Mixed Low-bit Quantization for Model Compression with Layer Importance and Gradient Estimations

by

Hongyang Liu

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science
University of Alberta

Abstract

Deep neural networks (DNNs) have been widely used in the modern world in recent years. However, due to the substantial memory consumption and high computational power use of DNNs, deploying them on devices with limited resources is challenging. Model compression methods can provide us with a remedy here. Among those techniques, neural network quantization has achieved a high compression rate using low bit-width representation of weights and activations while maintaining the accuracy of the high-precision original network. However, mixed precision (per-layer bit-width precision) quantization requires careful tuning to maintain accuracy while achieving further compression and higher granularity than fixed precision quantization. In this thesis, We propose an accuracy-aware criterion to quantify the layer's importance rank. Our method applies imprinting per layer, which acts as a proxy module for accuracy estimation in an efficient way. We rank the layers based on the accuracy gain from previous modules and iteratively quantize those with less accuracy gain. Previous mixed-precision methods either rely on expensive search techniques such as reinforcement learning (RL) or end-to-end optimization with a lack of interpretation to the quantization configuration scheme. Our method is a one-shot, efficient, accuracy-aware information estimation and thus draws better interpretability to the selected bit-width configuration. We have also pointed out the problem of the Straight-Through Estimator (STE), which is commonly used for gradients estimation in the quantization field. We've discussed some ways to address the problem of using STE.

Preface

This thesis is an original work by 'Hongyang Liu'. Parts of the thesis have been published as Hongyang Liu, Sara Elkerdawy, Nilanjan Ray, Mostafa Elhoushi, "Layer Importance Estimation With Imprinting for Neural Network Quantization", in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2021, pp. 2408-2417. [1] All works are original.

"Thus, when Heaven is about to confer a great office on any man, it first exercises his mind with suffering, and his sinews and bones with toil."

- Mencius

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Nilanjan Ray for his guidance. I very much appreciate the opportunity he gave me to explore the model compression field. It would never be possible for me to take this work to complete without his support and encouragement during my hardest time.

I'm also grateful to Dr. Mostafa Elhoushi from Huawei and Sara Elkerdawy from our vision and robotics lab. Their vision and innovative thoughts gave me directions and help me explore the quantization field for model compression. It would take me much more time to find a clear direction without their help.

Finally, I would also like to thank all individuals who work in the Vision and Robotics Research Lab for helping me finish my thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this section, we introduce the background of the model compression field and the need for quantization of neural networks as well as the problems existing in the quantization field. Then, we outline the structure of this thesis.

1.1 Motivation and Background

Nowadays, with the fast developments of deep learning, artificial neural networks are widely used to solve many complex problems and tasks, such as image classification, image segmentation, speech recognition, machine translation, and so on. However, these developments came at a cost. In order to boost the performance of neural networks, researchers seek to develop deeper and more complex models. The training and deploying of such deeper models require significant amount of storage and computation resources. On the other hand, as resource-constrained edge devices such as mobile phones and tablets are widely used worldwide, the need to deploy highly efficient artificial neural networks with a low resource consumption is increasing.

To reduce the storage and computation cost of neural networks, many works have been proposed in the model compression and acceleration field. Some of them try to design lightweight footprint architectures. Some find ways to modify the existing successful architectures. In this thesis, we summarize most of these methods into four categories: (i)Compact Architecture Design, (ii) Knowledge Distillation, (iii)

Pruning, and (iv) Quantization.

1.1.1 Compact Architecture Design

Compact Architecture Design refers to architectures that are designed specifically for edge devices and the methods that create compact network architectures based on certain constraints.

SqueezeNet [2] introduced by Iandola, MobileNetV1-V3 [3–5] introduced by Google and ShuffleNetV1-V2 [6, 7] are some of the state of the art compact neural network designs. The most common idea of these networks is to use a smaller convolution filters (1x1, 3x3) instead of larger ones (5x5).

SqueezeNet consists of multiple Fire Modules, convectional layers, pooling layers, and fully connected layers. The squeeze layers of the Fire Module use a 1x1 filter instead of 3x3 and the expanded layers of the Fire Module also reduce the usage of 3x3 convolution filters. This model achieves a similar accuracy level as AlexNet [8], while the model size is compressed to 50 times as small as the original one.

MobileNetV1 uses depthwise separable convolutions to replace the original convolution. The computation of such convolution is a combination of depth-wise convolution and point-wise convolution. For depth-wise convolutions, the number of filters is the same as that of the input channels and each filter only needs to take care of one channel. These filters are usually 3 x 3. Then, the resulting feature maps are combined and passed into the point-wise convolution which is similar to the regular convolutions but only uses 1x1 filters. The whole process of the depthwise separable convolution is shown in Figure 1.1. MobileNetV2 introduced Inverted Residuals to extract more features to reduce the inference time. MobileNetV3 makes use of hardware-aware network architecture search to obtain smaller networks. SENet [9] was also added to the blocks of MobileNetV3 in order to improve the accuracy.

ShuffleNetV1 makes use of group convolution to reduce model sizes and it uses channel shuffle to increase connections among feature maps. ShuffleNetV2 mentioned

four practical guidelines for efficient network architecture design and they are: 1) Equal channel width minimizes memory access cost. 2) Excessive group convolution increases memory access cost. 3) Network fragmentation reduces the degree of parallelism, and 4) Element-wise operations are non-negligible. ShuffleNetV2 improved ShffleNetV1 based on these 4 guidelines and achieved higher accuracy and shorter run time.

Compact architecture design makes use of smaller filters and specialized convolutions to reduce the model parameter and size. However, it is challenging to be use these designs together with other model compression methods.

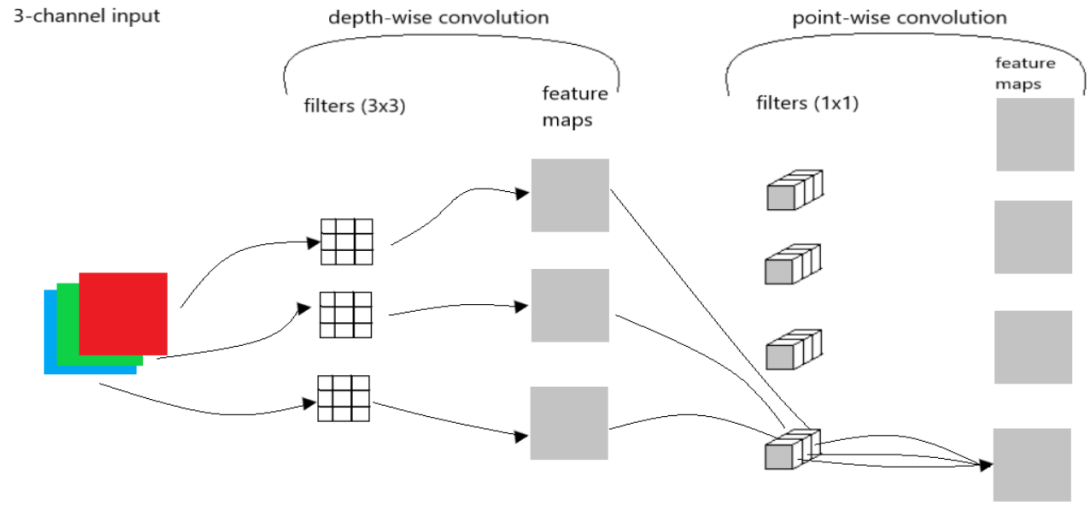


Figure 1.1: An example of depthwise separable convolution

1.1.2 Knowledge Distillation

Knowledge distillation is the process of transferring the knowledge learned by a large model to a smaller compact model. In knowledge distillation, the model obtained from the training of a large deep neural network is usually called the Teacher Model. The model obtained from the training of a light compact neural network is called Student model, MobileNet for example. In 2015, Hinton [10] introduced the concept of a knowledge distillation framework, which uses the output of a Teacher model as

the soft target to guide the training of a student network. The reason is that the soft targets that have high entropy contain much more information than the hard targets while it introduces less variance in the gradient. The process of knowledge distillation is as shown in Figure 1.2. The soft target is calculated as in eq. (1.1):

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)} \tag{1.1}$$

where i is the i-th target class, $j \in \{1, 2, ..., k\}$, k is the number of classes and T is a temperature. When T is set to 1, q_i is a regular softmax function. A higher temperature T will produce a softer probability distribution over classes, which is called "soft target". For the same input x, both the teacher network and the student network will produce a soft target. The student will then use the two soft targets together with the hard target ($T = 1$) in the cross-entropy loss to learn the weights.

The goal of knowledge distillation is to transfer the representative knowledge that's contained in the original network into a smaller neural network. The teacher network and the student network usually are created to minimize the KL divergence. Tian [11] pointed out that KL divergence ignored some important structural knowledge in the teacher network and introduced the concept of contrastive learning. The key idea of contrastive learning is to learn a representation that is close to "positive pairs" and push away the representation that is close to "negative" pairs. Experiments show that this method outperforms traditional knowledge distillation on many transfer tasks.

Nowadays, researchers try to use knowledge distillation to design a better student model. The student model can not only learn the behavior of the teacher model, but it can also surpass the teacher model by learning knowledge from other sources. Furlanello [12] introduced Born Again Neural Networks (BAN) that outperforms the teacher network significantly. Differing from the original knowledge distillation, the goal of BAN is no longer model compression. It tries to train the student model parameterized identically to the teacher network. The main idea is to train the

student model with the goal of predicting the correct labels and matching the output distribution of the teacher model after the teacher model converges. One example is to have DenseNet as the teacher and ResNet as the student. The accuracy of ResNet after being distilled by BAN is higher than DenseNet. Gao [13] brought out the method called Residual Knowledge Distillation that incorporates the concept of an Assistant. The assistant guides the student network by learning the residual error between teacher and student.

Methods based on knowledge distillation can greatly compress the model and help reduce the computation cost. Nevertheless, most of the knowledge distillation methods are used for the prediction tasks that have softmax as the loss function. Recently, many researchers are exploring ways to use knowledge distillation on compressing models of tasks like object detection and semantic segmentation. He [14] introduced an efficient knowledge distillation method for semantic segmentation and immensely improved and compressed the student model without introducing extra parameters and computations. This method consists of two parts: one is to compress the knowledge in the teacher model by using an autoencoder, and the other is to use an affinity distillation module which handles the inconsistency between the feature maps of the teacher and student model, to capture the long-dependent relationship from the teacher network.

Knowledge distillation can be used in many fields including natural language processing and semi-supervised learning. The soft output of the teacher model can guide the student model to achieve better performance.

1.1.3 Pruning

Pruning usually tries to determine the importance of parameters by designing some estimation metrics and standards then prune the redundant parameters based on these metrics. Pruning can also reduce the complexity of neural networks and therefore, mitigate the over-fitting problem.

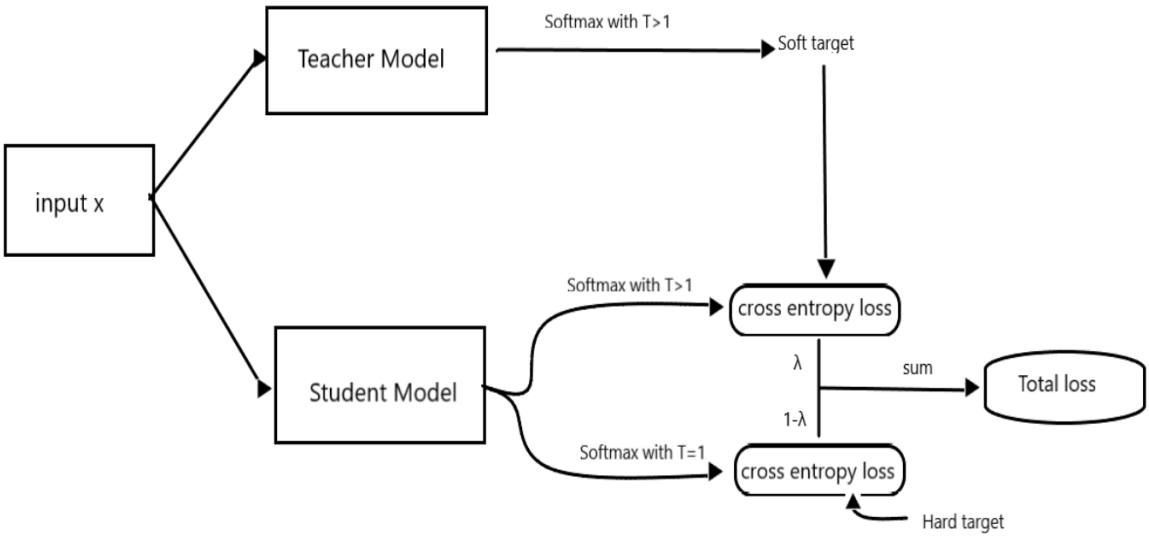


Figure 1.2: An example process of a typical knowledge distillation mentioned by Hinton [10]

Based on different granularities, pruning is usually categorized into four different categories [15]. The first and coarsest granularity is layer-wise pruning. A whole layer can be pruned if necessary at this level. The second granularity is removing the feature maps/filters. Feature map is the output of the network and filters are the parameters. Pruning a feature map is the same as pruning the filters of the previous layer to obtain a thinner network. The next granularity is kernel-wise pruning, which is removing channels of a filter. The finest granularity is to prune the weights of a kernel (intra-kernel pruning), which results in a more sparse matrix. These four granularities can be further categorized as structured pruning and unstructured pruning. Structured pruning means pruning a larger part of the network [16]. Layer-wise pruning, feature map pruning, and kernel pruning are structured pruning, whereas intra-kernel pruning is unstructured.

