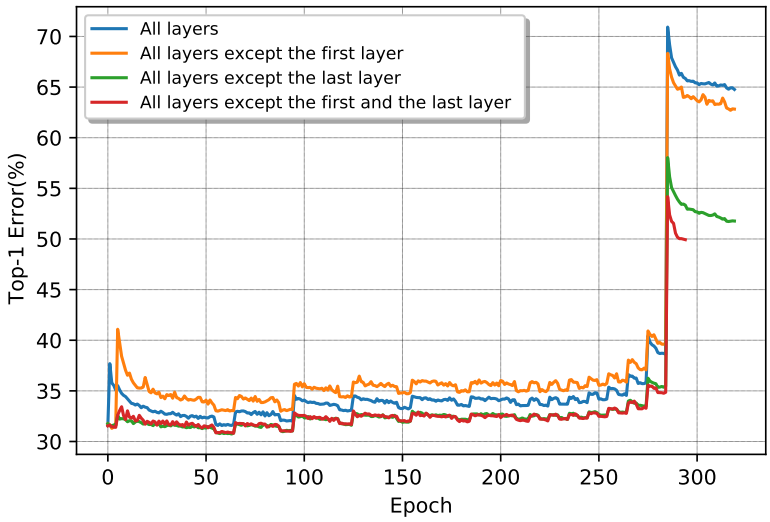

Figure 5.9.: Comparison of quantizing the first and the last layer

Moreover, Figure 5.9 depicts the whole training for all these experiments where it shows how the accuracy is affected when we quantize the first layer, last layer, or both along with all the other layers of the network for *"low magnitude"* weight partitioning. We also ran these experiments with more epochs after the final quantization steps. However, it did not reduce the error significantly.

The first conclusion that can be drawn from the Figure 5.9 is that quantizing both the first and the last layers reduces the accuracy significantly, no matter which weight partitioning we use. So, it can be said that networks become uncompetetive if we quantize both the

first and the last layer for *"high magnitude"* weight partitioning scheme. However, we can still achieve 95% quantization in an unstructured way with a loss of approximately ∼2.8% accuracy from full-precision network. Another thing worth mentioning here is that the quantization of all the layers except the first one incurs a relatively greater loss in accuracy as compared to quantizing the whole network except the last layer.

5.8. Sparsity analysis

The last analysis that was done on the experiments was about the sparsity in the network after the quantization. For all the experiments that were performed, it was observed that sparsity is less in the deeper layers as compared to the initial layers of the network. To illustrate this even better, the experiment performed to make a ResNet-18 ternary weights baseline was taken as an example and the fraction of sparsed weights in each layer was calculated. This is shown in a pictorial fashion in Figure 5.10 to provide a better understanding.

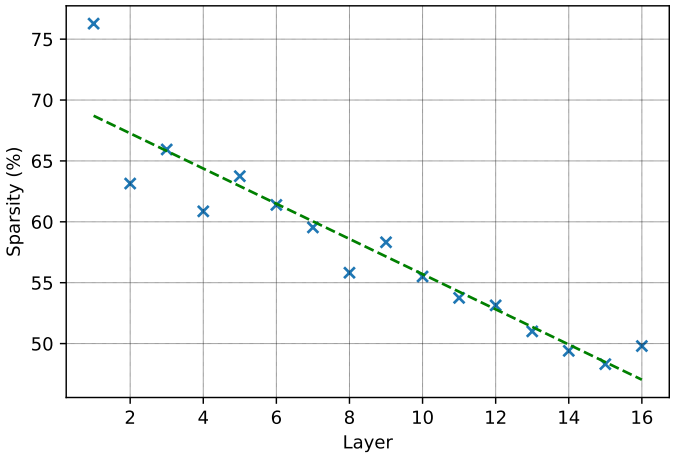

Figure 5.10.: Percentage of sparsed weights in each layer

Moreover, after 2-bits weight quantization of all the layers, almost half of the weights become zero. For that purpose, the total sparsity in the layers that we quantized, was monitored for the same experiment throughout the training. Figure 5.11 shows the sparsity in the network at each quantization step. However, the detailed comparison of how the sparsity in the network evolves throughout the training is presented in section 5.5 as it is highly dependent on the partition strategy for the INQ method.

