A Comprehensive Survey on Model Quantization for Deep Neural Networks

BABAK ROKH
Department of Computer Engineering, University of Zanjan, Zanjan, Iran, babak.rokh@znu.ac.ir

ALI AZARPEYVAND
Department of Electrical and Computer Engineering, University of Zanjan, Zanjan, Iran, azarpeyvand@znu.ac.ir

ALIREZA KHANTEYMOORI
Neurozentrum Department, Universitätsklinikum Freiburg, Freiburg, Germany, alireza.khanteymoori@uniklinik-freiburg.de

Recent advances in machine learning by Deep Neural Networks (DNNs) are significant. DNNs accompany with a vast array of parameters and computation, which leads to processing challenges on devices with limited hardware resources. In order to reduce parameters and computations for constructing optimized accelerators, compression techniques are suggested. A promising approach is quantization, where the full-precision values are stored in low bit-width precision, resulting in memory saving, and replacement of high-cost operations with low-cost ones. DNN quantization is flexible and efficient in hardware design, which led many methods to propose it. The necessity of an integrated report for understanding, analyzing, and comparing quantization methods cannot be overstated. Consequently, it is intended to provide a comprehensive survey of quantization concepts and categorize methods from different perspectives. In this regard, the clustering-based quantization methods are described, and it is discussed how a scale factor parameter is applied to approximate the full-precision values in quantization. For the first time, we review the training of a quantized DNN and using Straight-Through Estimator thoroughly. Replacement of the floating-point operations with low-cost bitwise ones in the quantized DNN and the sensitivity of different layers in quantization are investigated. Finally, the quantization methods are evaluated, and the accuracy of the state-of-the-art methods on CIFAR-10 and ImageNet is presented.

Keywords: Quantization, Model compression, Deep neural network acceleration, Discrete neural network optimization, Straight-through estimator.

1 INTRODUCTION

Deep Neural Networks (DNNs) have grown significantly in the recent decade, and results are considerable in some areas of machine learning, such as image classification [1, 2], Natural Language Processing (NLP) [3-5], and speech recognition [6, 7]. First, in the ImageNet challenge in 2012, AlexNet [1], a Deep Convolutional Neural Network (DCNN), achieved the best accuracy among all the models. DCNNs then developed and achieved higher accuracy in the ImageNet challenge [1, 2, 8, 11]. By adding more layers, they have outperformed humans in terms of image classification accuracy in some cases [8, 9]. However, DCNNs have a huge number of parameters for storage and heavy computations that pose a challenge for being used on usual hardware. Table 1 shows some of the most popular DCNNs with their storage and computational specifications. The main operation in DCNNs is Multiply-ACCumulate (MAC) in Convolution and Fully-Connected (FC) layers.
Using DNNs in the inference phase, in comparison with the training, is more challenging as a model is installed on an end-user device such as a smartphone or an embedded device with limited hardware resources and run many times; as well, it is commonly real-time, and there is a pre-defined certain limitation for the user request and the model response. But the training phase is typically conducted once for the model building and sometimes at periodic intervals for model updating. Moreover, it is possible to perform this phase on high-performance servers and in a distributed way.

Accordingly, optimization and speed-up of DNNs processing are essential. Thus, the topic “acceleration and compression of DNNs” has been widely raised recently. In designing accelerators, researchers focus on network compression, parallel processing, and optimizing memory transfer for processing speed-up [15-39].

In the early years, the focus was on hardware optimization for processing speed-up in DNN accelerators [26, 27, 40, 41]. Later some researchers concluded that compression and software optimization of DNNs can be more efficient before touching hardware. Therefore, many works have suggested DNN compression by eliminating unnecessary parameters and computations with acceptable accuracy [17-19, 21-25, 42-46]. Figure 1 shows the designing stages in two generations of DNN accelerators.

Compression has been a topic in neural networks since the end of the 1980s [47, 48]. With development of using DNNs on limited hardware resources devices, compression techniques are more promising for decreasing parameters and computations, resulting in reduction of memory access which typically consumes high energy. A smaller model is saved on an energy-efficiency on-chip memory rather than high energy consumption off-chip DRAM. Moreover, the deployment of a compressed model on hardware becomes cheaper since it requires fewer hardware resources and smaller chip die area. Figure 1b represents approaches in DNN compression as are listed as

1. **Quantization** approximates the network components with low bit-width precision [17, 21-25, 42]. For instance, 32-bit floating-point weights are mapped to a 16-bit fixed-point or 8-bit integer.

2. **Pruning** is one of the first approaches for compressing neural networks. This method is used for removing unnecessary or less important connections in the neural network and making a sparse network which reduces memory usage as well as computations. Papers [47, 50] are among the first network pruning works done on shallow neural networks. In recent years, this approach has been widely used in DNN compression [17-19, 21, 25-26, 42-46].

3. **Low-rank approximation**, as an approach to simplify matrices and images, makes a new matrix close to the weight matrix which has lower dimensions and fewer computations in DNNs. Works [46, 61-66] employed low-rank approximation approach for DNN compression.

4. **Knowledge Distillation (KD)**, also known as teacher-student, is considered a complex model as a teacher trains a simple model and finally the complex is replaced with the simple. The teacher model is trained and then the

---

**Table 1: Specifications of Some DCNNs**

| DCNN                  | Input size | Number of Conv layers | Number of FC layers | number of weights | Total MACs ($\times 10^9$) |
|-----------------------|------------|-----------------------|---------------------|------------------|---------------------------|
| AlexNet [1]           | 227×227    | 5                     | 3                   | 61M              | 0.72                      |
| OverFeat [10]         | 231×231    | 5                     | 3                   | 146M             | 2.8                       |
| VGGNet-16 [11]        | 224×224    | 13                    | 3                   | 138M             | 15.5                      |
| GoogleNet (Inception-V1) [2] | 224×224 | 57                    | 1                   | 7M               | 1.43                      |
| ResNet-50 [8]         | 224×224    | 49                    | 1                   | 25.5M            | 3.9                       |
| SqueezeNet [12]       | 224×224    | 26                    | 0                   | 1.2M             | 1.7                       |
| MobileNetV1 [13]      | 224×224    | 27                    | 1                   | 4.2M             | 0.57                      |
| ShuffleNet [14]       | 224×224    | 49                    | 1                   | 1.4M             | 0.14                      |
knowledge of the teacher model is used for training the student model. In fact, the goal in KD is hiring a simple model with the generalization and accuracy close to the complex model. Thus, the KD approach is effective for reducing the number of parameters and computations in DNNs. Works [34-39] used KD technique for DNN compression.

Some works employed multiple approaches to have more efficient compression. For instance, works [17, 67, 68] used pruning and quantization, and [69] utilized the combination of KD and quantization. Some works suggested joint optimization of the compression approaches instead of independent optimization to increase the effect of compression [70, 71]. In this work, we focus on the quantization approach for DCNN compression.

One of the important approaches to compress DNNs is quantization. Quantization is frequently performed for implementing optimal hardware. It is also applied for reducing memory usage in many applications. The quantization of neural networks dates back to the beginning of the 1990s and some methods were adopted for quantization of the weights in shallow networks [72-75]. Since 2014, many methods have been proposed for DNN quantization [21-25]. Here, we survey quantization methods for DNNs from different perspectives and then mention the advantages and challenges of DNN quantization. Some important advantages of quantization are:

1. High compression is achieved by quantization compared to other approaches, whereas the accuracy reduction is less [76].
2. One of the reasons for the excessive use of quantization is its flexibility. Since quantization is not dependent on the network architecture, a quantization algorithm can be used for all types of DNNs. Many quantization methods designed for DCNNs are used for Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) [77-79].
3. In quantization, the floating-point operations are replaced with low-cost operations which requires a small number of cycles on hardware, such as FPGAs.
4. Quantization reduces the cost of hardware accelerator design. For instance, in 1-bit quantization, a 32-bit floating-point multiplier can be replaced by an XNOR operator which decreases the cost by the rate of 200 in Xilinx FPGA [77].
5. As quantization presents simpler parameters for the network, it can help to control overfitting.

A small fraction of recent survey papers focused on quantization methods and most of them reviewed all DNNs compression and acceleration approaches and allocated a small section to quantization [80-90]. Paper [91] focused on the pruning and quantization approaches and discussed some basic concepts of them. Papers [92-94] surveyed quantization methods. Qin et al. [92] almost had a comprehensive discussion on binary quantization. Guo et al. [93] reviewed quantization methods from four perspectives: network components that can be quantized, deterministic and stochastic quantization, fixed and adaptive codebook quantization, and quantization during and after training. Gholami et al. [94] explained the uniform and non-uniform quantization, distillation-assisted quantization, hardware-aware quantization, and extreme quantization (quantization in the low bit-width) and referred to quantization on processors.

The present paper is aimed at concentrating on the quantization methods in the image classification task for DCNNs, mentioning the concepts of quantization thoroughly and then reviewing the previous methods in some categories from different perspectives. It describes using the scale factor and the clustering-based approach for approximating the quantization levels accurately. Additionally, the training of a quantized neural network and using Straight-Through Estimator (STE) in the Error Back Propagation (EBP) algorithm is reviewed comprehensively. The effect of quantization on the MAC operation of DCNNs and the sensitivity of DCNNs layers in quantization, and mixed-precision quantization are discussed. Finally, we denote the common benchmark datasets for evaluating the methods and present the results of state-of-the-art (SOA) methods.

The organization of the remaining sections is summarized as follows. In section 2, preliminary concepts of neural networks are described. In section 3, the basic concepts of quantization are explained. Section 4 details the training of a quantized network. Section 5 describes the simple operations in a quantized DCNN. In section 6, the sensitivity of DCNN layers in the quantization is discussed. Section 7 presents the evaluation of SOA methods, analysis of the results, and discussion. Finally, in section 8, there are conclusions and some suggestions for future researches.

2 PRELIMINARY CONCEPTS OF NEURAL NETWORKS

A neural network is an acyclic graph consisting of some nodes (neurons) organized in multiple layers. Neurons in each layer are connected to neurons in another layer. In general, each neural network has three types of layers: one input layer, multiple hidden layers, and one output layer. Figure 2 shows a typical neural network. Neurons in hidden layers include an activation function. Neurons in the output layer do not have an activation function or are thought to have a linear activation function.

2.1 Convolutional Neural Networks

Depending on the application, there are various types of neural networks with different architectures. One of the most common neural networks is Convolutional Neural Network (CNN). CNNs were inspired by the visual cortex organization of human. The first CNN is Time delay Neural Network (TNN) devised by Alexander Waibel in 1987 for speech recognition [95], then in 1998 Yann LeCun et al. presented LeNet-5 which was regarded as the turning point in the development of CNNs [96]. CNN gradually extracts high-level and more complex features by combining low-level and simple features [97]. This is similar to what is performed in the humans’ visual system. As CNNs were originally proposed for image processing, in their architecture the input is assumed to be an image [98].

Deep CNNs have superior accuracy, yet with high computational complexity. For this reason, Graphics Processing Unit (GPU) is applied for processing these networks. GPU advances have led to development of CNNs and the possibility of increasing the number of layers. Today, considerable efforts have been devoted to development of DCNNs and since 2012, all the winner models in the ImageNet challenge have been DCNNs [99].
There are different types of layers in a DCNN. Common layers include convolution layer, normalization layer, pooling layer, and FC layer. The convolution layer is the main layer in DCNN. Most parameters and computations are in FC and convolution layers. The main layer in DCNN is convolution which is formed in three dimensions. Due to its three-dimensional structure, most computations in DCNNs are in this layer. In the convolution layer, multiple filters (weights) are convolved with the input feature map and the result is the output feature map. Figure 3 shows the convolution of a filter with a patch of input feature map. The filter are showed with different colors and the number of them is equal to the output channel size.

The size of channel in the filters and input feature map is the same and the size of channel in the output feature map is equal to the number of filters. There is weight sharing in the convolution layer. Thus, each weight is applied to different connections in a single layer. This property significantly reduces parameters in the convolution layer. In the FC layer, a neuron is connected to all neurons in the previous and next layers. This property increases number of parameters in the FC layers. As most computations and parameters are in convolution and FC layers, the focus is on these layers in accelerators and compression.
3 BASIC CONCEPTS OF QUANTIZATION

Quantization is mapping from a continuous space to a discrete that full-precision values are transformed to new values with lower bit-width called quantization levels. To map full-precision values to quantization levels, a constant piecewise function is used as the following Equation (1).

\[ Q(x) = q_i \quad x \in [r_i, r_{i+1}), \quad i = 1, \ldots, m \]  

(1)

Quantization reduces the distortion by mapping full-precision values to the appropriate quantization levels. Therefore, computing the optimal quantization levels is essential for having accurate quantization. Figure 4 shows a quantization from continuous values in the range \([-\infty, +\infty]\) to a limited range and the values after quantization are \(Q = \{q_1, q_2, \ldots, q_m\}\). Step size specifies the distance between two successive quantization levels (Equation (2)). In the following, different types of quantization for neural networks are discussed from different perspectives.

\[ \text{Step size} = q_{i+1} - q_i \]  

(2)

Figure 4: Quantization map function.