Many pruning methods in the early ages are unstructured. LeCun [17] and Hassibi [18] came up with Optimal Brain Damage (OBD) and Optimal Brain Surgeon (OBS) methods in 1990 and 1993. The latter is an improvement based on the former one. The basic idea is to use the loss function with respect to the Hessian matrix to measure the importance of the weights in order to remove the redundant weights. Both methods

can improve the accuracy but it takes a long time to train. Therefore, they can not be used on a larger network. Zhang [19] brought out a weight pruning system called the alternating direction method of multipliers (ADMM). With the constraints specifying the sparsity requirements, the weight pruning problem is converted as a nonconvex optimization problem. ADMM, then, decomposes the nonconvex optimization into two sub-problem that are solved iteratively. One of the problems is solved by using stochastic gradient descent and the other is solved analytically. Such a system helps compress the weight parameters of AlexNet and LeNet [20] on a large scale on ImageNet and MNIST dataset. The above unstructured pruning methods only make the weight matrix sparse. However, the computation needed is still the same. In addition, to speed up the computation with sparse matrices, special software and hardware may be needed. To tackle such a problem, Ma [21] brought out the PCONV method which introduced fine-grained pruning patterns inside the coarse-grained structures. PCONV comprises two kinds of prunings: intra-convolution kernel pruning that generates filters with different sparsity and inter-convolution kernel pruning that further prunes the filters to obtain connectivity sparsity. Ma also designed a special compiler to achieve a high compression rate and inference speed on large-scale DNN.

Besides specialized pruning structure and compiler, structured pruning can also avoid the disadvantages of unstructured pruning. The main idea of structured pruning is to remove the unimportant channel or filters without hurting the accuracy. He [22] firstly uses LASSO regression to choose channels for pruning, then reconstruct the output using the least square operation. Chin [23] mentioned a way to learn a global ranking of filters of different convolutional layers. Such ranking is used to create a set of architectures that are ranked by their accuracy/latency trade-offs. The architecture with the lowest rank can, therefore, be pruned. Molchanov [24] uses the first and second order of Taylor expansion to estimate the contribution of filters and pruning the low contribution filters layer by layer.

Structured pruning usually uses the norm as the measurement to estimate the

importance of a filter. There are two conditions when using norm as the estimation: one is that the variance of the norm should be large and the other is that the minimum norm should be close to 0. When using the norm that satisfies those two conditions as the pruning standard, the reconstructed network will usually encounter less accuracy drop. However, it is hard for the norms to satisfy both conditions in reality. One way to overcome this problem is to use the similarity between the filters to determine whether a filter is redundant. He [25] uses the Geometric Median to find the similarity between filters and uses the filter that is further away from the median to replace the ones that are close to the median. Lin [26] finds another workaround for the norm problem. Lin mentioned that no matter the batch size of the input, the average rank of the feature maps that are created by the same filter is always the same. Feature maps with high rank usually contain more information, so filters that produce low-rank feature maps can be pruned.

Pruning is basically removing unimportant weights from the network. The pruned network can also be seen as a sub-network of the original network, from another point of view. Pruning can shrink the search space of neural networks can it can usually be combined with quantization for model compression purposes.

1.1.4 Quantization

Quantization is to use a fewer bits to represent real numbers used in parameters, such as 8-bit, 4-bit, and 1-bit (binary). It usually requires the help of specialized hardware chips to deploy in practice. After the weights and activations in the neural network are quantized, the multiply-accumulate operations can be replaced by low-bit operations which are cheaper and faster. Multiplication can even be avoided under some extreme low bit situations such as binary networks [27–29] and ternary networks [30, 31]. Therefore, the use of low-bit quantization can greatly reduce the storage and computation cost. Meanwhile, low-bit quantization can also help advance the development of hardware chips that are targeting neural networks.

Quantization usually means to find the best optimization that gives the smallest error between the quantized values and full precision values, as shown in the eq. (1.2).

$$min\ J(q_x(x)) \equiv ||x - q_x(x)||_2^2 \tag{1.2}$$

where x is the full precision value, $q_x(x)$ is the quantized low-bit value and $J(q_x(x))$ is the error between the full precision value and quantized value.

Methods in quantization typically can be summarized as weight sharing and low-bit representations. Weight sharing is usually realized by clustering. Han [32] uses K-Means clustering for the weight matrix of each layer, and uses the cluster centroid as a value for the weights that are in the cluster. Since weights in the same cluster share the same centroid, only the centroid needs to be stored as an index. Then, they use a lookup table to find the corresponding value. Another way for weight sharing is to use hash tables for quantization. HashedNets introduced by Chen [33] uses a low-cost hash function to randomly group connection weights into hash buckets and the connection weights within the same buckets share a single parameter value.

The above methods are traditional quantization methods. Stock [34] introduced a vector quantization method that reduces the bit length based on Product Quantization. Unlike the traditional vector quantization method [35], this method focus on the importance of the activation rather than the weights. The key idea is to minimize the reconstruction error for the in-domain inputs and use the full-precision network as the teacher to guide the training of the quantized network. This method can efficiently run the inference on the CPU.

Many low-bit quantizations are networks improved based on binary neural networks [27]. Binarization uses 1-bit values to greatly quantize the parameters into two possible values $0(-1)$ and $+1$. After the BinaryConnect was brought out by Courbariaux [36] in 2016, quantizing weights and activation into binary numbers has been an efficient way to compress neural networks. With binarization, complex matrix multiplication can be simplified as XOR and bit-shifting operations. However, due

to the discreteness and limited expressiveness of binary numbers, lots of information is lost during the forward and backward passes. In the forward pass, the diversity of the model will be greatly reduced when the weights and activation are limited to 2 numbers, which will result in the decrease of the model accuracy. To solve this, Qin [37] introduced Information Retention Network (IR-Net) that retains the information in the forward activations and backward gradients. IR-Nets consists of two processes: first, balancing and standardizing weights in the forward propagation to minimize the loss; second, minimizing the quantization error while maximizing the parameter information entropy, which reduces the information loss of weights and activation without adding additional operations. The most frequently used function for binarization is the non-differentiable sign function. Therefore, gradient estimation is needed during the backward propagation. IR-Net uses an error decay estimator to gradually approximate the sign function in the backward propagation to minimize the information loss of the gradients. Such a method is experimented with various neural networks on CIFAR-10 and the ImageNet dataset. The results show that it achieves better accuracy than vanilla binarization methods.

Except for binary network, ternary and int8 quantization is also frequently seen in the low-bit quantization methods. Wang [38] introduced the Two-Step Quantization framework that converts the neural network quantization into two steps. The first step is to use the sparse quantization method to quantize the activations. Only important positive values are quantized. The rest are set to zeros. The second step is a non-linear least square regression problem with the constraint on the bit-length, which can be iteratively solved. Mellempudi [39] mentioned the ternarization method that minimizes the quantization error by exploiting the local correlations of the parameters in dynamic range. Zhu [40] shows us an INT8 quantization method that uses Deviation Counteractive Learning Rate Scaling and Direction Sensitive Gradient Clipping, which addressed the accuracy loss problem introduced by quantization.

When using a low-bit quantization, the accuracy of the quantized neural network

is usually significantly lower than the full-precision network, because the noise introduced during the quantization process is unavoidable. When using extreme low-bit values to quantize the weight and activation, such a problem is even more severe. On the other hand, the limitation of the structured matrix may also lead to accuracy loss. Therefore, quantization is mostly used together with other compression methods.

1.2 Thesis Scope

This thesis will focus on the quantization of neural networks. However, instead of exploring and experimenting with extreme low-bit quantization like binary and ternary quantization, we focus on mixed low-bit quantization in particular, since different layers in a neural network contain different levels of information and should not be treated the same. Certain quantization methods are selected, analyzed, experimented and compared in this thesis. We will talk more about why we chose mixed quantization and the results we observed.

This thesis will also have some analysis of the gradient estimation methods that are used for quantization. Since the process of quantization contains non-differentiable operations, some estimation method has to be used to let the gradients successfully back-propagate to update the parameters.

1.3 Thesis Contributions

In this thesis, we include the following contributions:

- We introduce the idea of accuracy estimation to bring any fixed-precision quantization to a compressed mixed-precision quantization so that further compression can be reached with higher flexibility.
- We propose an accuracy-aware criterion to weigh the importance of each layer. This will allow us to have a better interpretation of the final bit-length configuration compared to other search methods.

- We analyzed the reason to that quantized weights is better than full-precision weights for estimating the gradients of quantization.

1.4 Thesis Outline

In the next chapter, we will see a detailed literature review on some related quantization papers and their limitations. We will also explain our algorithm and our layer importance estimation methods for the mixed-precision quantization. In chapter 3, we will present our experiments with our method and the results that we obtained and compare our method with the existing method to discuss why and when to choose our method. In chapter 4, we will review some papers that explored the options of gradient estimation for the quantization process and our experiments and results will be discussed in chapter 5. The last chapter will conclude this thesis and point out some future work.

Chapter 2

Related Works and Mixed Precision Quantization

In this chapter, we will first study some state-of-the-art quantization methods. Then we will introduce our method [1] for mixed-precision quantization that can be applied to most of the existing quantization methods.

2.1 Preliminary

The quantization process can be summarized as projecting real-valued numbers into discrete quantum numbers. The set of these discrete quantum numbers are called quantization levels. When applied to neural networks, those real-valued numbers to quantize are weights and activations. Since the weights and activations are not infinite, (i.e. they have upper and lower bonds), we need to project (i.e., round) those real-valued numbers onto a discrete set of values. The activations are similarly treated except that the minimum value for the activations is set as zero. We will discuss mostly about weights in the following chapters but the activations can just be treated in the same manner. Since the distribution of the weights is bell-shaped [41], certain clipping methods can be applied before projection to increase the precision for the majority values around the center of the distribution. Suppose we define the weight of a convolutional layer as W, then the quantized weight will be defined as:

$$\tilde{W} = \prod_{Q(\alpha,b)} \lfloor W, \alpha \rceil \tag{2.1}$$

where $\lfloor W, \alpha \rceil$ is the clipping function that clips W to the range $[-\alpha, \alpha]$. $Q(\alpha, b)$ is a set of quantization levels whose maximum is α, and b is the bit-width. $\prod$ denotes the projection function which will project the clipped weight onto the quantization levels.

2.2 Different Quantization Levels

The quantization levels are the set of discrete numbers that the real-number values need to be projected to. Usually, once the quantization level is set, the real numbers are projected using a round function. We will discuss papers based on the quantization level schema they used.

2.2.1 Uniform Quantization

Many existing quantization methods use uniform quantization [42–44], whose projection level can be defined as:

$$Q_{uni}(\alpha, b) = \alpha \times \{0, \frac{\pm 1}{2^{b-1}-1}, \frac{\pm 2}{2^{b-1}-1}, \frac{\pm 3}{2^{b-1}-1}, ..., \pm 1\} \tag{2.2}$$

where α is the maximum quantization level and b is the bit-width.

Parameterized Clipping Activation (PACT) [43] introduced by Choi et al. uses uniform quantization together with a new way to quantize the activations. PACT uses a new trainable parameter α to represent the clipping range for the activations, to make the quantized activations small enough to reduce the quantization error but not too small to allow the gradients to flow effectively. ReLU function has been commonly used as the activation function for most of the CNNs since it allows gradient of activations to propagate through deep layers [45]. However, applying quantization

without clipping directly can hurt the model accuracy significantly as seen in Figure 2.1. It has been studied that putting an upper bound to the output of the activation function during quantization can greatly reduce such quantization error [46, 47]. Since it's hard to determine a globally optimal activation clipping value, PACT is introduced to dynamically determine the clipping value layer by layer during the training process. The original ReLu function, which is shown in eq. (2.3), is converted as eq. (2.4)

$$y = Relu(x) = 0.5(x + |x|) = \begin{cases} 0, & x \in (-\infty, 0) \\ x, & x \in (0, +\infty) \end{cases} \tag{2.3}$$

$$y = PACT(x) = 0.5(|x| - |x - \alpha| + \alpha) = \begin{cases} 0, & x \in (-\infty, 0) \\ x, & x \in [0, \alpha) \\ \alpha, & x \in [\alpha, +\infty) \end{cases} \tag{2.4}$$

where x is the input to the activation function, y is the output of the activation function and α is the upper bound of the clipped activation function. After applying the uniform quantization, the quantized activation can be then calculated as:

$$y_q = round(y \cdot \frac{2^b - 1}{\alpha}) \cdot \frac{\alpha}{2^b - 1} \tag{2.5}$$

Since the $round(\cdot)$ function in the quantization process is non-differentiable, to update the α in the back-propagation, the gradient of $\frac{\partial y_q}{\partial y}$ needs to be estimated. Bengio et al. [48] introduced the Straight-through estimator (STE) that calculates the gradient of $\frac{\partial y_q}{\partial y}$ as 1. The gradient of the quantized activation after PACT is, therefore:

$$\frac{\partial y_q}{\partial \alpha} = \frac{\partial y_q}{\partial y} \frac{\partial y}{\partial \alpha} = \begin{cases} 0, & (-\infty, \alpha) \\ 1, & [\alpha, +\infty) \end{cases} \tag{2.6}$$

The results of the PACT are shown in Table 2.1. It is shown that PACT can outperform other uniform quantization methods that do not handle the activation clipping

ranges. More results can be found in the original paper. Choi et al. in their paper showed that this PACT method quantizes activations very effectively while simultaneously allowing weights to be heavily quantized.