Last sparsity analysis that we did was on the partitioning schemes and observed how sparsity evolves throughout the training. Unstructured 2-bits weight and 8-bits activation quantization using vanila INQ method was chosen again. We conducted two experiments

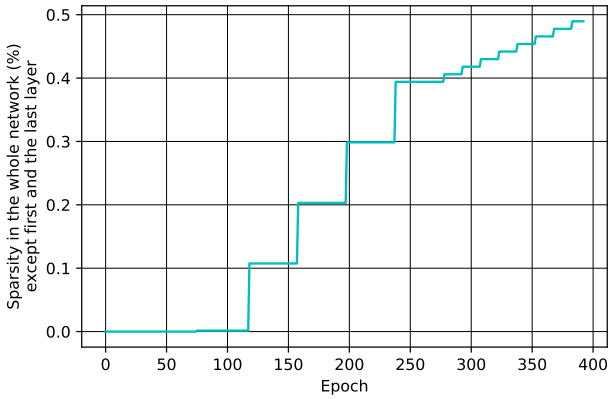

Figure 5.11.: Sparsity after each quantization step

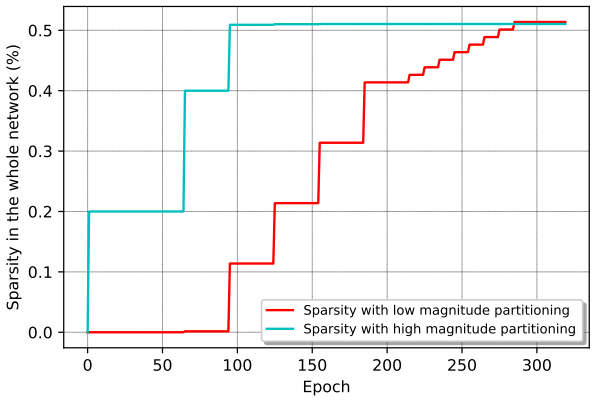

Figure 5.12.: Sparsity comparison for different partitioning schemes

using *"low magnitude"* and *"high magnitude"* partioning strategies and the comparison is illustrated in Figure 5.12. It can be seen that using the low magnitude strategy for quantization partition, the sparsity in the network increases heavily in the initial steps and it becomes almost constant after quantizing approximately 60% of the weights. On the other hand, negligible sparsity is achieved in the initial steps while using a high magnitude strategy for quantization partition. Nevertheless, the final sparsity in the network at the end is approximately the same for both strategies.

Chapter 6

Conclusion and Future Work

Nowadays, visual data is everywhere. A lot of research has been conducted on how the video and image data is increasing and will continue to increase in the near future. This visual data can be used to do wonders. CNNs in particular have proved themselves in solving many computer vision tasks using this visual data. However, it is not an easy task to deploy these CNNs in reality. This research focuses on a reduction in model complexity for their practical application in embedded systems, mobile, and Internet-of-Things (IoT) devices since they do not have extensive power/computational resources on board. To meet these necessities, various possibilities for partially quantizing CNNs were explored within the scope of this thesis.

6.1. Conclusion

The prime goal of this work was to impair accuracy as little as possible and come up with mixed-precision approaches to quantize the network. In this regard, we propose "Hardware-friendly Mixed-precision Neural Networks", which is a trade-off between fully quantized and full-precision models. The target bit-precision was 2 and 8 bits for the network weights while the activations were quantized to 8 bits. We analyzed that quantizing the activations to 8-bits precision does not really affect the accuracy of the network.