3.1 Network Components and Quantization

Each numerical component in the neural networks can be quantized. These components are divided into three main categories: weights, activations, and gradients. Each parameter in the neural networks can be quantized and quantization of the weights is the most common. Bias and other parameters such as batch normalization parameters in most works are kept in full-precision in view of the fact that they include a very small rate of neural networks parameters and the quantization of them is less efficient in compression. Nevertheless, some of the works have quantized them such as paper [100] which represents biases in 1-bit precision.

Activations quantization pose more challenges in comparison with weight quantization. Weights are fixed after the training, whereas activations change at the inference time according to the input data. At first, most methods quantized only weights. While the activations remain in the full-precision, the efficiency of weight quantization is limited, especially in the CNNs, where the values of the weights in the convolution layers are relatively smaller compared to the FC layers [103]. Moreover, weight sharing in the convolutional layers reduces the number of parameters in these layers. The memory usage of activations is more than weights [104]. In recent years, most methods have quantized weights and activations. When weights and activations are stored in the low bit-width, the MAC operation is performed by low-cost operations. As a result, the computational cost and memory usage are reduced.

Gradients quantization is merely efficient for speed-up in the training phase. It is more difficult than weights and activations quantization. The gradients propagate from the output to the first layer of a network in the backward pass of
the EBP algorithm. High-precision gradient is essential for convergence of the optimization algorithm during training. Moreover, the values of gradients are in wide range. As a result, more bits are required for accurate quantization [101]. Almost all the works which quantized activations and gradients, necessarily quantized weights too. In an experiment, [102] quantized activations, while the weights were kept in the full-precision. Table 2 determines the components that were quantized in the previous works.

3.2 Time of Quantization

There are two types of Quantization: Quantization-Aware Training (QAT), and Post-Training Quantization (PTQ). The difference between QAT and PTQ is related to the time of quantization. QAT is performed during training, whereas the PTQ is performed after training.

3.2.1 Quantization-Aware Training

QAT is such a way that the quantization and training are performed simultaneously, and the network is trained with quantized values. In the QAT approach, the network is trained with quantized and discrete values. In a low-precision quantized network, the convergence of the learning algorithm is challenging. The training of a quantized network commonly requires more iterations than the full-precision network for convergence. It necessitates customized solutions compatible with a discrete network. QAT approach is employed for compression and speed-up in both the training and inference phases.

3.2.2 Post-Training Quantization

PTQ is performed after network training and building the model with full-precision values. Therefore, this approach is employed for compression and speed-up in the inference phase. PTQ for weight quantization reduces model accuracy, and retraining after quantization improves the accuracy. As a result, most methods have proposed fine-tuning after PTQ for compatibility of the trained model with the quantized values. After quantization, the network is retrained with the quantized weights once or repeatedly to reach an acceptable accuracy. Sung et al. [105] showed that there is a big gap between the model accuracy in PTQ quantization with and without retraining. Figure 5 demonstrates the overall steps in the PTQ approach.

![Figure 5: PTQ and retraining until acceptable accuracy.](image)

The model accuracy in the QAT approach is commonly higher than PTQ, where the quantized model is more compatible with the quantized values. Table 2 specifies the time of quantization in some previous works.

3.3 Deterministic and Stochastic Quantization

Full-precision values are mapped to quantization levels deterministically or stochastically. In deterministic quantization, a full-precision value is always mapped to a definite quantization level. In stochastic quantization, it may be mapped to
each of the quantization levels with a probability although it is always more likely to encode a specific quantization level with the highest probability.

Table 2: The quantized components in some methods: Weights (W), Activations (Act), and Gradients G) and their type of quantization: Uniform (U) or Non-uniform (NU), and Time of Quantization: QAT or PTQ

| Method                  | Components Quantization | Time of Quantization |
|-------------------------|-------------------------|----------------------|
| BinaryConnect [21]      | W (U)                   | QAT                  |
| TernaryConnect [101]    | W (U)                   | QAT                  |
| Sung et al. [105]       | W (U)                   | PTQ                  |
| DeepCompression [17]    | W (NU)                  | PTQ                  |
| Bitwise Neural Networks [100] | W (NU)   | Act (NU)    | QAT |
| TWN [106]               | W                       | QAT                  |
| BNN [42]                | W (U)                   | Act (U)              | QAT |
| XNOR-Net [22]           | W                       | Act                  | G   | QAT |
| Dorefa-net [23]         | W                       | Act                  | G   | QAT |
| TTQ [107]               | W                       | QAT                  |
| Miyashita et al. [102]  | W (NU)                  | Act (NU)             | G(UN)| QAT |
| QNN [77]                | W (U)                   | Act (U)              | G(U)| QAT |
| LogNet [108]            | W (NU)                  | Act (NU)             |     | PTQ |
| Tang et al. [109]       | W (U)                   | Act                  |     | QAT |
| HWGQ [110]              | W                       | Act                  |     | QAT |
| ABC [111]               | W                       | Act                  |     | QAT |
| INQ [112]               | W (NU)                  | Act                  |     | QAT |
| FGQ [113]               | W (U)                   | Act                  |     | QAT |
| WQ [79]                 | W (NU)                  | Act (NU)             |     | QAT |
| Balanced quantization [78] | W (NU)              | Act (NU)             | G(F/NU)| QAT |
| Zhuang et al. [24]      | W (NU)                  | Act (NU)             |     | QAT |
| PACT [103]              | W                       | Act                  |     | QAT |
| Xu et al. [114]         | W (NU)                  | Act                  |     | QAT |
| ELQ [115]               | W                       |                      |     | QAT |
| LQ-Nets [116]           | W (U)                   | Act (U)              |     | QAT |
| BitPruning [117]        | W (U)                   | Act (U)              |     | QAT |
| APoT [118]              | W (NU)                  | Act (NU)             |     | QAT |
| Bi-RealNet [25]         | W                       | Act                  |     | QAT |
| IR-Net [119]            | W (U)                   | Act (U)              |     | QAT |
| DJPQ [71]               | W (NU)                  | Act (U)              |     | QAT |
| DMBQ [120]              | W (U)                   | Act (U)              |     | QAT |
| SQ [121]                | W(U)                    |                      |     | QAT |

3.3.1 Deterministic Quantization Works

One of the deterministic functions used for binary quantization is the Sign function:

\[
b = \text{sign}(x) = \begin{cases} 
+1 & x \geq 0 \\
-1 & \text{otherwise}
\end{cases}
\]  

(3)

In Equation (3) the full-precision values (x) are mapped to the binary values: +1 and -1. Papers [22, 25, 42, 100, 109, 110] used Equation (3) for quantization. Deterministic ternary quantization is defined as
In Equation (4), $\Delta$ is a threshold. Papers [106, 113, 115, 122] used Equation (4) for quantization. Paper [105] proposed a deterministic quantization and used

$$Q(x) = \text{sign}(x) \cdot \min \left( \frac{\lceil x \rceil}{d}, \frac{M - 1}{2} \right)$$  \hspace{1cm} (5)$$

In Equation (5), $d$ is the step size, and $M$ is an odd number and determines the number of quantization levels. Consequently, the quantization levels include zero, positive and negative values symmetrically.

Papers [23, 24, 78] performed deterministic quantization with $k$ bit-width. First, they mapped the values to the range $[0,1]$, then used Equation (6) for quantization with $k$ bit-width.

$$\text{quantize}_k = \frac{1}{2^{k-1}} \text{round} \left( (2^k - 1)x \right), \quad 0 \leq x \leq 1 \hspace{1cm} (6)$$

Equation (6) maps the full-precision values in the range $x \in [0,1]$ to $2^k$ quantization levels in the same interval with step size $\frac{1}{2^{k-1}}$. For $k$ bit-width, the quantization levels are $L_q = \{0, \frac{1}{2^{k-1}}, \frac{2}{2^{k-1}}, \ldots, 1\}$. For example, for $k=3$, there are $2^3=8$ quantization levels. A quantization level is the result of an inner product between a basis vector and a $k$-bit binary vector:

$$Q(x, v) = v^T e_t \quad e_t \in \{ -1, 1 \}^k, \quad x \in (t_t, t_{t+1}) \hspace{1cm} (7)$$

In Equation (7), $x$ is a full-precision value, $v \in \mathbb{R}^K$ is the learnable floating-point basis vector, and $e_t$ is a $k$-bit binary vector from $[-1, -1, \ldots, -1]$ to $[1, 1, \ldots, 1]$.

### 3.3.2 Stochastic Quantization Works

Few previous research works used stochastic quantization [21, 77, 123]. The BinaryConnect method [21] used a stochastic binary quantization:

$$b = \begin{cases} +1 & x > \Delta \\ 0 & |x| \leq \Delta \\ -1 & x < -\Delta \end{cases}$$  \hspace{1cm} (4)$$

In Equation (8), $p$ and $q$ are the probabilities of 1 or -1. $\sigma$ is the hard sigmoid function as

$$\sigma(x) = \text{clip}(\frac{x + 1}{2}, 0, 1) = \max \left( 0, \min \left( 1, \frac{x + 1}{2} \right) \right) \hspace{1cm} (9)$$

Equation (9) assigns a probability to the values in the range (-1,1). For $x \in (0,1)$, the probability is bigger than 0.5 and smaller than 0.5 for $x \in (-1, 0)$. For $x$ equal to zero, the probability is equal to 0.5. Therefore, according to Equation (8), the full-precision values belonging to the interval (0,1) are mapped to +1 with a higher probability and for the values close to 1, this probability is higher. For the interval (-1,0) the higher probability is for mapping to -1 and the probability increases for the values close to -1.

Lin et al. changed Equation (8) for stochastic ternary quantization with [-1, 0, 1] quantization levels in the TernaryConnect method [101]. The values in the interval [-1,1] are mapped to a quantization level by the following probabilities:
\[
\begin{align*}
\text{if } x > 0: & \quad p(t = 1) = x; \quad p(t = 0) = 1 - x \\
\text{if } x < 0: & \quad p(t = -1) = -x; \quad p(t = 0) = 1 + x
\end{align*}
\]

In Equation (10), \( t \) and \( x \) are ternary and full-precision weight, respectively.

### 3.3.3 Deterministic and Stochastic Quantization Comparison

Stochastic quantization has better model generalization than deterministic quantization [123]. In stochastic quantization, the parameters are not mapped to a definite quantization level. Therefore, it is not possible to rely on any features and the weights are spread out like a regularizer. In fact, the stochastic quantization can act like a desired noise generator that is injected into the training process. Implementation of stochastic quantization is more difficult and costly than deterministic quantization. It requires a random bit generator in hardware implementation [42].

### 3.4 Quantization Levels Based on Distribution

An efficient quantization covers the distribution of full-precision values and preserves informative parts of the original data. In quantization, multiple values are limited to one value. To minimize quantization accuracy degradation, determining quantization levels is very important. In this section, the methods for determining quantization levels in previous works based on distribution of the full-precision values are discussed.

#### 3.4.1 Uniform and Non-uniform Quantization

By adjusting the step size, the distribution of quantization levels is changed and the quantization is divided into two general categories: uniform and non-uniform. In uniform quantization, the step size is constant, and in non-uniform quantization, it is variant. Figures 6a and 6b show uniform and non-uniform quantization, respectively. All quantization functions which are introduced in section 3.3 are uniform. Uniform quantization is implemented more simply than non-uniform quantization. In the non-uniform quantization, the step size is determined according to the data distribution. Hence, it is certainly more complex and accurate than the uniform quantization.

Logarithmic quantization is a type of non-uniform quantization with exponential quantization levels. Figure 6c represents a base-2 logarithm quantization and Equation (11) shows its relation.

\[
Q(x) = \text{Sign}(x)2^{\text{round}(|\log_2|x|)}
\]

In Equation (11), \( x \) is a full-precision number. Logarithmic quantization encodes a larger range of numbers in the same storage in comparison with the uniform quantization owing to the fact that the small integer exponent is saved instead of the floating-point number.