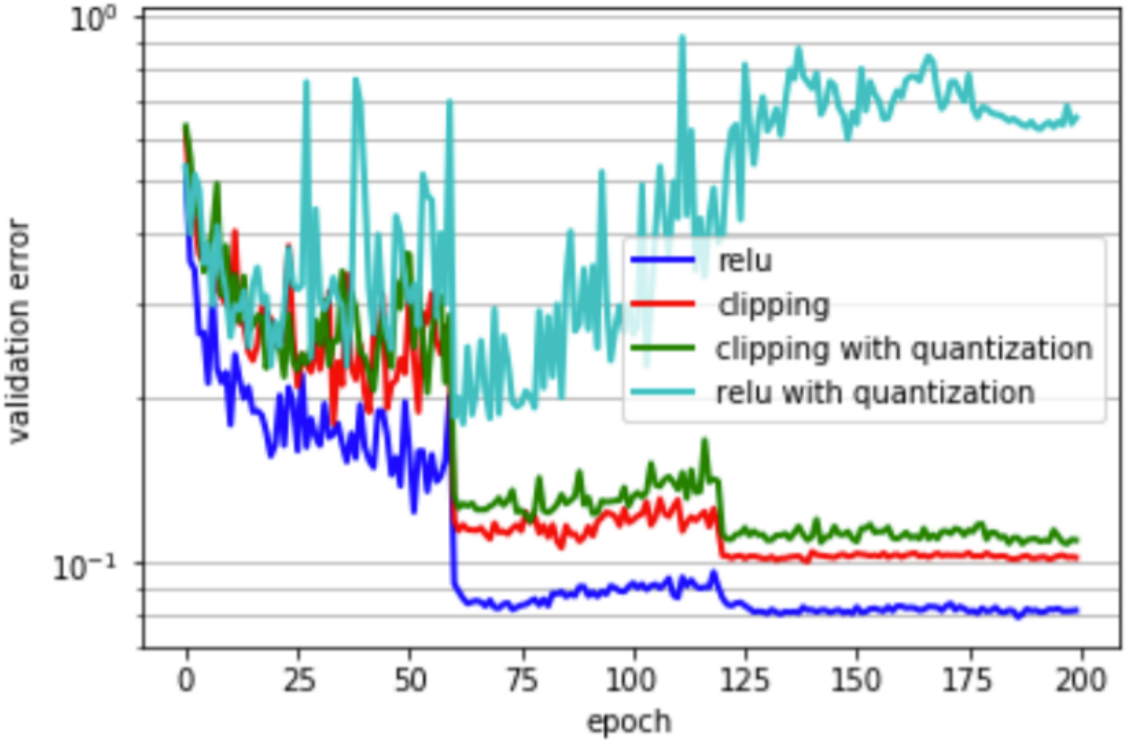


Figure 2.1: Validation Error over Number of Epochs of Relu Activation Function with and without clipping and quantization for ResNet20 on CIFAR10.

Network	FullPrec	DoReFa				PACT			
		2b	3b	4b	5b	2b	3b	4b	5b
CIFAR10	0.916	0.882	0.899	0.905	0.904	0.897	0.911	0.913	0.917
SVHN	0.978	0.976	0.976	0.975	0.975	0.977	0.978	0.978	0.979
AlexNet	0.551	0.536	0.550	0.549	0.549	0.550	0.556	0.557	0.557
ResNet18	0.702	0.626	0.675	0.681	0.684	0.644	0.681	0.692	0.698
ResNet50	0.769	0.671	0.699	0.714	0.714	0.722	0.753	0.765	0.767

Table 2.1: Comparison of top-1 accuracy between DoReFa and PACT. Weights are quantized with DoReFa scheme, whereas activations are quantized with PACT scheme

However, quantizing all the layers in the same network to the same bit-length is

not necessarily the best option. Bitpruning [42] in 2020 offers a solution to squeeze out every possible benefit from quantizing the neural network. It introduces a way to learn the bit precision by using gradient descent. The first step of Bitpruning is to convert the discrete quantization levels back into real numbered values so that they can be learned for different layers. Bitpruning also uses uniform quantization but instead of setting a clipping range, it makes use of the minimum and maximum values of the weights. Unlike eq. (2.5), the converted discrete quantization is calculated as:

$$Q_d(V, b) = L_{min} + round(V \cdot \frac{2^b - 1}{L_{max} - L_{min}}) \cdot \frac{L_{max} - L_{min}}{2^b - 1} \tag{2.7}$$

where $V \in [L_{min}, L_{max}]$ is the values by rounding it to the nearest greater integer. Some of the results of Bitpruning is shown in table 2.2. The bitlength of the weights and activations are average values of all the layers. The results in the paper show that bitprunning can bring the average bit length to relative low settings without hurting the accuracy too much.

$$L = L_l + \gamma \sum (\lambda_i \times b_i) \tag{2.8}$$

where L_l is the original loss function, γ is the regularization coefficient used for selecting how aggressive the quantization should be, λ_i is the weight corresponding to the importance of the i-th group of values, and b_i is the bitlength of the activations or weights in that group.

Another method that uses mixed-precision uniform quantization is ZeroQ [44] by Cai et al. ZeroQ addresses the problem that the original training data set is not always available for quantization of an existing model. It proposes a zero-shot quantization framework for quantizing a pre-trained neural network model without accessing the original data. There are 3 steps in total. The first step is to use the statistics obtained from the batch normalization layers to generate distilled data. In order to successfully generate the data without accessing the original data, it is required for all the layers in the neural network to be followed by a batch normalization layer. Initially, the

		Non-integer Bitlengths			Rounded Integer Bitlengths		
Network	Regularizer	Accuracy	Weights # of bits	Activations # of bits	Final Accuracy	Weights # of bits	Activations # of bits
ResNet18	Baseline	94.9	32float	32float	94.9	32float	32float
	γ=0.5	94.2	1.67	2.73	94.4	1.9	3.38
	1	93.5	1.3	2.26	94.1	1.43	2.9
	2.5	93.1	1.15	2.01	93.4	1.24	2.43
	5	92.8	1.14	1.99	93.3	1.24	2.48
	10	94.1	1.61	2.35	94.2	1.9	2.9

Table 2.2: Activation/weight bitlengths and achieved accuracy of aggressive quantization and different strength reguralizers on CIFAR10 for non-integer and integer bitlengths.

data is randomly sampled from a Gaussian distribution. It is then updated using backpropagation by minimizing the following equation:

$$\min_{x^r} \sum_{i=0}^{L} ||\tilde{\mu}_i^r - \mu_i||_2^2 + ||\tilde{\sigma}_i^r - \sigma||_2^2 \tag{2.9}$$

where x^r is the generated data, μ_i and σ_i are the mean and standard deviation of the i-th layer, obtained from the batch normalization layers, $\tilde{\mu}_i^r$ and $\tilde{\sigma}_i^r$ are the mean and standard deviation computed using x^r. Once we have the distilled data, the next step is to determine the sensitivity of each layer with respect to quantization. The sensitivity is measured using KL divergence of the output between the original model and the quantized model. One example process is shown in fig. 2.2. The last step is to find the final bit configuration of the whole network given a model size constraint. ZeroQ uses Pareto frontier optimization to solve this constraint satisfaction problem. The paper presented extensive experiments and results and it is shown that ZeroQ could exceed the previous zero-shot quantization method.

2.2.2 Non-uniform Quantization

Uniformly quantize the weights and activations is convenient but not ideal. Weights are more gathered around the center as shown in Figure 2.3. Non-uniform quantiza-

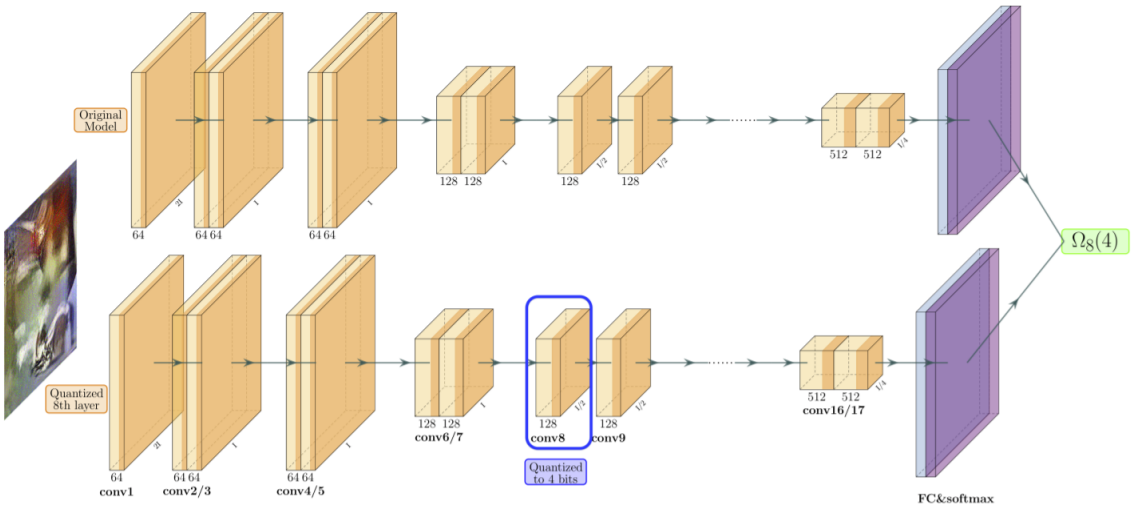


Figure 2.2: An illustration of sensitivity computation for ResNet18 on ImageNet. $\Omega_8(4)$ means the sensitivity (i.e. the KL divergence) of the 8-th layer when quantized to 4-bit.

tion levels may be more suited.

One of the popular non-uniform quantization levels is to use power-of-two (PoT) values [49, 50]. That is the quantization level is defined as:

$$Q_{pot} = \alpha \times \{0, \pm 2^{-2^{b-1}+1}, \pm 2^{-2^{b-1}+2}, ..., \pm 2^{-2^{-1}}, \pm 1\} \tag{2.10}$$

Apparently, as a non-uniform quantization schema, PoT fits better for the weights than the uniform quantization because it has a higher resolution around the distribution center. Another huge advantage is that the multiplication of a power-of-two value and any other number can be implemented using bit-shifting operations instead of the digital multipliers. The Incremental Network Quantization (INQ) presented in [50] shows that using power-of-two values can not only faster the training and inference time, it can also reach similar accuracy of full precision networks, as shown in table 2.3.

However, Li et al [51] pointed out that one drawback of the PoT method is its rigid resolution around the tail of the distribution. It means that PoT quantization does not benefit from having more bits. When bit length is increased by 1, the quantization level only increases around the tail of the distribution, which is shown in fig 2.4 (b).

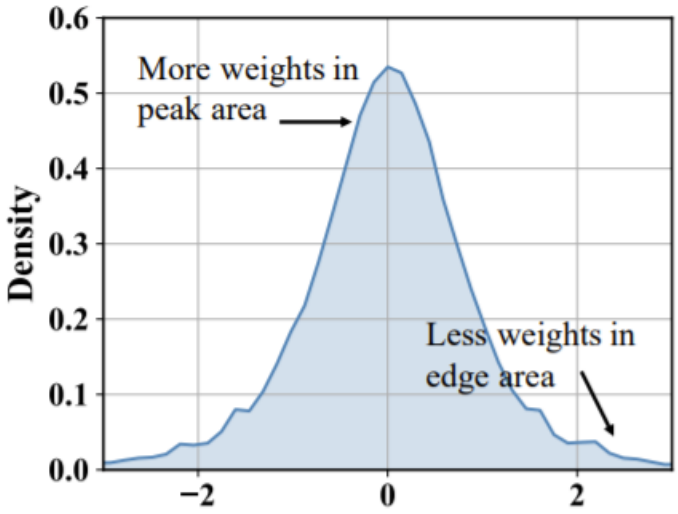


Figure 2.3: Density of weights in ResNet-18.

Model	Bit-width	Top-1 accuracy	Top-5 accuracy
ResNet-18	32	68.27%	88.69%
INQ	5	68.98%	89.10%
INQ	4	68.89%	89.01%
INQ	3	68.08%	88.36%
INQ	2	66.02%	87.13%

Table 2.3: INQ[50] generates extremely low-precision (4-bit and 3-bit) models with improved or very similar accuracy compared with the full-precision ResNet-18 model on ImageNet.

Additive Power-of-Two (APoT)[51] quantization is, therefore, introduced to solve the problem. The quantization levels for APoT is as shown below:

$$\begin{aligned} Q_{APoT}(\alpha, kn) &= \gamma \times \{\sum_{i=0}^{n-1} p_i\}, \\ \textit{where } p_i &\in \{0, \frac{1}{2^i}, \frac{1}{2^{i+n}}, ..., \frac{1}{2^{i+(2^k-2)n}}\} \end{aligned} \tag{2.11}$$

where k is a hyper parameter that defines base bit-width, and n is the number of additive terms. If the bit-width is b, n can be calculated as $n = \frac{b}{k}$. Finally, γ is a scaling coefficient to guarantee the maximum level equals α. These quantization levels will project the values non-uniformly and enable more levels near 0 which is the

center of the weights after clipping. This paper also introduced a way to dynamically clipping the weights just like how PACT clips the activations. With the help of STE and eq. (2.1), the gradients to update the α is:

$$\frac{\partial \tilde{W}}{\partial \alpha} = \begin{cases} \frac{\partial \lfloor W, \alpha \rceil}{\partial \alpha} = sign(W) & if |W| > \alpha \\ 0 & if |W| \leq \alpha \end{cases} \tag{2.12}$$

It is clear that the clipped weights are not contributing to the gradient. Thus, the estimation is not accurate here. Therefore, APoT proposed the following clipping function:

$$\tilde{W} = \alpha \prod_{Q(1,b)} \lfloor \frac{W}{\alpha}, 1 \rceil, \tag{2.13}$$

which scales the weights into range $[-1, 1]$ first, then scales them back after projection. The estimated gradient is, therefore, changed to

$$\frac{\partial \tilde{W}}{\partial \alpha} = \begin{cases} sign(W) & if |W| > \alpha \\ \prod_{Q(1,b)} \frac{W}{\alpha} - \frac{W}{\alpha} & if |W| \leq \alpha \end{cases} \tag{2.14}$$

which can be easily calculated. Some results of APoT are included in table 2.4. We can see that APoT can reach relatively high compression ratio while having the best accuracy.