Using the INQ method as a baseline, exploration of different weight and filter partitioning methods based on the magnitude of the weights was done. For the weight partitioning, the *"low magnitude"* based strategy works best until the network is almost ∼98% quantized. We can quantize ∼98% weights in the network with only ∼1.8% decrease in the accuracy. However, there is a huge drop in accuracy in the end which results in a very bad final prediction accuracy. This tells that rest of the 1-2% full-precision weights at the very end are really important to retain the accuracy. So, the *high magnitude* based strategy

can be considered as the best strategy for weight partitioning. On the other hand, *low magnitude* based scheme for filter partitioning was slightly better. While quantizing the network with these different schemes, there is a degradation after each step and network must be able to adapt to the changes. As far as the sparsity is concerned, both the strategies incur approximately the same amount of sparsity in the network.

The two quantization methods were used to achieve hardware-friendly mixed-precision networks and do quantization in more structured way. *Channel-wise Quantization* did not achieve good results as we quantize the whole filter instead of individual weights inside the filters and structural granularity becomes more coarse. On the other hand, *Fixed Channel Proportion Quantization* where we quantize the 90% channels only, give us $\sim$1.5% increase in accuracy. Moreover, this proposed method could be relatively much simpler to map onto the hardware. However, we can clearly see that when we try to enforce the structure, partial quantization is less advantageous.

While quantizing the network, sparsity analysis was performed on the network and it was observed that the initial layers of the network are relatively more sparse than the deeper layers. As far as the quantization of the first and the last layers of the network is concerned, detailed analysis provides enough evidence that quantization of the last layer incurs a significantly higher loss in prediction accuracy as compared to the quantization of the first layer. Moreover, if we quantize the first and the last layer of the networks as well, it becomes uncompetitive.

6.2. Future Work

Even after this extensive analysis of mixed-precision hardware-friendly neural networks using the INQ method as a reference, there is still room for a lot of improvement. Especially, *"Fixed Channel Proportion"* shows some good results as compared to the unstructured quantization baseline. In the future, exploration can be done on creating the initial channel partition based on some strategy. In this work, channels were picked randomly for the channel partition. Using the *"low magnitude"* and *"high magnitude"* based strategy for that purpose could be a good starting point.

Moreover, as it was observed that we were not able to reproduce the work of Zhou et al.[18], so switching to some other algorithm for network quantization would be recommended as well. Another important aspect to consider is the robustness of our model against the quantized fraction of weights in the network as it is also important for safety-critical applications e.g Autonomous driving, etc. Furthermore, combining the quantization with other model optimization techniques which are orthogonal to this work can help in deploying efficient CNNs in near future.

Appendix A

Declaration of Originality

Please find the declaration of originality on the next page.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

HARDWARE-FRIENDLY MIXED-PRECISION NEURAL NETWORKS

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
JAVED	SAQIB

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 05.03.2021	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Appendix B

Hyperparameters

Table B.1.: Hyperparameters configuration for AlexNet baseline on CIFAR-10.

Hyperparameter	**Config**
base learning rate	0.01
batch size	128
optimizer	SGD
momentum	0.9
total epochs	90
Number of epochs after each quantization steps	{24,15,15,15,10,10,5}
accumulated portions of quantized weights at iterative steps	{0.2, 0.4, 0.6, 0.7, 0.8, 0.9, 1}
weight partitioning scheme	high magnitude

Table B.2.: Hyperparameters configuration for ResNet-18 baseline on ImageNet.

Hyperparameter	**Config**
base learning rate	0.01
batch size	256
optimizer	SGD
momentum	0.9
total epochs (8-bits)	108
total epochs (2-bits)	392
weight partitioning scheme	high magnitude
Number of epochs after each quantization steps (8-bits)	{24,15,15,15,10,10,11}
Number of epochs after each quantization steps (2-bits)	{70,43,40,40,40,40,15,15,15,15,15,15,15,15}
accumulated portions of quantized weights at iterative steps (8-bits)	{0.2, 0.4, 0.6, 0.7, 0.8, 0.9, 1}
accumulated portions of quantized weights at iterative steps (2-bits)	{0.2, 0.4, 0.6, 0.7, 0.8, 0.9, 0.9125, 0.925,0.9375,0.95,0.9625,0.975,0.9875,1}

Table B.3.: Fixed Hyperparameters configuration after the baseline.