Previous studies revealed that weights in DCNNs have a commonly normal distribution with a mean equal to zero [79]. On the other hand, the quantization levels are denser for values close to zero in the logarithmic quantization. Therefore, the distribution of quantization levels is matched to the distribution of the full-precision weights. For instance, Figure 7 shows the distribution of the weights in convolution layers of the trained MobileNetV2 on ImageNet. As seen the weights have normal distribution in range [-0.4, 0.4] with mean close to zero.

The base-2 logarithm quantization is naturally a representation of the binary system. As a result, it is well-matched to the digital hardware and provides simple operations. Works [102, 108, 112, 117, 124] employed logarithmic quantization. Table 2 summarizes the previous methods in which the uniform or non-uniform quantization approaches were applied to weights, activations, and gradients separately.
3.4.2 Clustering-Based Quantization

Quantization is similar to clustering in that in the clustering, each value is assigned to a cluster, in quantization each full-precision value is mapped to one of the quantization levels. Many works have used clustering to quantize weights. In fact, clustering is a non-uniform quantization. In the clustering-based methods, the weights that are in the same cluster are reduced to one value. The number of quantization levels is equal to the number of clusters. After clustering, in the weight matrix, instead of saving the weight value (center of the cluster) which is a floating-point number, the cluster index is saved which is an integer number and uses less memory. Figure 8 shows an example of 2-bit clustering-based quantization. In this example, the weights are grouped into four clusters, quantized in 2 bits and the cluster numbers are saved in the codebook.
In the DeepCompression method, the weights are clustered by the k-means algorithm [17]. The weights in a cluster are close to each other and they are mapped to the same quantization level that is the center of the cluster.

Xu et al. proposed two clustering-based methods for high and low bit-width quantization. The first method is single level quantization (SLQ) for the weight quantization with high bit-width [114]. In this method, the weights of each layer are clustered separately using the k-means algorithm. After clustering, the clusters are grouped into two categories based on quantization loss. The clusters with small quantization loss (category 1) are quantized. For quantization, all the weights in a cluster are mapped to the center of the related cluster. The clusters in the second category are retrained. These steps are repeated until all the weights are quantized.

In SLQ, each layer is quantized separately. Xu et al. believed that the number of clusters in this method is small and the loss quantization is huge to be eliminated [114]. Therefore, this is not suitable for low-bit quantization and they suggested multiple level quantization (MLQ) for 2-bit and 3-bit quantization.

The SLQ method partitions the weights in a single layer, in other terms, in the width of the network whereas MLQ partitions the weights in the depth too. In the depth partitioning, the weights in each layer are clustered. Then the clusters called boundaries with more effect on the model accuracy are quantized. Unlike the SLQ method in which all boundaries in all layers are quantized at the same time, they are quantized iteratively and incrementally in MLQ. The idea behind this incremental approach comes from the different effect of layers on accuracy of a quantized model (this subject is discussed in section 6). The weights in each boundary are grouped, that is, clustering in width. The clusters in each boundary are quantized like SLQ by grouping the clusters into two categories. The iterations are continued until all clusters in depth are quantized. Figure 9 [114] shows an example of the MLQ method in which each layer is clustered into 3 clusters.

Park et al. adopted a clustering-based approach [79], thereby allocating more clusters to informative parts to preserve the information after quantization. They defined a weighted entropy measure for evaluating the quality of clustering. It determines the importance of full-precision values on the performance of the network. For N clusters, weighted entropy is defined as

$$ S = -\sum_n I_n p_n \log p_n \quad (12) $$

where

$$ p_n = \frac{|C_n|}{\sum_k |C_k|} \quad (13) $$

$$ I_n = \frac{\sum_m (n,m)}{|C_n|} \quad (14) $$

In Equations (13) and (14), C_n is the number of weights in the cluster n. Equation (13) indicates the density of cluster n. In Equation (14), I_{n,m} is defined as Equation (15), Where w_{n,m} indicates weight m in cluster n.
\[ i_{(n,m)} = w_{(n,m)}^2 \] (15)

Equation (15) determines the importance of the weights. It is motivated by the fact that the large weights have a more significant effect on the output. Equations (13) to (15) show that the clustering is performed based on the weights value and clusters density. The result of Equation (12) is maximized by finding the optimum weights clustering. To maximize the weighted entropy, different states of weights clustering were tested, which is very time-consuming.

Figure 9: The MLQ method.

**Challenges of clustering-based approach:** Clustering algorithms are effective for mapping optimal groups of weights to quantization levels. Nevertheless, it is not suitable for implementation in hardware and software. Time complexity and computations of these methods is considerable for reconstruction of the codebook [125]. Moreover, in a clustering-based approach, the weights in a cluster are not contiguous in memory, which leads to irregular memory accesses with long delays [113]. To solve this problem, Xu et al. proposed the Extended Single Level Quantization (ESLQ). It is an extended version of the SLQ method [114]. In the ESLQ, clusters centers as quantization levels change to a value with a specific type. For example, they are mapped to the closest number in the form of Power Of Two (POT) that is suitable for implementation on FPGA.

Clustering-based approach is performed for weight quantization. Weights are fixed after the network training and used in the inference phase whereas the distribution of activations changes in the inference time. Therefore, the clustering-based approach is not suitable for the activation quantization.

### 3.4.3 Scale Factor

Using a scale factor parameter is common for estimating the quantization levels accurately. The scale factor parameter is multiplied by the quantization levels. It is determined in such a way that the mathematical expectation of the full-precision values is close to the mathematical expectation of the related quantization levels. Therefore, quantization levels are commonly mapped to dense and informative places of the full-precision values using the scale factor. It is computed based on the mean of distribution of the full-precision values. The scale factor is especially effective in low bit-width quantization. The number of quantization levels is limited in low bit-width quantization and the scale factor helps to have a more accurate quantization.

Li and Liu in the Ternary Weight Network (TWN) method computed a scale factor and a threshold for each layer to approximate accurate ternary weights. The optimum scale factor and threshold are computed to minimize the difference between the quantized and the full-precision weights [106].

\[
\alpha^*, \Delta^* = \min_{\alpha > 0, \Delta > 0} ||W - \alpha W'||_2 \] (16)
In Equation (16) \( \alpha \) is the scale factor. \( W \) and \( W^t \) are the full-precision weights and ternary weights, respectively. \( \Delta \) is a threshold in the ternary quantization function:

\[
W^t_i = f_t(w_i, \Delta) = \begin{cases} +1, & w_i > \Delta \\ 0, & |w_i| \leq \Delta \\ -1, & w_i < -\Delta \end{cases} (17)
\]

Li and Liu proved the optimum \( \alpha \) and \( \Delta \) are computed by:

\[
\alpha^*_l = \frac{\sum_{j \in I_l} |w_j|}{|I_l|}, \quad \Delta^* = \max_{\Delta \geq 0} \left( \frac{\sum_{j \in I_l} |w_j|^2}{|I_l|} \right) (18)
\]

In Equation (18), \( I_l=\{1 \leq i \leq |w|:|w_i|>\Delta \} \) and \(|I_l|\) is the number of elements in \( I_l \). Since the value of \( \Delta^* \) is almost the same for different distributions, they computed \( \Delta^* \) for all distributions as

\[
\Delta^* \approx 0.7 \frac{\sum_{i} |w_i|}{n} (19)
\]

In XNOR-Net method, the scale factor is used for binarization of weights, activation, and gradients [22]. The scale factors for weights and activations are computed by the mean of values in each filter separately. For gradients, the scale factor is computed as

\[
g^* = \max \left( |g^{in}| \right) (20)
\]

\( g^a \) is gradients from the next layer in the backward pass. The range of gradients is wide and Equation (20) preserves maximum variations in all dimensions [22].

In the Dorefa-net method, the scale factor is computed for binary weights the same as the XNOR-Net method, but layer-wise instead of filter-wise [23]. Lin et al. proposed a linear combination of several binary weights for quantization in the Accurate Binary Convolutional (ABC) method [111]:

\[
W \approx \alpha_1 B_1 + \cdots + \alpha_M B_M, \quad B_i \in \{-1,1\}^{w \times h \times c_{in} \times c_{out}} (21)
\]

In Equation (21), \( W \) is the weight matrix of a layer with size of \( w \times h \times c_{in} \times c_{out} \). \( w \) and \( h \) are the width and length of the filters. \( c_{in} \) and \( c_{out} \) are the numbers of input and output channels. \( B \) and \( \alpha \) are the binary weights matrix and scale factor, respectively. To find the optimum scale factors in Equation (21), the solution in the TWN method [106] is applied. For activation quantization, a linear combination the same as Equation (21) is proposed with this difference that the scale factor in activations quantization is trainable.

**Trained scale factor:** In the Trained Ternary Quantization (TTQ) method scale factor is computed for the positive and negative ternary weights in each layer for covering weights distribution efficiently [107].

\[
w^p_i = \begin{cases} W^p_i \times +1 & w_i > \Delta \_l \\ 0 & |w_i| \leq \Delta \_l \\ W^p_i \times -1 & w_i < -\Delta \_l \end{cases} (22)
\]

In Equation (22), \( W_i \) and \( \Delta \_l \) are the full-precision weights and the threshold, respectively. \( W^p_i \) and \( W^n_i \) are the scale factors for the positive and negative weights, respectively. The value of the scale factor is updated during the training algorithm to achieve the optimum value. In the backward pass of the EBP algorithm, in addition to the gradient for updating the network weights, another gradient is computed for updating the scale factors \( W^p_i \) and \( W^n_i \). Equations (23) and (24) compute the first and second gradients, respectively.
Equation (23) shows the scale factors effects updating the weights. Furthermore, it is concluded from Equation (24) that the scale factors are updated with respect to the changes of the weights. Two heuristic methods are proposed for determining the layer-wise threshold. The first one is used in smaller datasets. In this method, threshold is computed according to the maximum absolute value of the weights as

$$\Delta_i = t \times \max(|\hat{w}|)$$

In Equation (25) finding the optimal $t$ is a problem. The second one that is used for large-scale datasets such as ImageNet determines the threshold based on the rate of sparsity in ternary quantization [126].

**Increasing quantization representational capability:** Mellempudi et al., for ternary weight quantization, divided the weights into $k$ subsets and used a separate scale factor for each subset [113]. Furthermore, they employed two distinct thresholds for the positive and negative weights in each subset. The idea of different thresholds comes from the different distributions of the positive and negative weights that are not always symmetric to zero. Thus, they used two different thresholds $\Delta_p$ and $\Delta_n$ in Equation (17). As a result, they increased the quantization representational capability from 3 levels $\{-\alpha, 0, +\alpha\}$ to $2k+1$ levels. The optimum scale factor and threshold are calculated the same as Equation (18) for each subset separately.

**Maximum weights in computing the scale factor:** In Explicit Loss-error-aware Quantization (ELQ) method [115], Equation (26) was proposed for computing the scale factor in binary and ternary quantization.

$$\alpha_t = \text{mean}(W) + \beta \max(W)$$

In Equation (26), $\beta$ is a positive coefficient which is considered 0.05. The scale factor is computed layer-wise. $\beta$ controls the effect of maximum weights. To compute the threshold in ternary weights, the scale factor is used as

$$t_i = \begin{cases} \alpha_t & W_i > 0.5\alpha_t \\ -\alpha_t & W_i < -0.5\alpha_t \\ 0 & a.w. \end{cases}$$

**The effect of scale factor in quantized network convergence:** Liu et al. in the Bi-RealNet method showed the scale factor efficiency in convergence of a quantized network [25]. They binarized weights by the $\text{Sign}$ function. Figure 10a [25] shows the full-precision and the binary weights distributions without using the scale factor in the ResNet. As it shows that the magnitude of the full-precision weights is equal to 0.1 whereas it is 1 for the binarized weights. Consequently, the full-precision weights have a big gap with the binarized weights that diverges quantized network training. Liu et al. proved that when there is a BatchNorm layer after the convolutional layer in the network, if all elements in the kernels multiply by $m$, the gradient decreases by rate $1/m$. Therefore, when $m$ is very large, the gradient is too far from its true value in the backward pass that leads to the training divergence. In Figure 10a [25], the magnitude of the weights increases by rate 10 after quantization. The scale factor is used to decrease the distance between full-precision and quantized weights. In Bi-RealNet,
the scale factor is set equal to the magnitude of full-precision weights. Figure 10b [25] shows after using the scale factor, the full-precision weights distribution is close to the binarized weights distribution.