2.3 Imprinting

Imprinting [52] is a method used in the few-shot learning to approximate a classifier's weights when only a few training samples are available. Qi et al. [52] are inspired by the effectiveness of embeddings in retrieving and recognizing objects from unseen classes in metric learning and they estimate the weights of the final layer by using embeddings. Then, Elkerdawy et al. [53] use imprinted weights to help the layer pruning. The imprinting process is as follows: Firstly, they use eq. (2.15) to calculate the weight from the embeddings obtained from feature maps.

$$W_i[:, c] = \frac{1}{N_c} \sum_{j=1}^{N} I_{[c_j == c]} E_j \tag{2.15}$$

Method	Precision (W/A)	Top-1 Accuracy	Model Size (MB)	Precision (W/A)	Top-1 Accuracy	Model Size (MB)
Baseline	32/32	70.2	46.8			
DoReFa-Net [47]	5/5	68.4	8.72	3/3	67.5	6.06
PACT	5/5	68.9	8.72	3/3	68.1	6.06
PoT	5/5	70.3	7.22			
APoT	5/5	70.9	7.22	3/3	69.9	4.56
DoReFa-Net [47]	4/4	68.1	7.39	2/2	62.6	4.73
PACT	4/4	69.2	7.39	2/2	64.4	4.73
APoT	4/4	70.7	5.89	2/2	67.3	3.23

Table 2.4: Results of APoT compared to other methods with ResNet18 on ImageNet.

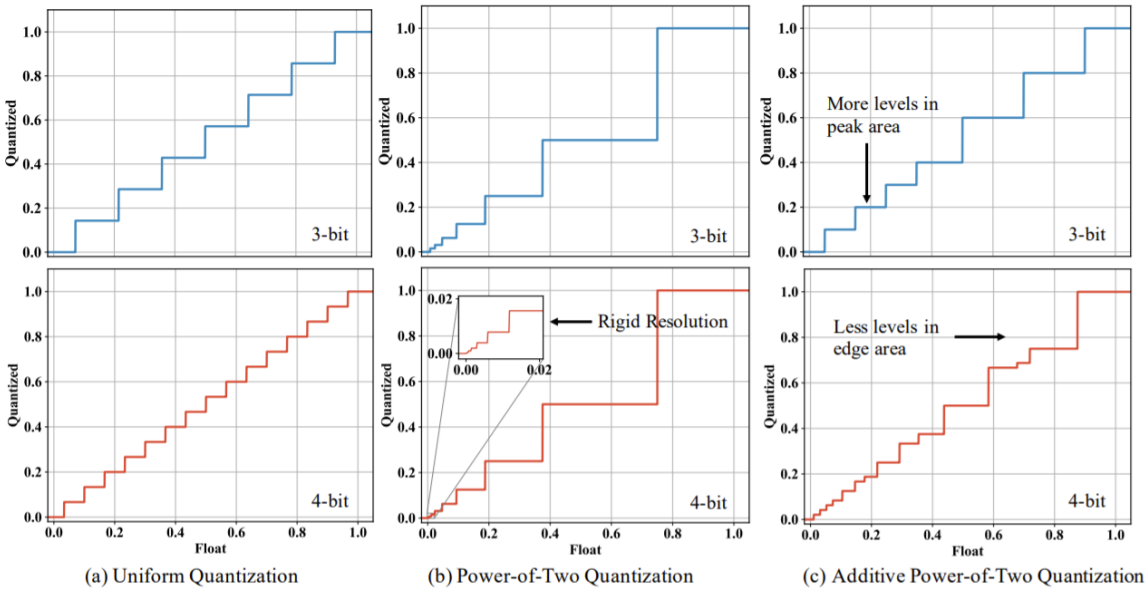


Figure 2.4: A example of quantization levels of unsigned data at 3 and 4 bit (α=1).

where W_i is the weight matrix for layer i, c is the class id, N_c is the number of samples in class c, N is number of samples, and E_j is the embedding for layer i. Then, the prediction of each class is obtained by finding the nearest class from the imprinted weights (see eq. (2.16)). The accuracy can be, therefore, calculated for each layer. One example result for using this imprinting for accuracy estimation is

shown in Figure 2.5. This method can obtain the accuracy for each layer with only one epoch. It is adopted in our bit selection algorithm. We'll see how to use it in the next chapter.

$$\hat{y}_j == \underset{c \in \{1,\ldots,C\}}{argmax} W_i[:, c]^T E_j \tag{2.16}$$

where i is the layer index, j is the sample index and E_j is the embedding.

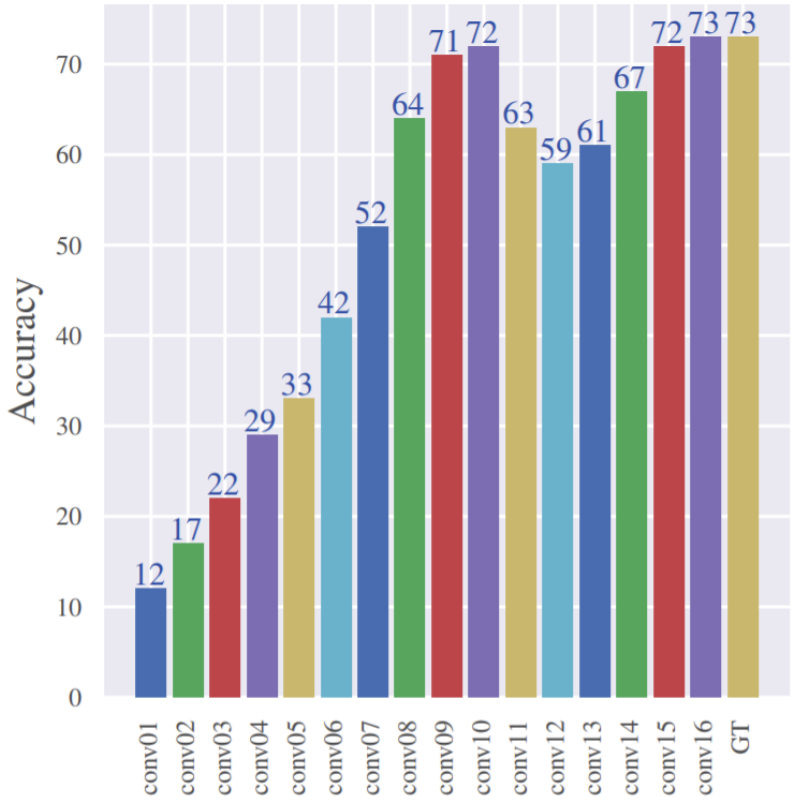


Figure 2.5: Layer-wise accuracy, rounded for better visualization, using imprinting for VGG19 on CIFAR100. GT shows the actual accuracy of the full model.

2.4 Layer Importance Estimation For Mixed Precision Quantization

In this section, we will present our layer importance estimation method for mixed-precision quantization.

2.4.1 Mixed precision with APoT

The additive power of two quantization (APoT) achieves high accuracy while decreasing the number of bits used by the convolutional layers in neural network architecture, as we discussed in section 2.2.2. The method sets the bit precision to be a fixed number for all layers, and the quantization range is set to be power-of-two values. However, convolutional layers in a neural network do not contain the same level of information. Layers with less information may, therefore, be further quantized to a lower precision. Hence, we will base on the APoT quantization schema to start our mixed-precision quantization. We will also continue to use the quantization formulas to update the clipping range α. However, our method can be applied to any fixed bit length quantization schema. We will show that in the next chapter.

2.4.2 The Bit Selection Algorithm

In order to better quantize the model, we can first measure the importance of each layer and then quantize the layers based on the importance. The general process is shown in Algorithm 1.

There are many choices for the importance measurement. We've tested different choices in the next chapter. The one that we are most satisfied with is to make use of the imprinting method, which can quickly give us the estimated accuracy at each layer. The detailed pipeline for using imprinting with the bit selection method is shown in 2. The process for using imprinting to estimate the layer accuracy in one epoch is as shown in Figure 2.6. The method is adopted from the method mentioned in Section 3.4.2. For each convolutional layer, we sample an embedding from the input feature maps using eq. (2.17). Then the imprinted weights are obtained by using eq. (2.15). The accuracy is finally calculated by using a dot product between the embedding and imprinted weights. With this bit selection algorithm, we are able to bring any fixed-precision quantization to a compressed mixed-precision quantization so that further compression can be reached with higher flexibility. The accuracy-aware criterion to

Algorithm 1 Bit Selection Algorithm: use importance measure to achieve mixed-precision quantization

Input: a pre-trained fixed-precision Neural Network with precision B_i, a target average precision interval (B_l, B_h)

Hyper-parameter: a small fine-tune number N

while current average precision $B > B_l$ **do**
    small fine-tune the network for N epochs
    choose the model with the best performance
    find a layer in the model that is the least important
    if the layer's precision $b > B_l$ **then**
        reduce the layer's precision by 1 and save the model
    else
        ignore this layer and find another least important layer
        repeat step 5
    end if
    fine-tune the model to recover the accuracy
    save the accuracy and bit-precision configuration
end while
Find the configuration that has the highest accuracy within the average precision range (B_l, B_h).

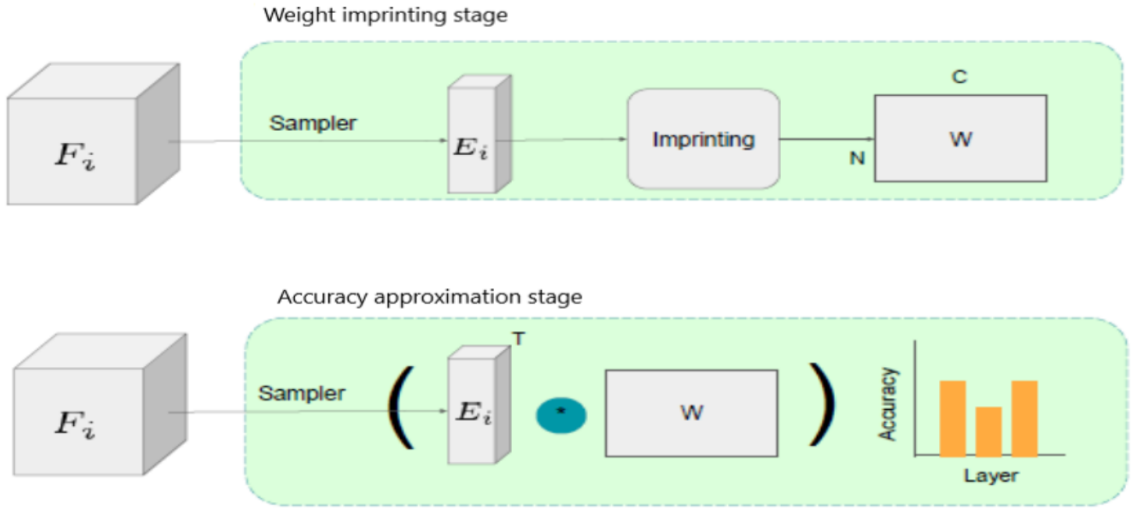


Figure 2.6: Pipeline illustrating accuracy approximation by imprinting. An embedding E_i is sampled from input feature maps F_i. Imprinting is then applied on the embedding to estimate the weight matrix W. Accuracy per layer is then calculated by a simple dot product between embedding and imprinted weights.

weigh the importance of each layer allows us to have a better interpretation of the final bit-length configuration compared to other search methods. Meanwhile, the

imprinting method we use is a one-shot step, which enables us to quickly identify rank the layers resulting in a shorter training time.

$$d = round(\sqrt{\frac{N}{f_i}}) \\ E_i = AdaptiveAvgPool(F_i, d), \tag{2.17}$$

where N is the length of embedding and f_i is the number of filters in the i-th layer. F_i is the i-th layer's feature map and E_i is our re-shaped embedding which is also the input of imprinting.