Hyperparameter	**Config**
base learning rate	0.01
batch size	256
optimizer	SGD
momentum	0.9
total epochs	320
Number of epochs after each quantization steps	{60,30,30,30,30,30,10,10,10,10,10,10,10,10}
accumulated portions of quantized weights at iterative steps (2-bits)	{0.2, 0.4, 0.6, 0.7, 0.8, 0.9, 0.9125, 0.925,0.9375,0.95,0.9625,0.975,0.9875,1}

Bibliography

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[2] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[3] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[6] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 1–13.

[7] M. Denil, B. Shakibi, L. Dinh, N. De Freitas *et al.*, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems*, 2013, pp. 2148–2156.

[8] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision

weights and activations," *CoRR*, vol. abs/1609.07061, 2016. [Online]. Available: http://arxiv.org/abs/1609.07061

[9] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015.

[10] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531

[11] Sun-qian. Applications of deep learning in relativistic hydrodynamics. [Online]. Available: https://indico.cern.ch/event/689516/contributions/3028020/attachments/1680198/2699102/2018-06-DeepLearning-Song.pdf

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[13] G. W. Lindsay, "Convolutional neural networks as a model of the visual system: Past, present, and future," *Journal of Cognitive Neuroscience*, p. 1–15, Feb 2020. [Online]. Available: http://dx.doi.org/10.1162/jocn_a_01544

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[15] L. Hauenschild, "Autoencoder-based low-rank filter-sharing for effi-cient convolutional neural networks," Master's thesis, Technical University of Munich, 2019.

[16] C. S. Wiki. Max-pooling. [Online]. Available: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

[17] F.-F. Li, J. Johnson, and S. Yeung. (2018) Cs231n linear classification. [Online]. Available: http://cs231n.github.io/linear-classify/

[18] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," 2017.

[19] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[20] A. Khvostikov, K. Aderghal, J. Benois-Pineau, A. Krylov, and G. Catheline, "3d cnn-based classification using smri and md-dti images for alzheimer disease studies," 01 2018.

[21] Sun-qian. Resnet-18 implements cifar-10 image classification pytorch. [Online]. Available: https://www.programmersought.com/article/68543552068/

[22] H. Kim, "Residual networks for tiny imagenet," http://cs231n.stanford.edu/reports/2016/pdfs/411_Report.pdf.

[23] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013.

[24] L. Mao, "Quantization for neural networks," https://leimao.github.io/article/Neural-Networks-Quantization/.

[25] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," 2016.

[26] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016.

[27] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," 2016.

[28] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," 2016.

[29] "Pytorch," https://ai.facebook.com/tools/pytorch/.

[30] "Geforce 10 series graphics cards - nvidia," https://www.nvidia.com/en-us/geforce/10-series/.

[31] "Geforce rtx 2080 graphics card | nvidia," https://www.nvidia.com/en-us/geforce/graphics-cards/rtx-2080/.

[32] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *http://www.cs.toronto.edu/kriz/cifar.html*, 2014.

[33] "Facebook's resnet archive," https://github.com/facebookarchive/fb.resnet.torch/tree/master/pretrained.

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

[35] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, jan 2015, dOI: 10.1007/s11263-014-0733-5.

[36] T. P. Morgan. (2020) Google teaches ai to play the game of chip design. [Online]. Available: https://fullstackfeed.com/google-teaches-ai-to-play-the-game-of-chip-design/

[37] M. Elgendy, *Deep Learning for Vision Systems*. Manning Publications, 2020.

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

[39] Z. Cao, G. Hidalgo, T. Simon, S. Wei, and Y. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," *CoRR*, vol. abs/1812.08008, 2018. [Online]. Available: http://arxiv.org/abs/1812.08008

[40] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed precision quantization of convnets via differentiable neural architecture search," *CoRR*, vol. abs/1812.00090, 2018. [Online]. Available: http://arxiv.org/abs/1812.00090

[41] B. Zhang, A. Davoodi, and Y. H. Hu, "Chapr: Efficient inference of cnns via channel pruning," in *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, 2020, pp. 1–6.

[42] S. Han, H. Mao, and W. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *arXiv: Computer Vision and Pattern Recognition*, 2016.