In the Bi-RealNet method, the scale factor is computed only in the network training and it is not computed in the inference phase. Liu et al. proved that the BatchNorm output is independent of the re-scaling of the weights in the inference phase.

**Challenge of using scale factor:** Using the scale factor has two problems. First, it requires complex computations. To solve this problem, Tang et al. did not use the scale factor in the quantization function and moved it to the activation function [109] and used **Parametric ReLU** [127]:

\[
X_{i} = \begin{cases} 
    x_i & x_i > 0 \\
    \alpha_i x_i & x_i \leq 0
\end{cases}
\]

In Equation (28), \(i\) specifies a channel, thus for each channel different \(\alpha\) is computed. \(\alpha\) is updated during training to the optimum value. The second problem arises from the floating-point value of the scale factor. Using floating-point operations in the convolution leads to large number of cycles on hardware and high computation cost. For solving this problem, some works proposed using POT numbers as the scale factor. Consequently, the floating-point multiply is replaced with low-cost bit-shift operator [119].

4 **TRAINING OF QUANTIZED NEURAL NETWORK**

The main algorithm for training the neural networks is EBP [128]. EBP uses the gradient descent algorithm and the chain rule of derivatives for tuning parameters. Suppose that Figure 11 shows an extract of a neural network. In Figure 11, \(j\) determines a hidden layer and the weights between layers \(k\) and \(j\) are updated as

\[
w_{kj}(n) = w_{kj}(n - 1) - \gamma \Delta w_{kj}, \Delta w_{kj} = \frac{\partial E}{\partial w_{kj}} = \sum_{i \in \mathcal{I}} (\delta_{i}w_{ij})h'(x_i)y_k = \delta_{j}y_k
\]

In Equation (29), \(w_{kj}(n-1)\) and \(w_{kj}(n)\) are the weights between \(k\) and \(j\) layers before and after updating, respectively. \(\gamma\) and \(\delta\) are the learning rate and the error signal, respectively. \(x_i\) and \(y_i\) are the inputs and outputs of layer \(i\), respectively, and \(h'\) is derivative of the activation function. As this equation shows for computing the error signal of a layer, the gradient
of activation function is multiplied by other terms. As a result, zero and undefined gradient of activation function make the training of the neural networks problematic and the weights are not updated. In quantization, a constant piecewise and discrete function is commonly used. Therefore, there are some places with undefined and zero derivatives. The problem is more in activation quantization, where the activation function is non-differentiable. To solve this problem, many works used the STE approach to estimate the gradient for the non-differentiable function.

4.1 Straight-Through Estimator

The STE idea dates back to the 1950s when it was used for the perceptron learning algorithm [129]. Unlike the EBP algorithm, in the perceptron algorithm, it is not possible to propagate the gradients from the last layer to the first layer. STE was raised in 2012 by Hinton again [130]. He developed this method for training the binary network with multiple layers, where the activation function is non-differentiable. He assigned the gradient equal to 1 to the positive arguments and zero otherwise, in the backward pass of EBP algorithm. In 2013, Bengio et al. studied the problem of estimating or propagating gradients through the stochastic discrete neurons [131]. They found in their experiments that the fastest training is performed by using the STE. They suggested using the derivative of the Sigmoid function as STE in the backward pass. Then STE has been widely used in QAT.

In quantization, STE is used for estimating the gradient of a non-differentiable quantized function in the forward pass by a differentiable function in the backward pass. It is essential that the estimated function in the backward pass be close to the activation function in the forward pass for convergence during training. Therefore, STE is an optimization problem as

\[
\min_{Q(x)} E = |Q(x) - \tilde{Q}(x)|
\]

In Equation (30), \( Q(x) \) and \( \tilde{Q}(x) \) are the forward activation function and its estimation in the backward pass, respectively.

**STE for the Sign function:** In the Bitwise Neural Networks method [100], the training was performed with binary weights and activations. The activations were quantized by the Sign function as Equation (3). The derivative of the Sign function is the Impulse function. Impulse function is non-differentiable (Figure 12a [25]). Therefore, in the backward pass, the STE is used and 1 is assigned to the gradient of the Sign function.

Liu et al. studied three differentiable functions for STE of the Sign function [25]. Figures 12b, 15c, and 15d [25] show the estimated functions and their derivative. The first function (Figure 12b [25]) is hard tanh (Clip). The most previous works such as the Binarized Neural Networks (BNN) [42] and XNOR-Net [22] methods employed it as STE of Sign function. The gradient of hard tanh is:

\[
h'(x) =\begin{cases} 
1 & -1 \leq |x| \leq 1 \\
0 & x < -1 \text{ or } x > 1 
\end{cases}
\]

(31)
Figure 12: a) The Sign function and its derivative. b) The hard tanh function and its derivative. c) A second-order estimator function for the Sign function and its derivative. d) A third-order estimator function for the Sign function and its derivative.

In the hard tanh, the values in the range [-1,1] have the gradient of 1 and the values outside this interval have the gradient of zero. Liu et al. introduced a new function similar to hard tanh for STE of Sign function as Figure 12c [25]. Equations (32) shows this function and Equation (33) represents its gradient.

\[
F(a_r) = \begin{cases} 
-1 & a_r < -1 \\
2a_r + a_r^2 & -1 \leq a_r < 0 \\
2a_r - a_r^2 & 0 \leq a_r < 1 \\
1 & a_r \geq 1 
\end{cases} \tag{32}
\]

\[
\frac{\partial F(a_r)}{\partial a_r} = \begin{cases} 
2 + 2a_r & -1 \leq a_r < 0 \\
2 - 2a_r & 0 \leq a_r < 1 \\
0 & \text{otherwise} 
\end{cases} \tag{33}
\]

The shaded areas in Figure 12 [25] show the difference between the Sign function and the estimated function. The shaded area in hard tanh is equal to 1 and in the proposed function is equal to 2/3. It means that the function is a better estimator for the Sign function. As the order of the function in Equation (32) increases, the difference between the estimator and the Sign function decreases. Figure 12d [25] shows the estimated function with third-order, and Equation (34) determines its relation.

\[
(a_r) = \begin{cases} 
-1 & a_r < -1 \\
(a_r + 1)^3 - 1 & -1 \leq a_r < 0 \\
(a_r - 1)^3 + 1 & 0 \leq a_r < 1 \\
1 & a_r \geq 1 
\end{cases} \tag{34}
\]

Liu et al. concluded that higher-order function requires more complex computations. Therefore, they believed that the second-order function is acceptable.

Error Decay Estimator: The Information Retention Network (IR-Net) method suggested Error Decay Estimator (EDE) [119] for estimating the Sign function in the backward pass as

\[
F(x) = k \tanh (tx) \tag{35}
\]
where $k$ and $t$ are computed as

$$\begin{align*}
    t &= \frac{\log_{10} T_{\text{max}}}{\log_{10} T_{\text{min}}}, \quad k = \max(1/t, 1) \tag{36}
\end{align*}$$

In Equation (36), $i$ determines epoch number from $N$ epochs, $T_{\text{min}}=10^{-4}$, and $T_{\text{max}}=10$. EDE is between the identity ($y=x$) and hard tanh functions. hard tanh is close to the Sign function. It ignores the parameters outside [-1,1] and they are not updated anymore. As a result, some information is lost. On the other hand, although the identity function covers the parameters outside [-1,1], it has a huge difference with the Sign function as is shown in Figure 13 by the shaded area. EDE makes a trade-off between the identity and hard tanh functions using $k$ and $t$ parameters. $k$ and $t$ are varying during training. First, $k$ is bigger than 1 and EDE is close to the identity function. While the number of epochs increases, $k$ gradually tends to 1. Therefore, EDE changes to hard tanh for having a more accurate estimation.

**Figure 13**: Distance between the Sign and identity functions.

### Quantized ReLU and STE

Cai et al. proposed Half-Wave Gaussian Quantization (HWGQ) for activations quantization [110]. HWGQ is a half-wave rectifier like ReLU known as quantized ReLU [132].

$$Q(x) = \begin{cases} 
    q_i & x \in (t_i, t_{i+1}] \\
    0 & x \leq 0 
\end{cases} \quad i = 1, \ldots, m, \quad q_i, t_i \in \mathbb{R}^+ \quad t_0 = 0, t_{m+1} = \infty \tag{37}$$

In Equation (37), $q_i$ shows the $i$th quantization level. Figure 14a [110] shows the HWGQ function with quantization levels $\{0, q_1, q_2\}$. Finding the optimal values of $q_n$ and $t$s is the main challenge in the HWGQ function and depends on the distribution of full-precision values. $q_n$ is chosen in such a way that the values greater than it are negligible. The derivative of HWGQ is zero. Cai et al. examined three functions as an estimator in the backward pass: 1) Vanilla ReLU 2) Clipped ReLU 3) Log-Tailed ReLU. In the following, these functions are discussed. The ReLU function is an estimator of HWGQ in the backward pass that is denoted as Vanilla ReLU. The derivative of Vanilla ReLU is:

$$Q' = \begin{cases} 
    1 & x > 0 \\
    0 & \text{otherwise} \end{cases} \tag{38}$$

HWGQ is bounded to $q_n$ for $x>0$, whereas Vanilla ReLU goes to infinity. Therefore, there is a mismatch between two functions in the interval $(t_n, \infty)$. Consequently, using ReLU in the backward pass leads to inaccurate gradients and unstable...
learning. The influence of HWGQ and vanilla ReLU mismatch is more significant in deeper networks. The second function is Clipped ReLU for estimating HWGQ in the backward pass (Figure 14c [110]). Clipped ReLU and its derivative are:

\[
\tilde{Q} = \begin{cases} 
q_m & x > q_m, \\
0 & 0 < x \leq q_m, \\
otherwise & 
\end{cases}, \\
\tilde{Q}' = \begin{cases} 
1 & 0 < x \leq q_m, \\
0 & otherwise 
\end{cases}
\] (39)

In Clipped ReLU, the weak point of the Vanilla ReLU is modified, where the gradient of \(x \geq q_m\) is zero. Consequently, it is a suitable estimator for HWGQ. The idea behind this behavior comes from the fact that the frequency of the large values is commonly low and these values are interpreted as outliers. The third estimator function for HWGQ is Log-Tailed ReLU.

Log-Tailed ReLU is between Vanilla ReLU and Clipped ReLU (Figure 14d [110]). The Log-Tailed ReLU function and its derivative are in Equations (40) and (41), respectively.

\[
\tilde{Q} = \begin{cases} 
\frac{q_m + \log(x - \tau)}{x} & x > q_m, \\
0 & 0 < x \leq q_m, \\
\tau = q_{m-1} & x \leq 0 
\end{cases}
\] (40)
Log-Tailed ReLU does not ignore the values $x > q_m$ completely. The gradient of values $x > q_m$ goes to zero gradually while $x$ tends to infinity. The experiments show that the Log-Tailed ReLU reaches higher accuracy compared to Clipped ReLU. However, the results of Clipped ReLU are better than Log-Tailed ReLU in VGGNet and ResNet which are deeper than Alexnet [132].

Bounded rectifier and STE: In the ABC method [111] a bounded rectifier activation function was proposed for mapping the full-precision activations to the range $[0,1]$ as

$$h_v(x) = \text{clip}(x + v, 0, 1) = \begin{cases} 1 & x + v > 1 \\ x + v < 1 \\ 0 & x + v < 0 \end{cases} \quad (42)$$

In Equation (42), $v$ is a trainable shift parameter. After mapping the values to the range $[0,1]$, Equation (43) is used for binarization as

$$A = H_v(R) = 2I_{h_v(R) > 0.5} - 1 = \begin{cases} +1 & h_v \geq 0.5 \\ -1 & h_v < 0.5 \end{cases} \quad (43)$$

In Equation (43), $I$ is the Indicator function. In the backward pass, STE is used and the gradient is computed as

$$\frac{\partial c}{\partial R} = \frac{\partial c}{\partial A} \log A - v = 1 \quad (44)$$

Parametric quantization function and STE: Choi et al. [103] introduced a PArametric Clipped Activation (PACT) function as

$$y = PACT(x) = 0.5(|x| - |x - \alpha| + \alpha) = \begin{cases} 0, & x \in (-\infty, 0) \\ x, & x \in [0, \alpha] \\ \alpha, & x \in [\alpha, +\infty) \end{cases} \quad (45)$$

The PACT maps the full-precision activations to the range $[0, \alpha]$. Then the outputs of the PACT function are quantized to $k$ bit-width by:

$$y_q = \text{round}(y \frac{2^{k-1}}{\alpha}) \quad (46)$$

If $\alpha$ is equal to 1, the PACT function is the bounded rectifier function with $v=0$ in the ABC method [111]. The optimum $\alpha$ is found during training for minimizing the accuracy drop in quantization. The optimum value is different from one layer to another layer and from one model to another model. Equation (46) is not differentiable hence, STE is used for updating $\alpha$ as

$$\frac{\partial y_q}{\partial \alpha} = \frac{\partial y_q}{\partial y} \frac{\partial y}{\partial \alpha} = \begin{cases} 0 & x \in (-\infty, \alpha) \\ 1 & x \in [\alpha, +\infty) \end{cases} \quad (47)$$

The training is depended on value of $\alpha$. If the initial value of $\alpha$ is too small, due to Equation (47), the most activations are in the range with non-zero gradient. In this regard, $\alpha$ frequently changes during training, leading to low model accuracy. On the other hand, if the initial value of $\alpha$ is too large, the gradient is zero for the most activations. As a result, the gradients
become too small that leads to the gradient vanishing in EBP. Although $\alpha$ is not very large, it is initialized with a large value, and reduced in L2-norm regularization.