Algorithm 2 Pipeline for bit-selection with imprinting

Input: initial bit length configuration of all convolutional layers B_{start}, minimum bit length configuration of all convolutional layers B_{end} , number of convolutional layers K **Output:** bit configuration for all convolutional layers B

1: Calculate maximum total bit-length $\hat{B}_{max} = \sum_{i=0}^{K} b_i$, $b_i \in B_{start}$
2: Calculate minimum total bit-length $\hat{B}_{min} = \sum_{i=0}^{K} b_i$, $b_i \in B_{end}$
3: Calculate maximum of iterations needed $N_{max} = \hat{B}_{max} - \hat{B}_{min} + 1$
4: Initialize the configuration to use $B_{current} = B_{start}$
5: **for** *iteration* $N = 1, 2, \ldots, N_{max}$ **do**
6: With $B_{current}$, use imprinting with quantization to get the estimated accuracy of each convolutional layer, $Acc = \{acc_1, acc_2, \ldots, acc_K\}$
7: Find the difference between each layer and its previous layer, $diff = \{\|acc_i - acc_{i-1}\| \,|\forall i \in [2, K-1], acc_i \in Acc\}$
8: Find the index of the minimum difference, $idx = argmin(diff)$)
9: Record the configuration as B_N
10: Record the accuracy of the last layer as Acc_N
11: Update the configuration $B_{current}$ by setting $b_{idx} = b_{idx} - 1$
12: **end for**
13: Choose the best configuration $B = B_N$ where $Acc_N = max(\{Acc_i|\forall i \in [1, N_{max}]\})$

Chapter 3

Experiments for Mixed Precision Quantization

In this chapter, we demonstrate our experiments around mixed-precision quantization. These experiments were run on P100 GPU with Pytorch. We will show why to choose mixed precision. We will also show the effect of different importance estimation metrics used in Algorithm 1.

3.1 Why Mixed Precision?

To study the impact of mixed quantization, we first start our experiment using APoT with all layers quantized to 4-bit. Then, in each step, we will randomly choose one layer to be further quantized to 3-bit and repeat until all layers are 3-bit. The same procedure is then repeated starting with all layers to be 3-bit. We run these experiments multiple times and for each run, we record the best accuracy reached as the best result for that run. The final results are shown in Table 3.1a and Table 3.1b. From the tables, we can see that sometimes mixed-precision configuration outperforms the original APoT settings with an even lower average bit length, so it is possible to further quantize the fixed bit-length model without hurting the accuracy. However, comparing within the same bit-length group (e.g. bit-length between 3 and 4), the random results vary a lot. It gives us the hint that different layers may have different importance levels during quantization and the order in which layers are chosen for

quantization may have a significant impact on the final accuracy of that model.

ResNet20	AVG bits	Top-1
APoT	4	92.45
Ours(random)	3.75	92.48
Ours(random)	3.75	92.34
Ours(random)	3.7	92
Ours(random)	3.25	91.61
APoT	3	92.49
Ours(random)	2.75	91.83
Ours(random)	2.8	91.75
APoT	2	90.96

(a) Results for randomly quantizing layers with ResNet20 on CIFAR-10.

ResNet56	AVG Bits	Top-1
APoT	4	93.93
Ours(random)	3.5	93.76
Ours(random)	3.5	94.1
APoT	3	93.77
Ours(random)	2.78	93.54
Ours(random)	2.73	93.61
APoT	2	93.05

(b) Results for randomly quantizing layers with ResNet56 on CIFAR-10.

Table 3.1: Results for quantizing layers randomly. (a) was run with ResNet20 on CIFAR-10 and (b) was run with ResNet56 on CIFAR-10. "AVG Bits" is the average bit-length of convlutional layers and "Top-1" is the top-1 accuracy reached with that specific bit-length configuration.

3.2 Hyper-parameters in The Bit Selection Algorithm

There's a "small fine-tune number" hyper-parameter in Algorithm 1. It represents how many times we want to fine-tune the model to recover the accuracy before finding the actual accuracy level that can accurately represent the relative performance of the current bit-precision configuration. This number can be as small as 1 and it can go up to as many as we like. However, the higher the number is, the longer would be the training. Therefore, we conduct some experiments with different N. The results are shown in Figure 3.1. We notice that for N=1, the accuracy fluctuates a lot. However, with N=2 and N=4, the relative highest accuracies for each precision ranges are similar. To be safe, we use N=4 for our experiments unless specified otherwise.

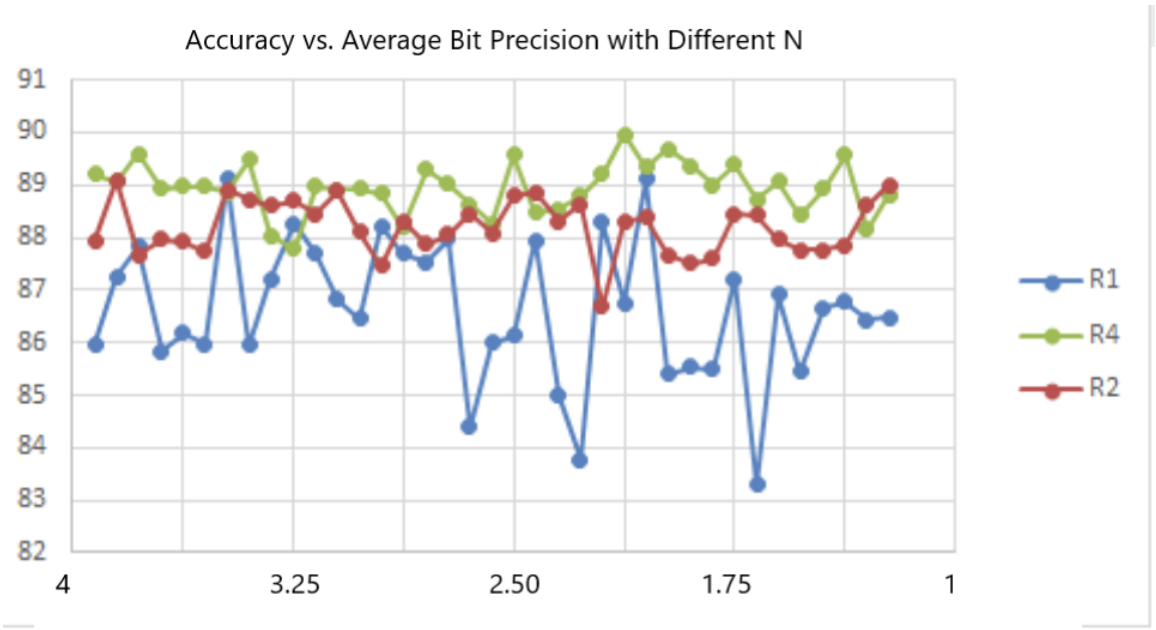


Figure 3.1: Accuracy vs. Average Bit Precision with different N. N = 1, 2 and 4 for series R1, R2 and R4 respectively.

3.3 Full-precision vs. Quantized weights

There are lots of ways to represent a layer's importance, we first tried using the norm of the weights. The intuition here is that a lower norm implies less importance for a layer. Thus, during step 4 in Algorithm 1, the layer with the lowest average norm is chosen. However, during the quantization process, we have two kinds of weights that we can use: full-precision weights and quantization weights. The results in table 3.2 show the effect of using quantized vs. full-precision weights. From the results, we can see that using full-precision weights either has a higher accuracy with similar model sizes or has a significantly smaller model size and a similar accuracy. We can conclude that using the metric of full-precision weights to measure the importance level is better than using that of the quantized weights. Therefore, for the rest of the experiments, we use full-precision weights most of the time unless specified otherwise.

Dataset	AVG Bits	Accuracy	Size (MB)
CIFAR10	4	92.45	0.14
	3	92.49	0.10
	2	90.96	0.07
CIFAR100	4	66.95	0.16
	3	66.98	0.13
	2	66.42	0.09

(a) Baseline results for APoT method with ResNet20.

Dataset	Bit Range	AVG Bits	Accuracy	Size (MB)
CIFAR10	3-4	3.90	92.66	0.14
	2-3	2.95	92.16	0.09
	1-2	1.95	91.09	0.06
CIFAR100	3-4	3.85	67.58	0.16
	2-3	2.95	67.32	0.12
	1-2	1.95	66.53	0.08

(b) Results for quantizing layers based on the average norm of full-precision wieghts with ResNet20.

Dataset	Bit Range	AVG Bits	Accuracy	Size (MB)
CIFAR10	3-4	3.80	92.56	0.14
	2-3	2.85	92.25	0.13
	1-2	1.85	90.82	0.08
CIFAR100	3-4	3.80	68.81	0.16
	2-3	2.88	66.75	0.15
	1-2	1.88	64.13	0.10

(c) Results for quantizing layers based on the average norm of quantized wieghts with ResNet20.

Table 3.2: Results for using average norm as the importance estimation for quantization.The model used is ResNet20. 'Size (MB)' here refers to the model size. Table (a) is the baseline with APoT method. Table (b) is the results using full-precision weights and (c) is the results using quantized weights.

3.4 Other Importance Estimation Metrics

3.4.1 Min-variance vs. Max-variance

Considering that the process of quantization can be summarised as rounding real numbers to discrete values, the variance of the weights can also be a good metric for measuring the importance of layers. However, it is not clear whether higher variance or lower variance makes a layer less important during the quantization process. In order to figure out which one is a better metric, we conducted experiments with the results shown in Table 3.3. We refer to the metric that shows a layer is less important as the method name. For example, for the "Max-variance" method, the layer with the highest variance is chosen in step 4 of Algorithm 1 to be quantized further. From the results, we can see that min-variance performs better than the max-variance method most of the time, which makes it a better metric. One possible explanation is that when the variance is small, the layer's weights are more closely gathered around the center and when we try to quantize that layer, we can ignore more outliers and give the center more "quantization levels". Quantizing this layer will therefore contribute less error to the final accuracy. Comparing the results in Table 3.3 with the results in Table 3.2b, we can see that using full-precision norm is a better metric than using variance as the importance estimation of convolutional layers.

3.4.2 One-shot Selection Method with Imprinting

Inspired by the work done by Elkerdawy et al. [53], we find that we could make use of the imprinting method during the layer selection process. Imprinting is a method that can quickly predict the final model accuracy given a model's weights. If we attach an imprinting block after each convolutional layer, we could get an estimation of how well the current model performs up to each convolutional layer. An example results for imprinting is shown in Figure 3.2. The x-axis is different layers in ResNet20 and the y-axis is the estimated accuracy if exiting the model using up to the corresponding

Dataset	Bit Range	AVG Bits	Accuracy	Size (MB)
CIFAR10	3-4	3.80	92.68	0.13
	2-3	2.90	91.99	0.07
	1-2	1.90	90.73	0.06
CIFAR100	3-4	3.98	67.69	0.16
	2-3	2.90	66.59	0.09
	1-2	1.90	64.49	0.08

(a) Results for quantizing layers based on the average norm of full-precision wieghts with ResNet20.

Dataset	Bit Range	AVG Bits	Accuracy	Size (MB)
CIFAR10	3-4	3.90	92.78	0.14
	2-3	2.90	91.77	0.13
	1-2	1.95	90.69	0.07
CIFAR100	3-4	3.95	67.99	0.16
	2-3	2.7	66.05	0.15
	1-2	1.95	64.55	0.09

(b) Results for quantizing layers based on the average norm of quantized wieghts with ResNet20.

Table 3.3: Results for using the variance as the importance estimation for quantization.The model used is ResNet20. 'Size (MB)' here refers to the model size. Table (a) is using min-variance to find the layer to be further quantized. Table (b) is the results using the max-variance of the weights as the importance of the layer.

layer. Using the results of this imprinting method as the importance measure will allow us to have a better interpretation of the final bit-length configuration compared to other search methods. The modified algorithm is as shown in Algorithm 2. We use the difference of the accuracy between two layers as the importance measure of the later layer. If the difference is small, that means the later layer contains less new information and therefore is less important and can be further quantized. We compared this method with the methods that use statistical criterion as the importance measure that we mentioned before and the results are shown in Table

3.4. The model used is ResNet-20 on CIFAR-10. From the results, we can see that the effect of the two criteria is quite similar. However, as the imprinting method is one-shot and the statistical criteria require fine-tuning for a couple of epochs each selection, the imprinting method is more efficient in terms of the overall time and training epochs.

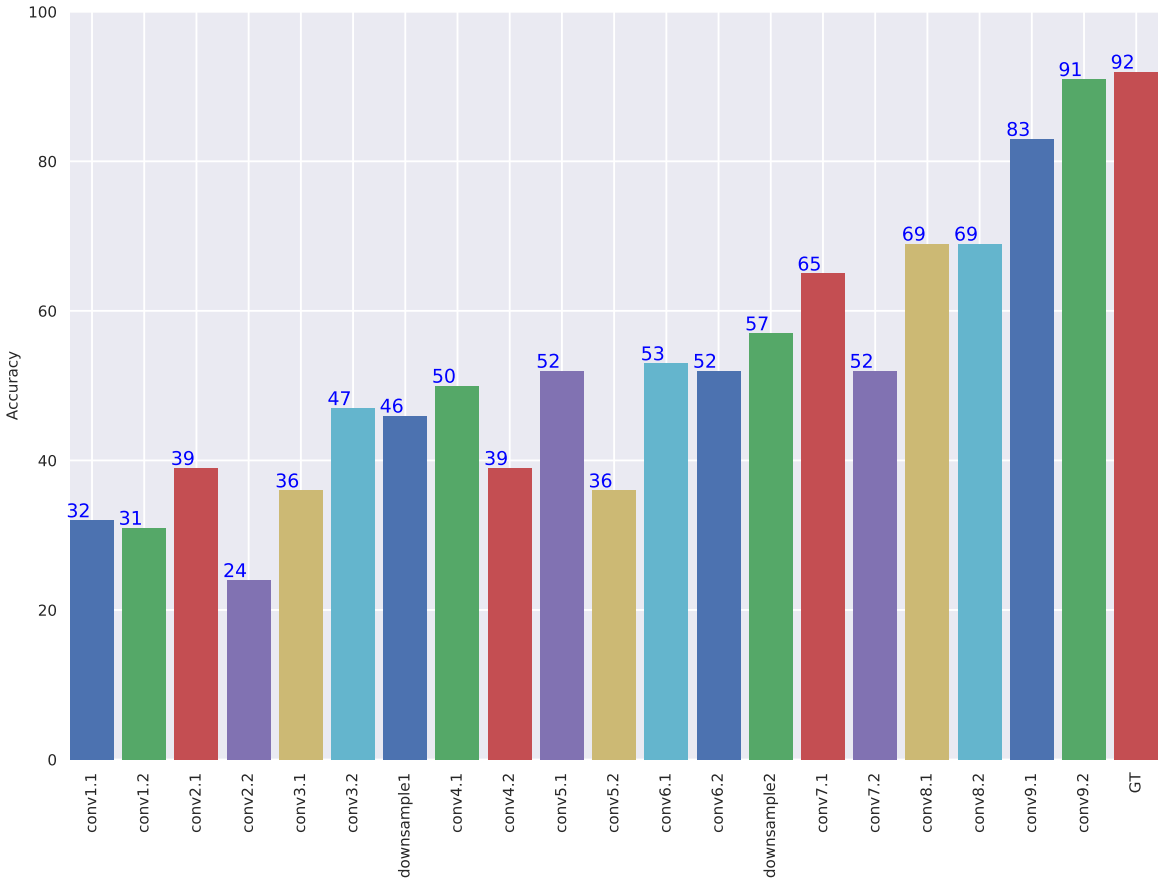


Figure 3.2: Accuracy approximation using imprinting. The model used is ResNet20 on CIFAR-10

3.5 Generalization

All the experiments above are using APoT as the base quantization method and the model and dataset are mostly ResNet20 and CIFAR. In order to see how well our method can be generalized, we also conducted experiments with (i) different models,

Criterion	Weights	Activations	Accuracy	Size (MB)	Epochs
max_variance	3.9	3.9	92.78	0.14	460
norm	3.9	3.9	92.66	0.14	460
qt_norm	3.8	3.8	92.58	0.14	460
Imprinting	3.85	3.85	92.82	0.13	300
min_variance	3.8	3.8	92.68	0.13	460
qt_norm	2.85	2.85	92.25	0.13	620
max_variance	2.9	2.9	91.77	0.13	620
Imprinting	2.3	2.7	91.65	0.1	300
norm	2.95	2.95	92.16	0.09	620
min_variance	2.9	2.35	91.99	0.07	620
qt_norm	1.85	2.8	90.82	0.09	780
max_variance	1.95	2.6	90.69	0.07	780
Imprinting	1.65	2.4	90.64	0.07	300
norm	1.95	2.2	91.09	0.06	780
min_variance	1.9	2.35	90.73	0.06	780

Table 3.4: Different bit selection criteria with ResNet-20 on CIFAR-10. The quantization method used is APoT[51]. "norm", "min_variance" and "max_variance" are the average statistics of full precision weights, whereas "qt_norm" is the average norm of quantized weights."Epochs" here means number of epochs needed to find the target configuration and fine-tune to the best accuracy.

(ii)a larger dataset, and (iii) different quantization base methods. Since the imprinting method takes less training time while giving similar or better performance and model compression levels, we will use imprinting other than the statistical criterion as the importance estimation. We compare our method with some other state-of-the-art quantization methods.

3.5.1 Different models

We applied our method in Section 3.4.2 and compared it with the original APoT method. The dataset used are CIFAR-10 and CIFAR-100 and the results are shown in Table 3.5. From the table, we can see that we can achieve results similar to APoT on most of the ResNet models, but with less average bit length and model size. We have a larger accuracy drop for MobileNet-V2 compared to APoT. However, we are still able to produce some mixed-precision configurations with less model size but higher accuracy than APoT. We also have a shorter training time than the APoT method when targeting at low bit precision.

Model	Method	Weights	Activations	Accuracy	Size (MB)	Epoch
ResNet18	Baseline	32	32	94.97	44.61	300
	APoT	4	4	94.57	5.62	300
	Ours	3.07	4.35	94.11	4.87	300
	Ours	2.7	3.11	94.02	4.79	300
	APoT	3	3	94.13	4.23	600
	Ours	1.96	2.55	93.42	2.95	300
	APoT	2	2	93.22	2.84	900
ResNet20	Baseline	32	32	92.96	1.04	300
	APoT	4	4	92.45	0.14	300
	Ours	3.85	3.85	92.82	0.13	300
	APoT	3	3	92.49	0.1	600
	Ours	2.3	2.7	91.65	0.1	300
	APoT	2	2	90.96	0.07	900
	Ours	1.65	2.4	90.64	0.07	300
ResNet56	Baseline	32	32	94.46	3.31	300
	APoT	4	4	93.93	0.42	300
	Ours	3.37	3.42	93.74	0.32	300
	APoT	3	3	93.77	0.32	600
	APoT	2	2	93.05	0.21	900
	Ours	2.37	2.75	92.89	0.2	300
	Ours	1.82	2.42	92.22	0.16	300
MobileNetV2	Baseline	32	32	94.24	8.7	300
	APoT	4	4	89.99	1.19	300
	APoT	3	3	83.85	0.92	600
	Ours	3.32	3.39	84.82	0.87	300
	APoT	2	2	69.79	0.66	900
	Ours	2.48	2.83	74.42	0.63	300
	Ours	1.32	2.14	63.92	0.55	300

(a) Comparing different Models (CIFAR-10)

Model	Method	Weights	Activations	Accuracy	Size (MB)	Epoch
ResNet18	Baseline	32	32	78.07	44.79	300
	APoT	4	4	77.75	5.8	300
	Ours	3.03	3.22	77.42	4.72	300
	APoT	3	3	76.11	4.41	600
	Ours	2.67	3.04	76.01	4.38	300
	APoT	2	2	71.7	3.01	900
	Ours	1.29	2.18	73.34	1.8	300
ResNet20	Baseline	32	32	66.93	1.09	300
	APoT	4	4	66.95	0.16	300
	Ours	3.8	3.8	67.9	0.15	300
	APoT	3	3	66.98	0.13	600
	Ours	2.3	2.75	66.42	0.12	300
	APoT	2	2	66.42	0.09	900
	Ours	1.8	2.45	64.25	0.09	300
ResNet56	Baseline	32	32	94.46	3.27	300
	APoT	4	4	93.93	0.42	300
	Ours	3.37	3.42	93.74	0.32	300
	APoT	3	3	93.77	0.32	600
	APoT	2	2	93.05	0.21	900
	Ours	2.37	2.75	92.89	0.2	300
	Ours	1.82	2.42	92.22	0.16	300
MobileNetV2	Baseline	32	32	75.58	9.13	300
	APoT	4	4	75.2	1.63	300
	Ours	3.94	3.94	75.21	1.62	300
	APoT	3	3	74.14	1.36	600
	APoT	2	2	67.4	1.09	900
	Ours	2.19	2.69	71.24	1.01	300
	Ours	1.71	2.37	62.9	1	300

(b) Comparing different Models (CIFAR-100)

Table 3.5: Comparing reults by applying our method (imprinting) on different models. Our method here refers to using imprinting as the metric for measuring the layer importance. (a) is using CIFAR10 as the dataset and (b) is using CIFAR100 as the dataset.

3.5.2 Larger Dataset

In order to test if our method works on a larger dataset. We applied the imprinting method in Section 3.4.2 with ResNet18 on the Imagenet dataset. The results are shown in Table 3.6. Our method outperforms the mixed quantization method BitPruning and the fixed quantization method PACT on a similar configuration. Although we achieve comparable accuracy to APoT, we perform quantization in constant time regardless of the budget constrain. Unlike APoT, which applies an iterative process where each fine-tuned model is used as a starting point for the next bit-width configuration. This means training time grows linearly as a lower budget is targeted.

Method	Weights	Activations	Accuracy
PACT [43]	5	5	69.8
APoT	5	5	70.75
Ours(APoT)	4.38	4.38	70.59
PACT [43]	4	4	69.2
APoT	4	4	70.74
Ours(APoT)	3.55	3.55	70.12
BitPruning [42]	3.38	4.14	69.19
PACT [43]	3	3	68.1
APoT	3	3	69.79
Ours(APoT)	2.72	2.72	69.84
APoT	2	2	66.46

Table 3.6: Results for ResNet18 on Imagenet. We compared our results with the fixed-precision methods PACT[43] and APoT[51], as well as mixed-precision method BitPruning[42].

3.5.3 Different quantization method

One advantage of our method is that it does not rely on a specific quantization method. APoT method uses non-uniform quantization which produces non-uniform

quantization ranges. To test how well our method works with other types of quantization methods, we applied it with Uniform and PoT quantizations as well and compared it with other state-of-the-art quantization methods, such as Differential Quantization (DQ) and Trained Quantization Threshold(TQT) methods. The results are shown in Table 3.7. PACT uses uniform quantization. Compared to our method with uniform quantization, PACT is outperformed. ZeroQ achieves better accuracy than our method on a similar model size but the average bit length for the activation is also higher than ours. Compared to DQ and TQT, we are able to maintain a comparable level of accuracy with a similar weight size with ResNet-20 on CIFAR-10. We can also see that as the average bit length gets smaller, the accuracy is not hurt significantly. One huge advantage of our method compared to APoT, is that, instead of using the fine-tuned pre-trained higher precision model as a starting point to train a lower bit configuration, our approach does not require such a high amount of fine-tuning epochs to reach a low bit configuration and smaller model size. It is worth mentioning that in some cases the model size can be larger even if the bit length for weights and activations are smaller. The reason is that these bit lengths are average values of all convolutional layers and some layers are much more compressed than in our mixed-precision quantized model than their counterpart fixed-precision quantized models.

Method	Weight	Activation	Size (MB)	Accuracy	Epoch
PACT [43]	4	4	-	91.3	-
APoT [51]	4	4	0.14	92.45	300
PoT	4	4	0.14	91.85	300
Uniform	4	4	0.14	92.86	300
ZeroQ [44]	learned	8	0.13	93.16	-
Ours(APoT)	3.85	3.85	0.13	92.82	300
Ours(Uniform)	3.1	3.3	0.12	92.04	300
Ours(PoT)	3.3	3.4	0.11	91.37	300
PACT [43]	3	3	-	91.1	-
Ours(Uniform)	2.8	3.15	0.11	91.87	300
APoT [51]	3	3	0.1	92.49	600
PoT	3	3	0.1	91.78	600
Uniform	3	3	0.1	92.36	600
Ours(PoT)	2.8	3	0.1	91.29	300
Ours(APoT)	2.3	2.7	0.1	91.65	300
PACT [43]	2	2	-	89.7	-
DQ(Uniform) [54]	learned	4	0.07	91.42	160
DQ(POT) [54]	learned	4	0.07	88.77	160
APoT[51]	2	2	0.07	90.96	900
PoT	2	2	0.07	91.14	900
Uniform	2	2	0.07	91.06	900
Ours(PoT)	1.7	2.4	0.07	90.28	300
TQT (Uniform) [55]	2.3	4	0.065	90.83	160
TQT (POT) [55]	2.3	4	0.065	88.71	160
Ours(APoT)	1.6	2.4	0.06	90.64	300
Ours(Uniform)	1.5	2.3	0.06	90.04	300

Table 3.7: Comparing different quantization methods with ResNet20 on CIFAR-10

Chapter 4

Gradients Estimation for Quantization

4.1 Different Gradient Estimation Methods

In this section, we will see different existing methods that try to solve the non-differential problem of quantization.

4.1.1 Straight Through Estimator

As we already know, the quantization process is a non-differential process as it contains rounding a real number to a discrete value. In order to overcome this non-differential property during the back-propagation, most of the current quantization work uses the Straight Through Estimator (STE) [48] to estimate the gradients of the quantization. The idea of STE is fairly simple. It just copies the gradient with respect to the stochastic output directly as an estimator of the gradient with respect to the sigmoid argument. In other words, it means backpropagate through the hard threshold function as if it had been the identity function. In the case of quantization, it's the projection function. That is to say that STE assumes that:

$$\frac{\partial \Pi_{Q(\alpha,b)} \lfloor W, \alpha \rceil}{\partial \lfloor W, \alpha \rceil} = 1 \tag{4.1}$$

where Π is the projection function. It means the variable before and after projection are treated the same in back propagation.

Yin et al. [56] justified that STE is properly chosen for estimating the gradients of the quantized neural networks and searching in the negative direction of the gradients that uses STE minimizes the training loss, even though the gradients that use STE is not the gradients of any function. However, since this STE is still just an estimation of the actual gradients, it contributes errors into the network and will affect the final accuracy.

4.1.2 Other Estimation methods

STE is a quick and easy way to estimate the gradients of the projection function, but researchers are still trying to find ways to decrease the error introduced by this gradient estimation.

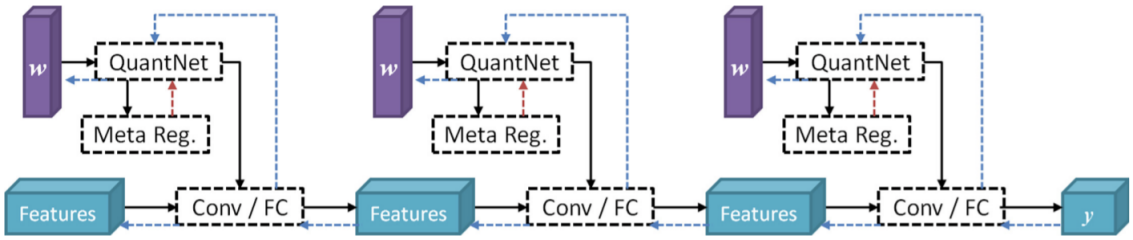


Figure 4.1: Pipeline of QuantNet [57] for binarizing neural networks. The solid lines are the forward pass and the dashed blue lines are the backward pass. The dashed red lines are the gradients flow from the meta regularization.