The results of previous works show that STE is effective in practice. However, a concern is existed that the efficiency of STE has been not proved theoretically. Therefore, in recent years, some researchers have tried to justify the performance of STE theoretically [133, 134]. Table 3 determines the forward quantization function and its estimator in the backward pass as STE for some previous works.

| Method              | Forward activation function | Estimator in backward |
|---------------------|-----------------------------|-----------------------|
| Bitwise Neural Networks [100] | $q = \text{Sigm}(r)$ | $g_r = g_q$ |
| BNN [42]            | $q = \text{Sigm}(r)$ | $g_r = g_q^{t \mid t+1}$ |
| XNOR-Net [22]       | $q = \text{Sigm}(r)$ | $g_r = g_q^{t \mid t+1}$ |
| Dorefa-net [23]     | $q = \text{Sigm}(r)$ | $g_r = g_q$ |
| QNN [77]            | $q = \text{Sigm}(r)$ | $g_r = g_q^{t \mid t+1}$ |
| HWGQ [110]          | $q = \{q_i, \ r \in (t_r, t_{r+1})\}$ | $g_r = g_q^{t \mid t+1}$ |
| ABC [111]           | $q = 2t_{\text{int}} - 1, \ x \in [0,1]$ | $g_r = g_q^{t \mid t+1}, \ r \in \mathbb{R}$ |
| Balanced quantization [78] | $q = \frac{2^x - 1}{x} \cdot \ x \in [0,1]$ | $g_r = g_q$ |
| PACT [103]          | $q = \frac{2^x - 1}{x} \cdot \ x \in [0, a]$ | $g_r = g_q^{t \mid t+1}, \ r \in \mathbb{R}$ |
| Bi-RealNet [25]     | $q = \text{Sigm}(r)$ | $g_r = g_q^{t \mid t+1}, \ r \in \mathbb{R}$ |
| IR-Net [119]        | $q = \text{Sigm}(r)$ | $g_r = g_q^{t \mid t+1}, \ r \in \mathbb{R}$ |

4.2 Weights Update in Quantized Network Training

After the weight quantization, the weights are limited to quantization levels. There is a distance between the quantization levels equal to the step size. In the update step of the EBP algorithm, the change of weights is smaller than the quantization step sizes. Therefore, if the quantized weights are used in the update relation in the training phase, they are not updated as Figure 15a shows. In Figure 15, suppose that $L_1, L_2,$ and $L_3$ are the quantization levels. $W_f^t$ and $W_f^{t+1}$ are the full-precision weights before and after update, respectively, and $W^t$ and $W^{t+1}$ are their corresponding values after the quantization. The relations related to Figure 15a are as

$$W^t = \text{quantize}(W_f^t) = L_1$$

$$W_f^{t+1} = W^t - \gamma \frac{\partial L}{\partial W^t}$$

$$W^{t+1} = \text{quantize}(W_f^{t+1}) = L_1 \Rightarrow W^t = W^{t+1} \Rightarrow \text{no - update} \quad (48)$$

In Figure 15a and relations in Equation (48), the initial value of quantized weight is equal to $L_1$. After weight update, the change is very small and the updated value is quantized to $L_1$ again, thus the weight is not updated. To solve this problem, most works keep the full-precision weights after quantization, then use them in the update step. Figure 15b and relations in Equation (49) show the update step by full-precision weights. The quantized weight is used in the backward step and
computing the gradient. Then the full-precision weight is used in the update step. Finally, the updated weight is quantized again. As seen, the weight value is updated from L1 to L2.

\[ W^t = \text{quantize}(W_f^t) = L_1 \]

\[ W_f^{t+1} = W_f^t - \gamma \frac{\partial L}{\partial W_f^t} \]

\[ W^{t+1} = \text{quantize}(W_f^{t+1}) = L_2 \Rightarrow W^t \neq W^{t+1} \Rightarrow \text{update} \quad (49) \]

---

4.3 Some Effective Parameters in Training

Due to the discretization in quantization, training in a quantized network is challenging. It requires new methods to increase the model accuracy. In a discrete space, the parameter space becomes much smaller and convergence in the training algorithm is a problem. The problem is more in lower bit-width quantization such as binary and ternary quantization. For this reason, new techniques are required for training the quantized neural networks. In this section, the effect of the learning rate, network structure, and regularization in the training of a quantized neural network is discussed.

4.3.1 Learning Rate

The effect of the initial value of the learning rate in neural network training is considerable. In quantized networks, a small learning rate often works better. Using a large learning rate leads to jumping weight values frequently among limited quantization levels. As a result, the training is not stable and does not converge. Work [109] discussed the effect of the learning rate in the training of a binarized neural network.

4.3.2 Network Structure

Sometimes structural changes are necessary for the neural network after quantization. For instance, the max-pooling layer in some binary DNNs was displaced. In DCNNs, a max pooling layer commonly comes immediately after the activation layer. On the other hand, in a binary neural network, the \textit{Sign} function is used for the binarization of the activation function output. After the binarization of a DCNN, using the max-pooling immediately after the \textit{Sign} function leads to an output...
matrix with only +1 elements as the values in a binarized matrix are -1 and +1. Consequently, the information will be lost. To solve this problem, [22, 111] placed the max-pooling layer after the convolution layer.

4.3.3 Regularization

In DNNs, regularization is necessary due to the large volume of parameters and the high possibility of overfitting. L1 and L2 regularization are commonly used in the full-precision networks. In L1 and L2 regularization, network parameters are pushed into near zero for reducing their effect. As quantization is a process in which weights are approximated by small values, it can act as a regularizer. Using L1 or L2 regularization for decreasing the effect of weights leads to more accuracy drop in quantization. To solve this problem, some previous works proposed a new regularization relation compatible with their quantization method [109, 135-138].

Bit-level Sparsity Quantization (BSQ) [135] suggested a regularization relation for mixed-precision quantization. Work [136] offered a periodic regularization to push the weights and activations to the quantization levels for increasing the quantization accuracy.

Tang et al. introduced a new regularization for binary quantization [109]. They pushed the weights to near quantization levels [-1,1] for decreasing the quantization error by:

$$J(W, b) = L(W, b) + \lambda \sum_{l=1}^{L} \sum_{i=1}^{N_l} \sum_{j=1}^{M_l} (1 - (W_{lij}))^2$$  \hfill (50)

In Equation (50), \(L(W, b)\) is the loss function and the second term is regularization relation. \(L\) determines the number of layers. \(N_l\) and \(M_l\) are the dimensions of the weight matrix in layer \(l\). \(\lambda\) controls the effect of the loss function and regularization term. ProxQuant [137] proposed a regularization for binary weights to push the full-precision weights to the binary quantization levels.

The Smart Quantization (SQ) method [121] quantized in 1-bit and 2-bit precision and presented an adaptive binary and ternary regularization. For binary quantization, the quantization levels are \([-\alpha, +\alpha]\) and for ternary quantization, the quantization levels are \([-\alpha, 0, +\alpha]\), where \(\alpha\) is the scale factor. The adaptive binary and ternary regularization is defined as

$$R(W, \alpha, \beta) = \min \left( |w| + \mu \, p, \, |w| - \mu \, p, \, \tan(\beta)|w|^p \right)$$  \hfill (51)

In Equation (51), \(w\) is the full-precision weight. \(p\) is the order of the regularization function and is selected as 1. \(\beta \in \left(\frac{\pi}{4}, \frac{\pi}{2}\right)\) controls the transition between the binary and ternary quantization. When \(\beta\) goes to \(\frac{\pi}{2}\), \(\tan(\beta)\) value is large and away from zero. Thus, Equation (51) changes to binary regularization. When \(\beta\) goes to \(\frac{\pi}{4}\), it converges to ternary quantization. By the small value for \(\tan(\beta)\), Equation (51) converges to the zero level.

Work [138] defined the quantization regularization as

$$R = \sum_{n=1}^{N} \sum_{l=1}^{\text{card}(W_n)} \frac{|w_{n_l} - w_{q_{n_l}}|}{\max(q_{n_l} \times \text{card}(W_n))}$$  \hfill (52)

where \(N, W, \) and \(W_q\) are the number of layers, full-precision weight, and quantization weight, respectively. \(\text{card}(W_n)\) determines the number of weights in layer \(n\). This regularization is the mean of the absolute difference between the full-precision weight and its corresponding quantization level. Equation (52) was extended to Equation (53) for POT quantization as

$$R = \sum_{n=1}^{N} \sum_{l=1}^{\text{card}(W_n)} \frac{|w_{n_l} - w_{q_{n_l}}| |w_{n_l}|}{\max(q_{n_l} \times \text{card}(W_n))}$$  \hfill (53)
In POT quantization, the density of the quantization levels decreases in large values. In that respect, Equation (53) makes regularization exert more effects on the larger weights. In fact, in all quantization regularization methods, the full-precision weights are pushed to the quantization levels gradually during training for alleviating accuracy drop stemming from quantization.

5 OPERATIONS IN QUANTIZATION

The main operation in DCNNs is MAC operation which is in convolution and FC layers. Figure 16 represents the MAC operation in convolution layer. MAC is a dot product between weights and inputs as

\[ y = WX \quad W \in \mathbb{R}^n, X \in \mathbb{R}^n \]  

(54)

In Equation (54), \( W \) and \( X \) are the weight and input matrices, respectively. There are \( n \) multiplies and adds floating-point operations in Equation (54). After quantization, 32-bit floating-point numbers are mapped to low bit-width values such as 8, 4, 2, or 1 bit. As a result, the floating-point operations are replaced with integer or bitwise ones. In the following, the proposed methods are discussed for speeding up the MAC operation in DCNN quantization.

Figure 16: Dot product of a filter and a patch of input.

Miyashita et al. [102] developed two methods based on logarithmic quantization. In the first method, only activations were quantized in base-2 logarithm, hence the MAC was performed as

\[ y = WX \approx \sum_{i=1}^{n} w_i \times 2^{\bar{x}_i} = \sum_{i=1}^{n} \text{Bitshift}(w_i, \bar{x}_i) \]  

(55)

In Equation (55), \( w_i \) is a fixed-point number and \( \bar{x}_i \) is an integer number which is computed as

\[ \bar{x}_i = \text{Quantize}(\log_2(x_i)) \]  

(56)

In Equation (55), the expensive floating-point multiply is replaced by the simple bit-shift operation and \( w_i \) shifts by \( \bar{x}_i \) in fixed-point arithmetic. In the second method, Miyashita et al. proposed the base-2 logarithm quantization of the weights and activations, and the convolution operation is changed to Equation (57). Therefore, the floating-point multiplications are replaced by bit-shift and integer add.

\[ y = WX \approx \sum_{i=1}^{n} 2^{\bar{w}_i} \times 2^{\bar{x}_i} = \sum_{i=1}^{n} 2^{\bar{w}_i + \bar{x}_i} = \sum_{i=1}^{n} \text{Bitshift}(1, \bar{w}_i + \bar{x}_i) \]  

(57)

Works [117, 139] quantized the weights and activations in POT quantization. Both of them used shift and integer add for MAC operation as Equation (57). When the weights and activations are quantized to fixed-point integer, the bitwise operations can be applied for MAC. Suppose that weights \( W \) are in M-bit fixed-point integer and activations \( X \) are quantized in N-bit fixed-point integer as

\[ W = \sum_{m=0}^{M-1} w_m 2^m \quad X = \sum_{n=0}^{N-1} x_n 2^n, \forall m, n, w_m, x_n \in \{0, 1\} \]  

(58)
MAC operation between $W$ and $X$ is computed as

$$y = WX = \sum_{m=0}^{M-1} \sum_{u=0}^{N-1} 2^{m+k} \text{bitcount} \left[ \text{and} \left( w_m, x_u \right) \right]$$  \hspace{1cm} (59)

The Dorefa-net [23], and Balanced quantization [78] methods used Equation (59). In a binary neural network, both weights and activations are quantized to 1-bit, and MAC operations can be performed by bitwise operations as

$$y = WX = \text{bitcount} \left( \text{xnor}(W, X) \right)$$  \hspace{1cm} (60)

In Equation (60), $W$ and $X$ are binary weights and activations matrices of $[-1,1]$, respectively. $\text{xnor}$ is the XNOR operation and is used instead of the floating-point multiplication. The $\text{bitcount}$ function counts the number of 1 in a bit string. It is applied instead of the floating-point add. If in binary quantization $[0,1]$ are used instead of $[-1,1]$, the XNOR operation will be replaced by the AND operation. The Bitwise Neural Networks [100], BNN [42], XNOR-Net methods, and [109] employed Equation (60) for MAC.