Liu et al. [57] introduced QuantNet that binarizes a full precision network without using STE or any estimator. The idea is to directly binarize the weights with the help of another neural network called a meta-based quantizer. The binarized network and the quantizer are trained and optimized jointly. The whole architecture presented by Liu et al. is shown in Figure 4.1. A meta quantizer, QuantNet, is attached to each layer of the network and is used to generate the binarized weights. The quantizer consists of three steps: encoding, compressing, and decoding. The sign of the quantized weights will be used as the weights for the binarized network. Some results of the QuantNet are shown in Table 4.1. More results are included in the original paper. From their results, we can see that their method can achieve higher

accuracy than the methods that use STE, which is commonly used in the quantization of neural networks. However, attaching a new network to each layer will add lots of overheads during the training process, especially when the network is deep. According to [57], it takes 1.7x the training time of the full precision training for ResNet34.

Method	Bit-width W/A	ResNet56	ResNet110
FP	32/32	71.22	72.54
DoReFa-Net [47]	1/2	66.42	66.83
QuantNet	1/2	69.38	70.17

Table 4.1: Top-1 accuracy on CIFAR-100. Comparison of QuantNet with the existing methods on ResNet-56 and ResNet-110

Zhuang et al. [58] propose to add a full-precision auxiliary module to guide the training of the low-precision network. During the training process, they connect the full precision module to the quantized network and train this newly formed mixed-precision network jointly with the quantized network through weight sharing. The process is shown in Figure 4.2. The training loss is the combined loss of the quantized network and the mixed-precision network (i.e. $\mathcal{L} + \mathcal{L}_{\text{aux}}$). During the inference time,

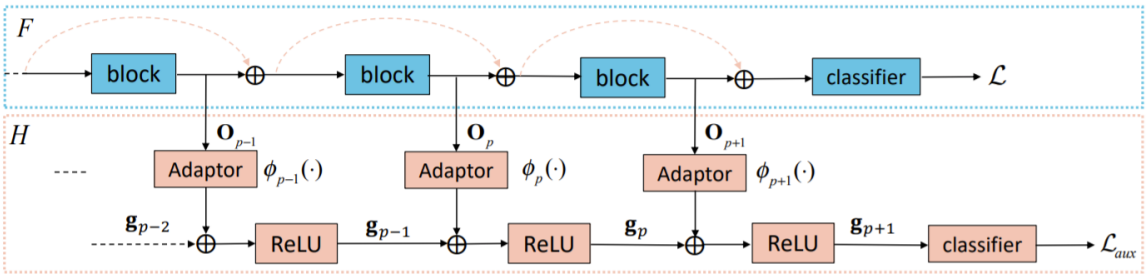


Figure 4.2: A overview of the method proposed in [58] Blue color represents low-precision operations and pink color represents full-precision operations.

only the quantized network is used. This auxiliary module can be added to many different existing quantization methods. The results in Table 4.2 show that the use of the full precision weights can help avoid using STE and increase the performance

of the quantized model.

Model	Method	Top-1 Accuracy
ResNet101	DoReFa-Net (2-bit)	70.8
	DoReFa-Net + Auxi	74.6
ResNet50	LQ-Net (3-bit)	74.2
	LQ-Net + Auxi	75.4
ResNet18	BiReal-Net	56.4
	BiReal-Net + Auxi	64.8

Table 4.2: Accuracy of different comparing methods on the ImageNet validation set.

4.2 Experiments for Gradients Estimations for Quantization

In this section, we will talk about our experiments for gradient estimation quantization and discuss the results.

4.2.1 Using full-precision vs. quantized weights

For most of the quantization methods, during the forward pass, the quantized input and weights are used. During the backward pass, the gradients for the quantized weights are estimated using STE, which will accumulate errors that will affect the model performance. To make up for the error accumulated by estimating the gradients of quantization, we try to make use of the full-precision weights that we have access to, during the training time. That is, using full-precision weights and inputs whenever quantized weights and inputs are needed originally during the backward pass.

We applied this"new gradients" method on APoT and vanilla uniform quantization. The results are shown in Table 4.3. We can see that this method works fine with high bit precision, but the accuracy drops a lot after further quantization.

In order to find out the reason for the accuracy drop, we plot the histogram of

Method	Weight Bits	Activation Bits	Accuracy
FP	32	32	92.96
APoT	4	4	92.45
APoT (new gradients)	4	4	92.69
Uniform (new gradients)	4	4	91.23
APoT	3	3	92.49
APoT (new gradients)	3	3	90.62
Uniform (new gradients)	3	3	83.29
APoT	2	2	90.96
APoT (new gradients)	2	2	70.31

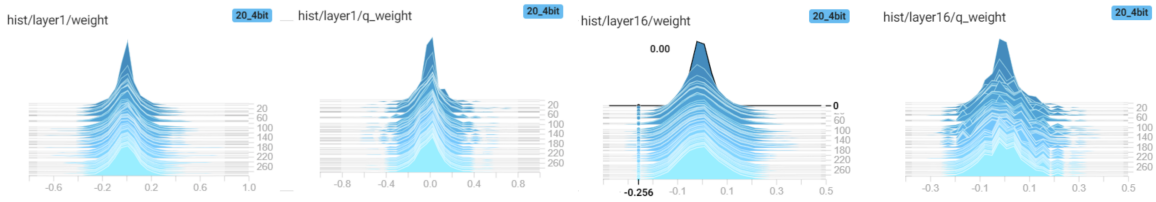
(a) Accuracy of training ResNet-20 on CIFAR10

	Weight Bits	Activation Bits	Accuracy
FP	32	32	94..46
APoT	4	4	93.93
APoT (new gradients)	4	4	94.13
Uniform (new gradients	4	4	91.73
APoT	3	3	93.77
APoT (new gradients)	3	3	92.61
Uniform (new gradients	3	3	84.21
APOT	2	2	93.05
APoT (new gradients)	2	2	81.56

(b) Accuracy of training ResNet-56 on CIFAR10

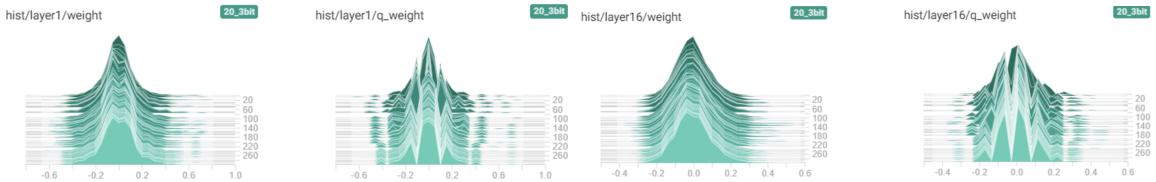
Table 4.3: Results for using full-precision weights in backward pass. $'APoT(newgradients)'$ means applying the method with APoT quantization. $'Uniform(newgradients)'$ means applying the method with Uniform quantization

the weights of some layers on ResNet20, as shown in Figure 4.3. From Figure A.2a and A.2b, we can see that the full-precision weights and the quantized weights have a similar histogram. However, as the precision goes lower, the histogram starts to show a noticeable difference between full-precision weights and the quantized weights,



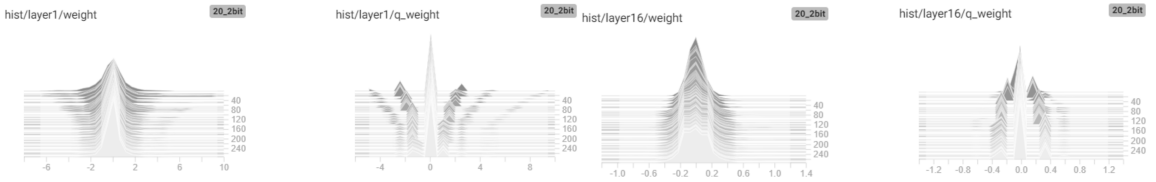
(a) Histogram of full-precision wieghts (left) vs. quantized weights (right) of the 1st layer of ResNet20 when bit precision is 4.

(b) Histogram of full-precision wieghts (left) vs. quantized weights (right) of the 16th layer of ResNet20 when bit precision is 4.



(c) Histogram of full-precision wieghts (left) vs. quantized weights (right) of the 1st layer of ResNet20 when bit precision is 3.

(d) Histogram of full-precision wieghts (left) vs. quantized weights (right) of the 16th layer of ResNet20 when bit precision is 3.



(e) Histogram of full-precision wieghts (left) vs. quantized weights (right) of the 1st layer of ResNet20 when bit precision is 2.

(f) Histogram of full-precision wieghts (left) vs. quantized weights (right) of the 16th layer of ResNet20 when bit precision is 2.

Figure 4.3: Histogram of full-precision weights (left) and quantized weights(right) for the 1st and 16th layer in ResNet20. (a) and (b) are when the bit precision is 4. (c) and (d) are when the bit precision is 3. (e) and (f) are when the bit precision is 2.

as shown in Figure A.2c to 4.3e. Thus, using full-precision weights to calculate the gradients during the back-propagation will introduce more errors as the bit precision goes lower, which could cause the accuracy to drop significantly.

Chapter 5

Conclusions, Recommendations, & Future Work

5.1 Mixed Precision Quantization

5.1.1 Conclusions

In this thesis, we conclude that it is recommended to have a mixed-precision quantization scheme to compress neural networks. We have also introduced a bit-length selection method to identify and rank the importance of each convolutional layer by using one-shot imprinting and other estimation metrics. This method gives us the ability to convert any fixed-precision quantization method into mixed-precision, which usually produces neural network models with smaller model sizes. The use of imprinting also reduces the training epochs required to reach a relatively low average bit length. We have acquired comparable results on CIFAR-10, CIFAR-100, and ImageNet, compared to APoT and other state-of-the-art mixed-precision quantization methods.

5.1.2 Future Work

Our current approach to search for a suitable bit configuration is almost a greedy solution. In the future, we can explore more options on the search method. For example, using a reinforcement learning agent to automatically search the configuration space may help reduce the use of labor. We could also add an assisting network to

predict the compress ratio of the bit configurations and use that instead of greedy searching.

Another direction for future work could be combining the quantization with other methods to achieve more compression ratio or faster inference time. As mentioned in Chapter 2, many other model compression techniques can be combined together. Recently, Kim et al. [59] introduced their PQK model compression method that consists of pruning, quantization, and knowledge distillation. We could try pruning combined with our quantization using imprinting.

5.2 Gradients Estimation Methods

5.2.1 Conclusions

In this thesis, we have seen different estimation methods that allow the gradients to propagate smoothly in the backward propagation. We have also learned from the experiments that directly using full precision weights instead of quantized ones won't make up with the error introduced by using STE and it will even contribute more error due to the different distribution between the weights before quantization and after.

5.2.2 Future Work

Even though directly using full precision weights to replace the quantized weights will not give us any performance improvement. We can still make use of full precision weights. Our potential future work is to use both full precision weights and quantized weights together to avoid dealing with the non-differential function. For example, we can use the full precision weights to guide the quantized weights to flow in the backward propagation. Another idea is to train a neural network that represents the process of non-differential projection function because of the universal approximation property of neural networks. Then we can use the gradients of such network in the backpropagation of the original quantized neural network. It is an idea similar to the

QuantNet [57] we have discussed in Section 4.1.2, but instead of especially targeting Binary Networks, we can focus on working with any kind of quantized networks.

References

[1] H. Liu, S. Elkerdawy, N. Ray, and M. Elhoushi, "Layer importance estimation with imprinting for neural network quantization," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021, pp. 2408–2417. DOI: 10.1109/CVPRW53098.2021.00273.

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

[3] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. arXiv: 1704.04861. [Online]. Available: http://arxiv.org/abs/1704.04861.

[4] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[5] A. Howard *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[6] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[7] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[9] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[10] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, 2015. arXiv: 1503.02531 [stat.ML].

[11] Y. Tian, D. Krishnan, and P. Isola, "Contrastive representation distillation," *CoRR*, vol. abs/1910.10699, 2019. arXiv: 1910.10699. [Online]. Available: http://arxiv.org/abs/1910.10699.

[12] T. Furlanello, Z. C. Lipton, M. Tschannen, L. Itti, and A. Anandkumar, *Born again neural networks*, 2018. arXiv: 1805.04770 [stat.ML].

[13] M. Gao, Y. Shen, Q. Li, and C. C. Loy, "Residual knowledge distillation," *CoRR*, vol. abs/2002.09168, 2020. arXiv: 2002.09168. [Online]. Available: https://arxiv.org/abs/2002.09168.

[14] T. He, C. Shen, Z. Tian, D. Gong, C. Sun, and Y. Yan, "Knowledge adaptation for efficient semantic segmentation," *CoRR*, vol. abs/1903.04688, 2019. arXiv: 1903.04688. [Online]. Available: http://arxiv.org/abs/1903.04688.

[15] S. Anwar and W. Sung, "Coarse pruning of convolutional neural networks with random masks," 2017.

[16] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *CoRR*, vol. abs/1810.05270, 2018. arXiv: 1810.05270. [Online]. Available: http://arxiv.org/abs/1810.05270.

[17] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2, Morgan-Kaufmann, 1990. [Online]. Available: https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

[18] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems*, S. Hanson, J. Cowan, and C. Giles, Eds., vol. 5, Morgan-Kaufmann, 1993. [Online]. Available: https://proceedings.neurips.cc/paper/1992/file/303ed4c69846ab36c2904d3ba85730
Paper.pdf.

[19] T. Zhang *et al.*, "A systematic DNN weight pruning framework using alternating direction method of multipliers," *CoRR*, vol. abs/1804.03294, 2018. arXiv: 1804.03294. [Online]. Available: http://arxiv.org/abs/1804.03294.

[20] 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. DOI: 10.1109/5.726791.

[21] X. Ma *et al.*, "PCONV: the missing but desirable sparsity in DNN weight pruning for real-time execution on mobile devices," *CoRR*, vol. abs/1909.05073, 2019. arXiv: 1909.05073. [Online]. Available: http://arxiv.org/abs/1909.05073.

[22] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," *CoRR*, vol. abs/1707.06168, 2017. arXiv: 1707.06168. [Online]. Available: http://arxiv.org/abs/1707.06168.

[23] T.-W. Chin, R. Ding, C. Zhang, and D. Marculescu, "Towards efficient model compression via learned global ranking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[24] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[25] Y. He, P. Liu, Z. Wang, and Y. Yang, "Pruning filter via geometric median for deep convolutional neural networks acceleration," *CoRR*, vol. abs/1811.00250, 2018. arXiv: 1811.00250. [Online]. Available: http://arxiv.org/abs/1811.00250.

[26] M. Lin *et al.*, "Hrank: Filter pruning using high-rank feature map," *CoRR*, vol. abs/2002.10179, 2020. arXiv: 2002.10179. [Online]. Available: https://arxiv.org/abs/2002.10179.

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

[28] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K. Cheng, "Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm," *CoRR*, vol. abs/1808.00278, 2018. arXiv: 1808.00278. [Online]. Available: http://arxiv.org/abs/1808.00278.

[29] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf.

[30] F. Li and B. Liu, "Ternary weight networks," *CoRR*, vol. abs/1605.04711, 2016. arXiv: 1605.04711. [Online]. Available: http://arxiv.org/abs/1605.04711.

[31] P. Wang and J. Cheng, "Fixed-point factorized networks," *CoRR*, vol. abs/1611.01972, 2016. arXiv: 1611.01972. [Online]. Available: http://arxiv.org/abs/1611.01972.

[32] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding*, 2016. arXiv: 1510.00149 [cs.CV].

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

[34] P. Stock, A. Joulin, R. Gribonval, B. Graham, and H. Jégou, "And the bit goes down: Revisiting the quantization of neural networks," *CoRR*, vol. abs/1907.05686, 2019. arXiv: 1907.05686. [Online]. Available: http://arxiv.org/abs/1907.05686.

[35] M. Á. Carreira-Perpiñán, "Model compression as constrained optimization, with application to neural nets. part I: general framework," *CoRR*, vol. abs/1707.01209, 2017. arXiv: 1707.01209. [Online]. Available: http://arxiv.org/abs/1707.01209.

[36] M. Courbariaux, Y. Bengio, and J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *CoRR*, vol. abs/1511.00363, 2015. arXiv: 1511.00363. [Online]. Available: http://arxiv.org/abs/1511.00363.

[37] H. Qin *et al.*, "Ir-net: Forward and backward information retention for highly accurate binary neural networks," *CoRR*, vol. abs/1909.10788, 2019, Withdrawn. arXiv: 1909.10788. [Online]. Available: http://arxiv.org/abs/1909.10788.

[38] P. Wang, Q. Hu, Y. Zhang, C. Zhang, Y. Liu, and J. Cheng, "Two-step quantization for low-bit neural networks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4376–4384. DOI: 10.1109/CVPR.2018.00460.

[39] N. Mellempudi, A. Kundu, D. Mudigere, D. Das, B. Kaul, and P. Dubey, "Ternary neural networks with fine-grained quantization," *CoRR*, vol. abs/1705.01462, 2017. arXiv: 1705.01462. [Online]. Available: http://arxiv.org/abs/1705.01462.

[40] F. Zhu *et al.*, "Towards unified INT8 training for convolutional neural network," *CoRR*, vol. abs/1912.12607, 2019. arXiv: 1912.12607. [Online]. Available: http://arxiv.org/abs/1912.12607.

[41] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding*, 2016. arXiv: 1510.00149 [cs.CV].

[42] M. Nikolic *et al.*, "Bitpruning: Learning bitlengths for aggressive and accurate quantization," *CoRR*, vol. abs/2002.03090, 2020. arXiv: 2002.03090. [Online]. Available: https://arxiv.org/abs/2002.03090.

[43] J. Choi, Z. Wang, S. Venkataramani, P. I. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: parameterized clipping activation for quantized neural networks," *CoRR*, vol. abs/1805.06085, 2018. arXiv: 1805.06085. [Online]. Available: http://arxiv.org/abs/1805.06085.

[44] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer, "Zeroq: A novel zero shot quantization framework," *CoRR*, vol. abs/2001.00281, 2020. arXiv: 2001.00281. [Online]. Available: http://arxiv.org/abs/2001.00281.

[45] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010, pp. 807–814. [Online]. Available: https://icml.cc/Conferences/2010/papers/432.pdf.

[46] 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. arXiv: 1609.07061. [Online]. Available: http://arxiv.org/abs/1609.07061.

[47] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *CoRR*, vol. abs/1606.06160, 2016. arXiv: 1606.06160. [Online]. Available: http://arxiv.org/abs/1606.06160.

[48] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *CoRR*, vol. abs/1308.3432, 2013. arXiv: 1308.3432. [Online]. Available: http://arxiv.org/abs/1308.3432.

[49] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *CoRR*, vol. abs/1603.01025, 2016. arXiv: 1603.01025. [Online]. Available: http://arxiv.org/abs/1603.01025.

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

[51] Y. Li, X. Dong, and W. Wang, "Additive powers-of-two quantization: A non-uniform discretization for neural networks," *CoRR*, vol. abs/1909.13144, 2019. arXiv: 1909.13144. [Online]. Available: http://arxiv.org/abs/1909.13144.

[52] H. Qi, M. Brown, and D. G. Lowe, "Learning with imprinted weights," *CoRR*, vol. abs/1712.07136, 2017. arXiv: 1712.07136. [Online]. Available: http://arxiv.org/abs/1712.07136.

[53] S. Elkerdawy, M. Elhoushi, A. Singh, H. Zhang, and N. Ray, "One-shot layer-wise accuracy approximation for layer pruning," in *2020 IEEE International Conference on Image Processing (ICIP)*, 2020, pp. 2940–2944. DOI: 10.1109/ICIP40778.2020.9191238.

[54] S. Uhlich *et al.*, "Differentiable quantization of deep neural networks," *CoRR*, vol. abs/1905.11452, 2019. arXiv: 1905.11452. [Online]. Available: http://arxiv.org/abs/1905.11452.

[55] S. R. Jain, A. Gural, M. Wu, and C. Dick, "Trained uniform quantization for accurate and efficient neural network inference on fixed-point hardware," *CoRR*, vol. abs/1903.08066, 2019. arXiv: 1903.08066. [Online]. Available: http://arxiv.org/abs/1903.08066.

[56] P. Yin, J. Lyu, S. Zhang, S. J. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," *CoRR*, vol. abs/1903.05662, 2019. arXiv: 1903.05662. [Online]. Available: http://arxiv.org/abs/1903.05662.

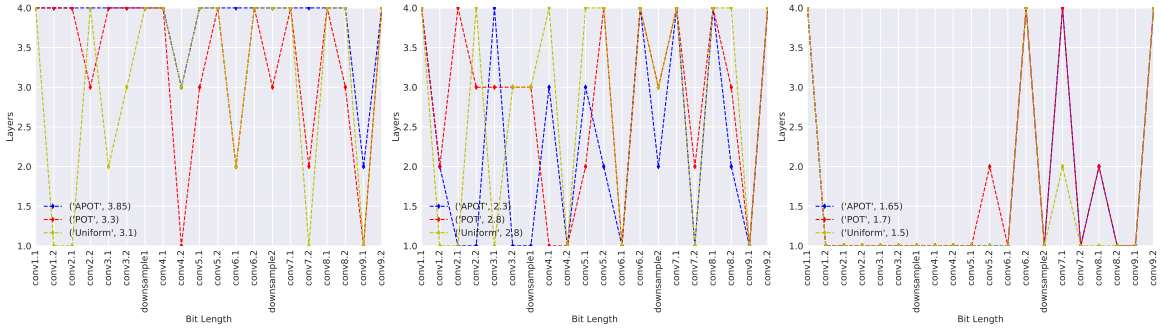
[57] J. Liu *et al.*, "Quantnet: Learning to quantize by learning within fully differentiable framework," in *Computer Vision – ECCV 2020 Workshops*, A. Bartoli and A. Fusiello, Eds., Cham: Springer International Publishing, 2020, pp. 38–53, ISBN: 978-3-030-68238-5.

[58] B. Zhuang, L. Liu, M. Tan, C. Shen, and I. Reid, "Training quantized neural networks with a full-precision auxiliary module," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[59] J. Kim, S. Chang, and N. Kwak, "PQK: model compression via pruning, quantization, and knowledge distillation," *CoRR*, vol. abs/2106.14681, 2021. arXiv: 2106.14681. [Online]. Available: https://arxiv.org/abs/2106.14681.

Appendix A: More Detailed Results for Mixed Precision Quantization

Here we provide more result details of the mixed precision quantization method that we introduced in Section **??**. In Figure A.1, we show our final bit-length configuration that's selected by the algorithm 2 with ResNet20 as the model and CIFAR10 as the dataset. Figure A.2 and Figure A.3 shows the intermediate accuracy estimation results with and without a small fine-tuning step before imprinting. The small imprinting will help to keep the accuracy of the last layer close to the ground truth, but it won't affect how the layers are selected. For reducing training time purpose, we do not include it in algorithm 2.

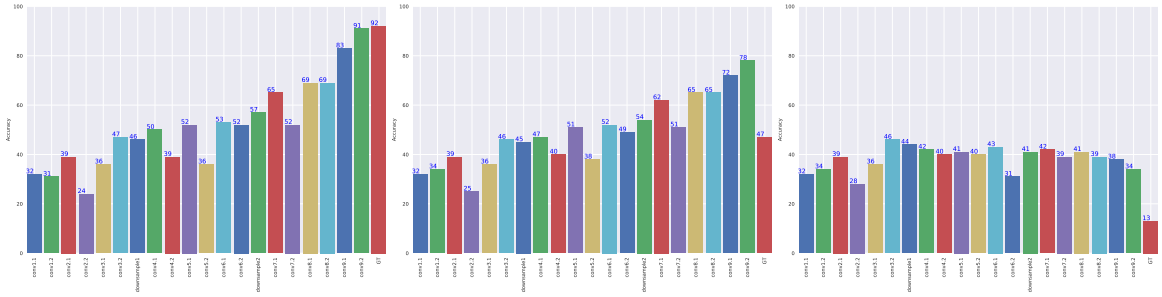


(a) The final bit-length configuration for models with average bit length $\in [3, 4]$

(b) The final bit-length configuration for models with average bit length $\in [2, 3]$

(c) The final bit-length configuration for models with average bit length $\in [1, 2]$

Figure A.1: The final big-length configuration for our ResNet20 models trained on CIFAR-10

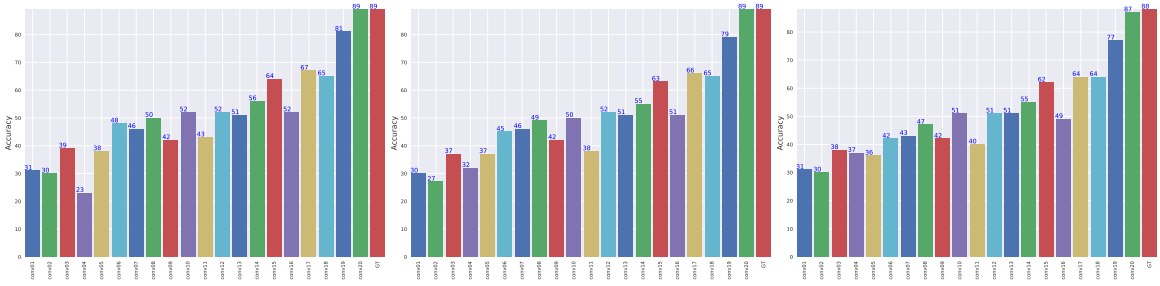


(a) An iteration from early stage. The second convolutional layer in the 8th block for ResNet20 will reduce its bitwidth by 1

(b) An iteration from middle stage. The second convolutional layer in the 8th block for ResNet20 will reduce its bitwidth by 1

(c) An iteration from late stage. The first convolutional layer in the 5th block for ResNet20 will reduce its bitwidth by 1

Figure A.2: Evolution of layer ranking in iterative imprinting. The model used is ResNet20 on CIFAR-10 dataset.



(a) An iteration from early stage. The second convolutional layer in the 1st block for ResNet20 will reduce its bitwidth by 1

(b) An iteration from middle stage. The first downsample layer (conv7 in the figure) for ResNet20 will reduce its bitwidth by 1

(c) An iteration from late stage. The second convolutional layer in the 6th block for ResNet20 will reduce its bitwidth by 1

Figure A.3: Evolution of layer ranking in iterative imprinting. We added a fine-tuning of 8 epochs between interations compared to Figure A.2. The model used is ResNet20 on CIFAR-10 dataset.