Zhang et al. [116] quantized the weights and activations by Equation (7) in $K_w$-bit and $K_a$-bit, respectively. They used bitwise operations for computing the MAC between filters and activations as

$$y = WX = Q(W, v^w)Q(X, v^a) = \sum_{i=1}^{K_w} \sum_{j=1}^{K_a} v_i^w v_j^a \text{bitcount} \left( \text{xnor}(W, X) \right)$$  \hspace{1cm} (61)

In Equation (61), $v^w \in \mathbb{R}^{K_w}$, $v^a \in \mathbb{R}^{K_a}$ are the learnable floating-point basis vectors for weights and activations, respectively like Equation (7).

In the quantization methods in which the zero level is defined, the vast majority of the full-precision numbers are mapped to zero. When one of the operands (weight or activation) is zero, the multiplication operation is omitted. Therefore, the computations decrease more. Moreover, the activations smaller than zero are mapped to zero using the ReLU function. For instance, Venkatesh et al. showed that only 16% of operation in the forward pass and 33% of operation in the backward pass are non-zero for 2-bit weight quantization in the training of Resnet-34 on ImageNet [122].

6 LAYERS IN QUANTIZATION

Quantization of different type of Layers in DCNN has a different effect on prediction accuracy. The quantization of convolution and FC layers produces different effects on DCNNs. In the FC layers, each weight is connected to just one input, whereas in the convolution layers, weights are shared in such a way that each weight is connected to different patches of the input feature map. Therefore, it seems that the quantization of the weights in the convolution layers has a greater effect on the network accuracy drop compared to the FC layers. Moreover, the place of layers in DCNN influences on the efficiency of quantization. The size of the input channel and output channel in the last convolution layers in the DCNNs are bigger than the first convolution layers generally. As a result, most parameters with a wide range of values are in the last layers, and quantization of them leads to more efficiency in memory saving and possessing speed-up.

6.1 First and Last Layers

Among all the layers of a DCNN, the first and last layers cause more accuracy drop in quantization. Input and output layers are related to the input and output of the network, respectively. The first convolutional layer is more sensitive compared with other layers. The input of the first layer is the main data. Filters in the first layer must extract the most important and informative features. In DCNNs, the features in a layer are dependent on the combination of the extracted features from the previous layers. If the first layer does not extract informative features, this will continue layer by layer and lead to accuracy drop. The last layer is related to the output, where the output of it is compared with the desired result, whereas
quantization makes a significant variation in the outputs [109]. On the other hand, the size of input channel and output channel in the first layer are commonly smaller than the other layers. Thus, a small number of weights and MACs are in the first layer compared with other layers. Quantization of first layer has small effect on compression rate. Considering the small number of operations in the last layer, which is a FC layer in most DCNNs, its quantization has small effect on processing speed-up. BRECQ [160] compared the effect of the first and last layers quantization in terms of memory, and latency. Their study shows that quantization of the last fully-connected layer is more efficient than first layer in memory saving as weights are more. In terms of latency, it depends on the architecture of DCNN. For some DCNNs, quantization of the first layer is more efficient than the last layer in the processing speed-up.

Some previous works kept the first and last layer in full-precision [22, 23, 24, 107, 140-143], and some quantized these layers with higher precision than other layers [24, 42, 144-146]. Work [42] quantized the activations in 1-bit, except the input of the first layer which is in 8-bit fixed-point. The AdaBits method [145] quantized the weights of the first and last layers in 8-bit fixed-point.

Some researches put forward an efficient solution for achieving more compression by quantization the first and last layers and preserving the model accuracy. Therefore, works [24, 109] offered a solution for quantization of the activations and weights in the last layer efficiently. They added a scale layer in the end of a DCNN. In scale layer, the output of the last layer is multiplied by a scale parameter to harmonize the last layer output with the desired output. Scale parameter is a learnable parameter and converges to the optimum value during training. Work [109] achieved almost 4.5× more compression rate while top-5 accuracy decreased only %1 in NIN-net [147] by using the scale layer and quantizing the last layer in comparison with the case in which the last layer is kept in the full-precision.

6.2 Mixed-Precision

Recently, some works have suggested mixed-precision quantization concerning the sensitivity of the layers in the network in which each layer has a different bit-width. The mixed-precision quantization is an optimization problem with two objectives: accuracy and compression rates. The goal is finding the optimal trade-off between the compression rate and accuracy. Search space for mixed-precision quantization is exponential in the number of layers. Suppose that there are M possible precisions for N layers. The time complexity of the search space for optimum precision is O(M^N). Therefore, finding the optimum point is a challenge. Works [71, 135, 142, 144, 148-153] proposed mixed-precision quantization.

Hessian AWare Quantization (HAWQ) [148] performed mixed-precision quantization in multi stages, where in each stage some layers of network were quantized. The bit-width of the layers was determined based on the layer’s Hessian spectrum and the number of parameters. A layer with smaller Hessian spectrum was less sensitive to quantization. On the other hand, a low bit-width quantization of layers which had large number of parameters led to high compression rate. As a result, HAWQ quantized layers with smaller Hessian spectrum and large number of parameters in lower-precision.

The Bit-level Sparsity Quantization (BSQ) method [135] proposed a regularizer based on the group Lasso [154] for performing the mixed-precision quantization during training. In the Differentiable Neural Architecture Search (DNAS) method [142], a loss function was offered as

\[ L(a,w_a) = \text{CrossEntropy}(a) \times C(\text{Cost}(a)) \]  \hspace{1cm} (62)

In Equation (62), \( a \) is model architecture, and \( \text{Cost}() \) determines the architecture cost. \( C \) is a weighting function for the trade-off between the cross entropy and the cost term. Model cost is computed based on the number of parameters, FLOPs, and bit-width of weights and activations. Equation (62) is a trade-off between accuracy and compression rate.
Work [153] performed mixed-quantization iteratively. It ranked the layers based on accuracy gain. They inserted a proxy classifier after each layer for estimating the accuracy gain of the layers. It quantized the layers which had less accuracy gain firstly. A fin-tunning was performed after choosing the bit-width of all layers.

Some works suggested the mixed-precision quantization channel-wise [120, 155-157]. In general, the optimum precision for different parts of neural network is dependent on the network architecture and dataset. The implementation of the mixed-precision quantization is more complex than the fixed-precision quantization in both software and hardware aspects [146].

7 EVALUATION AND DISCUSSION

To evaluate the quantization methods, the model accuracy, compression ratio, memory consumption, number of operations, energy consumption, and processing speed are commonly reported [17, 25, 42, 67, 71, 76, 101, 109, 112, 127, 135, 140, 143-145, 148, 149, 152, 153, 158-162].

Quantization decreases memory usage since the values are stored in the low bit-width and high-cost floating-point operations are replaced with low-cost simple ones. On the other hand, the accuracy commonly decreases after quantization since the information may be lost, which is due to the fact that the quantization levels are limited and the bigger weights which have more effect on the accuracy are omitted. Although the lower bit-width quantization leads to more memory saving and processing speed-up, the accuracy generally decreases. Therefore, a trade-off between compression ratio and model accuracy is required.

7.1 Datasets

For evaluation of DCNNs in the image classification task, the common benchmark datasets are:

1. MINST [96] is a dataset of handwritten digits with 10 classes corresponding to the 10 digits.
2. CIFAR-10 [163] includes 10 different classes and there are 6000 images of each class.
3. SVHN [164] is a dataset of real-world digits.
4. ImageNet (ILSVRC12) [126, 165] is a large-scale dataset of natural images with high resolution. Images are represented in 1000 categories and normally resized to 224×224 before feeding to the network.

Table 4 determines more specifications of above datasets.

| Dataset  | Training Samples | Test Samples | Type of Images | Size of Images |
|----------|------------------|--------------|----------------|----------------|
| MNIST    | 60K              | 10K          | Gray-scale     | 28×28          |
| CIFAR-10 | 50K              | 10K          | Color (RGB)    | 32×32          |
| SVHN     | 604K             | 26K          | Color (RGB)    | 32×32          |
| ImageNet | About 1.2M       | 100K         | Color (RGB)    | -              |

7.2 Accuracy on Small Datasets

MNIST, CIFAR-10, and SVHN are small datasets in comparison with ImageNet and models have relatively better accuracy on them. CIFAR-10 is used more than MNIST and SVHN in experiments. Table 5 shows the accuracy of SOA methods on CIFAR-10. The results are directly presented from the original papers. The situation of training for a full-precision network is different in various works. For example, they used different learning rates, initial values, and
framework. As a result, the reported accuracy for the full-precision model is different in previous works. In Table 5, we present the minimum to maximum reported accuracy for the full-precision models.

Table 5: THE ACCURACY OF SOME SOA METHODS ON CIFAR-10

| Network                  | Method            | Bit-Width (W/A) | Accuracy (%)       |
|--------------------------|-------------------|-----------------|--------------------|
| VGGNet-Small [21]        | Full-Precision    | 32/32           | 91.7 to 93.8       |
|                          | BinaryConnect [21]| 1/32            | 91.73              |
|                          | BNN [42]          | 1/1             | 89.85              |
|                          | HWGQ [110]        | 1/2             | 92.51              |
|                          | LQ-Nets [116]     | 3/2             | 93.8               |
|                          |                   | 2/32            | 93.8               |
|                          |                   | 2/2             | 93.5               |
|                          |                   | 1/32            | 93.5               |
|                          |                   | 1/2             | 93.4               |
|                          | RBNN [166]        | 1/1             | 91.3               |
|                          | IR-Net [119]      | 1/1             | 90.4               |
|                          | DGRL [161]        | 1/1             | 92.62              |
|                          | DMBQ [120]        | 0.7/32          | 93.7               |
|                          |                   | 1/2             | 93.9               |
| VGGNet-128 [106]         | Full-Precision    | 32/32           | 92.88 to 93        |
| TWN [106]                |                  | 2/32            | 92.56              |
| DSQ [141]                |                  | Mixed           | 91.54              |
| VGGNet-11 [11]           | Full-Precision    | 32/32           | 91.93 to 92.13     |
| BCGD [132]               |                  | 32/4            | 91.31              |
|                          |                  | 4/4             | 90                 |
|                          |                  | 1/4             | 89.12              |
|                          | Yin et al. [167]  | 2/32            | 91.01              |
|                          |                  | 1/32            | 89.28              |
| VGGNet-16 [11]           | Full-Precision    | 32/32           | 93.59              |
| Yin et al. [167]         |                  | 2/32            | 93.2               |
|                          |                  | 1/32            | 91.98              |
| ResNet-18 [8]            | Full-Precision    | 32/32           | 93 to 95.49        |
| Yin et al. [167]         |                  | 2/32            | 94.98              |
|                          |                  | 1/32            | 94.19              |
|                          | RBNN [166]        | 1/1             | 92.2               |
|                          | IR-Net [119]      | 1/1             | 91.5               |
|                          | Liu et al. [153]  | 3.07/4.35       | 94.11              |
|                          |                  | 2.7/3.11        | 94.02              |
|                          |                  | 1.96/2.35       | 93.42              |
| ResNet-20 [8]            | Full-Precision    | 32/32           | 91.25 to 92.96     |
| TTQ [107]                |                  | 2/32            | 91.13              |
| PACT [103]               |                  | 5/5             | 91.7               |
|                          |                  | 4/4             | 91.3               |
|                          |                  | 3/3             | 91.1               |
|                          |                  | 2/2             | 89.7               |
| MLQ [114]                |                  | 2/32            | 90.02              |
| ELQ [115]                |                  | 2/32            | 91.45              |
| Network       | Method               | Bit-Width (W/A) | Accuracy (%) |
|--------------|----------------------|-----------------|--------------|
| LQ-Nets [116]| 1/32                 | 91.15           |              |
|              | 3/32                 | 92              |              |
|              | 3/2                  | 91.6            |              |
|              | 2/32                 | 91.8            |              |
|              | 2/2                  | 90.2            |              |
|              | 1/32                 | 90.1            |              |
|              | 1/2                  | 88.4            |              |
| Yin et al. [167]| 2/32             | 90.07           |              |
|              | 1/32                 | 87.82           |              |
| ProxQuant [137]| 2/32              | 91.6            |              |
| BCGD [132]   | 32/4                 | 91.66           |              |
|              | 4/4                  | 91.65           |              |
|              | 2/4                  | 90.75           |              |
|              | 1/4                  | 89.98           |              |
| HAWQ [148]   | 2/4                  | 92.22           |              |
| RBNN [166]   | 1/1                  | 86.5            |              |
| IR-Net [119] | 1/32                 | 90.8            |              |
|              | 1/1                  | 85.4            |              |
| SLB [168]    | 4/32                 | 92.1            |              |
|              | 4/8                  | 91.8            |              |
|              | 4/4                  | 91.6            |              |
|              | 2/32                 | 92              |              |
|              | 2/8                  | 91.7            |              |
|              | 2/6                  | 91.3            |              |
|              | 1/32                 | 90.6            |              |
|              | 1/8                  | 90.5            |              |
|              | 1/4                  | 90.3            |              |
|              | 1/2                  | 89.5            |              |
|              | 1/1                  | 85.5            |              |
| DMBQ [120]   | 2/32                 | 92.5            |              |
|              | 1/32                 | 91.4            |              |
|              | 2/2                  | 91.7            |              |
|              | 1/2                  | 90.4            |              |
| Liu et al. [153]| 3.3/3.4         | 92.82           |              |
|              | 2.3/2.7              | 91.65           |              |
|              | 1.5/2.3              | 90.64           |              |
| ResNet-32 [8]| Full-Precision       | 32/32           | 92.33 to 93.4|
|              | TTQ [107]            | 92.37           |              |
|              | Yin et al. [167]     | 92.94           |              |
|              | ProxQuant [137]      | 92.35           |              |
|              | 1/32                 | 91.47           |              |
| ResNet-34 [8]| Full-Precision       | 32/32           | 95.7          |
|              | TTQ [107]            | 95.07           |              |
|              | 1/32                 | 94.66           |              |
| ResNet-44 [8]| Full-Precision       | 32/32           | 92.82         |
|              | TTQ [107]            | 92.98           |              |
| Network   | Method       | Bit-Width (W/A) | Accuracy (%) |
|-----------|--------------|-----------------|--------------|
| ProxQuant [137] | 2/32         | 92.95           |
|           | 1/32         | 92.05           |
| ResNet-56 [8] | Full-Precision | 32/32         | 93.03 to 94.46 |
|           | TTQ [107]     | 2/32           | 93.56        |
|           | ELQ [115]     | 2/32           | 93.7         |
|           | ProxQuant [137] | 1/32         | 92.3         |
|           | Liu et al. [153] | 3.37/3.42       | 93.74        |
|           |               | 2.37/2.75       | 92.89        |
|           |               | 1.82/2.42       | 92.22        |
| MobileNetV2 [169] | Full-Precision | 32/32         | 94.24        |
|           | Liu et al. [153] | 3.32/3.39       | 84.82        |
|           |               | 2.48/2.83       | 74.42        |
|           |               | 1.32/2.14       | 63.92        |

The results in Table 5 show that most methods with low bit-width achieved the accuracy close to the full-precision model or sometimes better than it. Yin et al. [167] with 2-bit weight quantization reached the accuracy 94.98 for ResNet-18 which was higher than the full-precision model. TTQ [107] achieved higher accuracy in comparison with the full-precision model in ResNet-44 and ResNet-56. ProxQuant [137] and ELQ [115] by ternary weight quantization achieved the accuracy higher than the full-precision model in ResNet-44 and ResNet-56, respectively.

7.3 Accuracy on ImageNet

ImageNet is a large-scale challenging dataset for achieving high accuracy. Therefore, this dataset is an important benchmark for evaluation of the accuracy of DCNNs models. The accuracy of some SOA methods on ImageNet is presented in Table 6.

| Network             | Method       | Bit-width (W/A) | Top-1 (%) | Top-5 (%) |
|---------------------|--------------|-----------------|-----------|-----------|
| AlexNet [1]         | Full-precision | 32/32         | 56.6 to 61.8 | 80.23 to 83.5 |
| DeepCompress [17]   | CONV:8, FC:5/32 | 57.22         | 80.3      |
|                     | CONV:8, FC:4/32 | 57.21         | 80.27     |
|                     | CONV:4, FC:2/32 | 55.23         | 77.67     |
| Dorefa-net [23]     | 1/4(G:32)     | 53             | -         |
|                     | 1/4(G:6)      | 48.2           | -         |
|                     | 1/3(G:32)     | 48.4           | -         |
|                     | 1/3(G:6)      | 47.1           | -         |
|                     | 1/2(G:32)     | 49.8           | -         |
|                     | 1/2(G:6)      | 46.1           | -         |
|                     | 1/1(G:32)     | 43.6           | -         |
|                     | 1/1(G:6)      | 39.5           | -         |
| TTQ [107]           | 2/32         | 57.5           | 79.7      |
| Miyashita et al. [102] | 32/4        | -              | 77.6      |
|                     | 32/3         | -              | 77.1      |
| QNN [77]            | 1/2         | 51.93          | 73.67     |
| LogNet [108]        | 5/32         | -              | 74.6      |
|                     | 4/32         | -              | 73.4      |
| Tang et al. [109]   | 1/2         | 46.6           | 71.7      |
| Network          | Method                | Bit-width(W/A) | Top-1 (%) | Top-5 (%) |
|------------------|-----------------------|----------------|-----------|-----------|
| HWGQ [110]       | 1/32                  | 52.6           | 75.9      |
|                  | 1/2                   | 52.7           | 76.3      |
| INQ [112]        | 5/32                  | 57.39          | 80.46     |
| FGQ [113]        | 2/8                   | 49.04          | -         |
|                  | 2/4                   | 49             | -         |
| WQ [79]          | 4/4                   | 55.8           | -         |
|                  | 2/3                   | 51.37          | 75.49     |
| Balanced [78]    | 2/2                   | 55.7           | 78        |
| Ziang et al. [24]| 4/4                   | 58             | 81.1      |
|                  | 2/2                   | 51.6           | 76.2      |
| PACT [103]       | 32/4                  | 55.5           | 77.6      |
|                  | 32/3                  | 55.6           | 77.8      |
|                  | 32/2                  | 54.9           | 77.2      |
|                  | 3/3                   | 55.6           | 78        |
|                  | 3/2                   | 54.6           | 77.1      |
|                  | 2/3                   | 55.4           | 77.9      |
|                  | 2/2                   | 55             | 77.7      |
| SLQ [114]        | 5/32                  | 57.56          | 80.5      |
| MLQ [114]        | 2/32                  | 54.24          | 77.78     |
| ELQ [115]        | 2/32                  | 57.88          | 80.22     |
|                  | 1/32                  | 56.95          | 79.77     |
| LQ-Nets [116]    | 2/32                  | 60.5           | 82.7      |
|                  | 2/2                   | 57.4           | 80.1      |
|                  | 1/2                   | 55.7           | 78.8      |
| Yang et al. [170]| 2/32                  | 60.9           | 83.2      |
|                  | 1/32                  | 58.8           | 81.7      |
|                  | 1/2                   | 55.4           | 78.8      |
|                  | 1/1                   | 47.9           | 72.5      |
| KDE-KM [125]     | 4/32                  | 47.43          | 72.1      |
| BitPruning [118] | 3.875/4.375           | 55.07          | -         |
| VGGNet-16 [11]   | Full-precision        | 32/32          | 68.54 to 71.55 | 88.65 to 90.33 |
|                  | DeepCompression [17]  | CONV:8, FC:5/32 | 68.83 | 89.09  |
| Miyashita et al. [102] | 32/4                  | -              | 89.8      |
|                  | 32/3                  | -              | 89.2      |
| LogNet [108]     | 5/32                  | -              | 86        |
|                  | 4/32                  | -              | 85.2      |
|                  | 4/4                   | -              | 84.8      |
| INQ [112]        | 5/32                  | 70.82          | 90.3      |
| SLQ [114]        | 5/32                  | 72.23          | 91        |
|                  | 4/32                  | 71.18          | 90.25     |
|                  | 3/32                  | 68.38          | 88.55     |
| KDE-KM [125]     | 4/32                  | 67.76          | 88.14     |
| VGGNet-Variant [110] | Full-precision        | 32/32          | 69.8      | 89.3     |
| HWGQ [110]       | 1/2                   | 64.1           | 85.6      |
| LQ-Nets [116]    | 2/2                   | 68.8           | 88.6      |
|                  | 1/2                   | 67.1           | 87.6      |
| GoogleNet [2]    | Full-precision        | 32/32          | 68.7 to 71.6 | 88.9 to 91.2 |
| BWN [22]         | 1/32                  | 65.5           | 86.1      |
| QNN [77]         | 6/6 (G:6)             | 66.4           | 83.1      |
| Network          | Method | Bit-width(W/A) | Top-1 (%) | Top-5 (%) |
|------------------|--------|----------------|-----------|-----------|
| INQ [112]        | 4/4    | 66.5           | 83.4      |
| HWGQ [110]       | 5/32   | 30.98          | 89.28     |
| Balanced [78]    | 1/2    | 63             | 84.9      |
| SLQ [114]        | 4/4    | 67.7           | 87.3      |
| LQ-Nets [116]    | 5/32   | 69.1           | 89.19     |
|                  | 2/2    | 68.8           | 88.1      |
|                  | 1/2    | 65.6           | 86.4      |
| ResNet-18 [8]    |        |                |           |           |
|                  | 32/32  | 69.3 to 71.08  | 89.2 to 89.6 |
| TWN [106]        | 2/32   | 61.8           | 84.2      |
| BWN [22]         | 1/32   | 60.8           | 83        |
| XNOR-Net [22]    | 1/1    | 51.2           | 73.2      |
| TTQ [107]        | 2/32   | 66.6           | 87.2      |
| HWGQ [110]       | 1/32   | 61.3           | 83.6      |
|                  | 1/2    | 59.6           | 82.2      |
| ABC [111]        | 1/32   | 68.3           | 87.9      |
|                  | 1/1    | 65             | 85.9      |
| INQ [112]        | 5/32   | 68.98          | 89.1      |
|                  | 4/32   | 68.89          | 89.01     |
|                  | 3/32   | 68.08          | 88.36     |
|                  | 2/32   | 66.02          | 87.13     |
| Balanced [78]    | 2/2    | 59.4           | 82        |
| PACT [103]       | 32/4   | 70             | 89.3      |
|                  | 32/3   | 69.2           | 88.9      |
|                  | 32/2   | 67.5           | 87.6      |
|                  | 3/3    | 68.1           | 88.2      |
|                  | 2/2    | 64.4           | 85.6      |
|                  | 1/32   | 65.8           | 86.7      |
|                  | 1/3    | 65.3           | 85.9      |
|                  | 1/2    | 62.9           | 84.7      |
| SLQ [114]        | 5/32   | 69.09          | 89.15     |
| ELQ [115]        | 2/32   | 67.52          | 88.05     |
|                  | 1/32   | 64.72          | 86.04     |
| LQ-Nets [116]    | 4/32   | 70             | 89.1      |
|                  | 4/4    | 69.3           | 88.8      |
|                  | 3/32   | 69.3           | 88.8      |
|                  | 3/3    | 68.2           | 87.9      |
|                  | 2/32   | 68             | 88        |
|                  | 2/2    | 64.9           | 85.9      |
|                  | 1/2    | 62.6           | 84.3      |
| Yin et al. [167] | 2/32   | 66.3           | 87.3      |
|                  | 1/32   | 63.2           | 85.1      |
| BCGD [132]       | 4/8    | 68.85          | 88.71     |
|                  | 4/4    | 67.36          | 87.76     |
|                  | 1/4    | 64.36          | 85.65     |
| Yang et al. [170]| 32/2   | 65.7           | 86.5      |
|                  | 5/32   | 70.6           | 89.6      |
|                  | 2/32   | 69.1           | 88.5      |
|                  | 1/32   | 66.5           | 87.3      |
|                  | 1/2    | 63.4           | 84.9      |
| Network       | Method         | Bit-width(W/A) | Top-1 (%) | Top-5 (%) |
|--------------|----------------|----------------|-----------|-----------|
|              |                | 1/1            | 53.6      | 75.3      |
| KDE-KM [125] | 4/32           | 61.82          | 83.89     |
| LSQ [171]    | 4/4            | 71.1           | 90        |
|              | 3/3            | 70.2           | 89.4      |
|              | 2/2            | 67.6           | 87.6      |
| DSQ [141]    | Mixed          | 69.27          | -         |
| AdaRound [172]| 4/32           | 68.71          | -         |
|              | 4/8            | 68.55          | -         |
| BitPruning [118]| 3.38/4.14     | 69.19          | -         |
| Bi-RealNet [25]| 1/1           | 56.4           | 79.5      |
| RBNN [166]   | 1/1            | 59.9           | 81.9      |
| LNS [140]    | 1/1            | 59.4           | 81.7      |
| IR-Net [119] | 1/32           | 66.5           | 86.8      |
| SLB [168]    | 2/32           | 68.4           | 88.1      |
|              | 2/8            | 68.2           | 87.7      |
|              | 2/4            | 67.5           | 87.4      |
|              | 2/2            | 66.1           | 86.3      |
|              | 1/32           | 67.1           | 87.2      |
|              | 1/8            | 66.2           | 86.5      |
|              | 1/4            | 66             | 86.4      |
|              | 1/2            | 64.8           | 85.5      |
|              | 1/1            | 61.5           | 83.1      |
| AdaRound [172]| 4/32           | 68.71          | -         |
|              | 4/8            | 68.55          | -         |
| BRECQ [160]  | 4/32           | 70.7           | -         |
|              | 4/4            | 69.6           | -         |
|              | 3/32           | 69.81          | -         |
|              | 2/32           | 66.3           | -         |
|              | 2/2            | 64.8           | -         |
| DGRL [161]   | 1/1            | 60.45          | -         |
| DMBQ [120]   | 2/32           | 70.1           | 89.3      |
|              | 1/32           | 65.9           | 87.1      |
|              | 3/3            | 70             | 89.4      |
|              | 2/2            | 67.8           | 88.1      |
|              | 1/2            | 63.5           | 85.5      |
| Liu et al. [153]| 4.38/4.38      | 70.59          | -         |
|              | 3.35/3.55      | 70.12          | -         |
|              | 2.72/2.72      | 69.84          | -         |
| HAWQ-V3 [144]| 8/8            | 71.56          | -         |
|              | 4/4            | 70.22          | -         |
|              | 4/4            | 68.65          | -         |
| ResNet-34 [8]| Full-precision | 32/32          | 73.27 to 74.1 | 91.26 to 91.8 |
| Venkatesh et al. [122]| 2/32       | 71.6          | 90.37     |
| HWGQ [110]   | 1/2            | 64.3           | 85.7      |
| ABC [111]    | 1/1            | 68.4           | 88.2      |
| LQ-Nets [116]| 3/3            | 71.9           | 90.2      |
|              | 2/2            | 69.8           | 89.1      |
|              | 1/2            | 66.6           | 86.9      |
| BCGD [132]   | 4/8            | 72.18          | 90.73     |
| Network          | Method                   | Bit-width(W/A) | Top-1 (%) | Top-5 (%) |
|------------------|--------------------------|----------------|-----------|-----------|
|                  |                          | 4/4            | 70.81     | 90        |
|                  |                          | 1/4            | 68.43     | 88.29     |
| LSQ [171]        |                          | 4/4            | 74.1      | 91.7      |
|                  |                          | 3/3            | 73.4      | 91.4      |
|                  |                          | 2/2            | 71.6      | 90.3      |
| Bi-RealNet [25]  |                          | 1/1            | 62.2      | 83.9      |
| RBNN [166]       |                          | 1/1            | 63.1      | 84.4      |
| IR-Net [119]     |                          | 1/32           | 70.4      | 89.5      |
| DMBQ [120]       |                          | 2/2            | 72.1      | 90.7      |
|                  |                          | 1/2            | 69.8      | 89.2      |
| ResNet-50 [8]    | Full-precision           | 32/32          | 73.22 to 77.39 | 91.24 to 93.4 |
| Venkatesh et al. [122] |                | 2/32           | 73.85     | 91.8      |
| HWGQ [110]       |                          | 1/2            | 64.6      | 85.9      |
| ABC [111]        |                          | 1/1            | 70.1      | 89.7      |
| INQ [112]        |                          | 5/32           | 74.81     | 92.45     |
| FGQ [113]        |                          | 2/8            | 70.76     | -         |
|                  |                          | 2/4            | 68.38     | -         |
| Zuang et al. [24]|                          | 4/4            | 75.7      | 92        |
|                  |                          | 2/2            | 70        | 87.5      |
| PACT [103]       |                          | 32/4           | 75.9      | 92.9      |
|                  |                          | 5/5            | 76.7      | 93.3      |
|                  |                          | 4/4            | 76.5      | 93.2      |
|                  |                          | 3/3            | 75.3      | 92.6      |
|                  |                          | 2/4            | 74.5      | 91.9      |
|                  |                          | 2/2            | 72.2      | 90.5      |
|                  |                          | 1/2            | 67.8      | 87.9      |
| LQ-Nets [116]    |                          | 4/32           | 76.4      | 93.1      |
|                  |                          | 4/4            | 75.1      | 92.4      |
|                  |                          | 3/3            | 74.2      | 91.6      |
|                  |                          | 2/32           | 75.1      | 92.3      |
|                  |                          | 2/2            | 71.5      | 90.3      |
|                  |                          | 1/2            | 68.7      | 88.4      |
| Yang et al. [170]|                          | 5/32           | 76.4      | 93.2      |
|                  |                          | 2/32           | 75.2      | 92.6      |
|                  |                          | 1/32           | 72.8      | 91.3      |
| LSQ [171]        |                          | 8/8            | 76.8      | 93.4      |
|                  |                          | 4/4            | 76.7      | 93.2      |
|                  |                          | 3/3            | 75.8      | 92.7      |
|                  |                          | 2/2            | 73.7      | 91.5      |
| HAWQ [148]       |                          | 2/4            | 75.48     | -         |
| HAWQ-V2 [149]    |                          | 2/4            | 75.76     | -         |
| SAT [173]        |                          | 4/32           | 76.6      | 93        |
|                  |                          | 3/32           | 76.3      | 93        |
|                  |                          | 2/32           | 75.3      | 92.4      |
|                  |                          | 4/4            | 76.9      | 93.3      |
|                  |                          | 3/3            | 76.6      | 93.1      |
|                  |                          | 2/2            | 74.1      | 91.7      |
| AdaRound [172]   |                          | 4/32           | 75.23     | -         |
|                  |                          | 4/8            | 75.01     | -         |
| Network                  | Method         | Bit-width(W/A) | Top-1 (%) | Top-5 (%) |
|-------------------------|----------------|----------------|-----------|-----------|
| Bi-RealNet [25]         | 1/1            | 62.6           | 83.9      |           |
| PWLQ [174]              | 8/8            | 76.1           | -         |           |
|                         | 6/8            | 76.08          | -         |           |
|                         | 4/8            | 75.62          | -         |           |
|                         | 4/4            | 74.85          | -         |           |
| AdaRound [172]          | 4/32           | 75.23          | -         |           |
|                         | 4/8            | 75.01          | -         |           |
| BRECQ [160]             | 4/32           | 76.29          | -         |           |
|                         | 4/4            | 75.05          | -         |           |
|                         | 3/32           | 75.61          | -         |           |
|                         | 2/32           | 72.4           | -         |           |
|                         | 2/4            | 70.29          | -         |           |
| HAWQ-V3 [144]           | 8/8            | 77.58          | -         |           |
|                         | 4.8/4.8        | 76.73          | -         |           |
|                         | 4/4            | 74.24          | -         |           |
| ResNet-101 [8]          | Full-precision | 32/32          | 77.5 to 78.2 | 94.1 |
| FGQ [113]               | 2/8            | 73.85          | -         |           |
|                         | 2/4            | 70.69          | -         |           |
| LSQ [171]               | 4/4            | 78.3           | 94        |           |
|                         | 3/3            | 77.5           | 93.6      |           |
|                         | 2/2            | 76.1           | 92.8      |           |
| ResNet-152 [8]          | Full-precision | 32/32          | 76.5 to 78.9 | 93.2 to 94.3 |
| Venkatesh et al. [122]  | 2/32           | 76.64          | 93.2      |           |
| LSQ [171]               | 4/4            | 78.5           | 94.1      |           |
|                         | 3/3            | 78.2           | 93.9      |           |
|                         | 2/2            | 76.9           | 93.2      |           |
| Bi-RealNet [25]         | 1/1            | 64.5           | 85.5      |           |
| MobileNetV1 [13]        | Full-precision | 32/32          | 70.9 to 72.4 | 89.9 to 90.2 |
| SAT [173]               | 4/32           | 72.1           | 90.2      |           |
|                         | 3/32           | 70.7           | 89.5      |           |
|                         | 2/32           | 66.3           | 86.8      |           |
|                         | 8/8            | 72.6           | 90.7      |           |
|                         | 6/6            | 72.3           | 90.4      |           |
|                         | 5/5            | 71.9           | 90.3      |           |
|                         | 4/4            | 71.3           | 89.9      |           |
| Phan et al. [175]       | 1/1            | 60.9           | 86.4      |           |
| ReActNet [176]          | 1/1            | 69.4           | -         |           |
| AdaBits [145]           | 8/8            | 72.4           | -         |           |
|                         | 6/6            | 72.4           | -         |           |
|                         | 5/5            | 72.1           | -         |           |
|                         | 4/4            | 71.1           | -         |           |
| MobileNetV2 [169]       | Full-precision | 32/32          | 71.72 to 72.1 | 90 to 90.5 |
| DSQ [141]               | 4/4            | 64.8           | -         |           |
| SAT [173]               | 4/32           | 72.1           | 90.6      |           |
|                         | 3/32           | 71.1           | 89.9      |           |
|                         | 2/32           | 66.8           | 87.2      |           |
|                         | 8/8            | 72.5           | 90.7      |           |
|                         | 6/6            | 72.3           | 90.6      |           |
|                         | 5/5            | 72             | 90.4      |           |
| Network          | Method       | Bit-width(W/A) | Top-1 (%) | Top-5 (%) |
|------------------|--------------|----------------|-----------|-----------|
|                  |              | 4/4            | 71.1      | 90        |
| DJPQ [71]        | Mixed        | 8/8            | 72.6      | -         |
|                  |              | 6/6            | 72.4      | -         |
|                  |              | 5/5            | 72.1      | -         |
|                  |              | 4/4            | 70.8      | -         |
|                  | AdaBits [145]| 4/32           | 69.78     | -         |
|                  |              | 4/8            | 69.25     | -         |
|                  | BitPruning [118] | 4.15/4.57     | 70.09     | -         |
| SqueezNet [12]   | Full-precision| 32/32          | 69.38     | -         |
|                  | HAWQ [148]   | 3/8            | 68.02     | -         |
|                  | HAWQ-V2 [149]| 3/8            | 68.38     | -         |
| DenseNet-121 [177]| Full-precision| 32/32          | 75        | 92.3      |
| LQ-Nets [116]    | 2/2          | 69.6           | 89.1      |           |

It is inferred from Table 6 that some weight quantization methods (with full-precision activations) achieved the accuracy close to the full-precision model even with 1-bit or 2-bit quantization. For clearer comparison, Figure 17 represents top-1 accuracy of the 1-bit (binary) and 2-bit (ternary) weight quantization methods on ResNet-18 [8] including: TWN [106], Binary-Weight-Network (BWN) [22], TTQ [107], HWGQ [110], ABC [111], INQ [112], PACT [103], ELQ [115], LQ-Nets [116], Yin et al. [167], MLQ [114], SLB [168], BRECQ [160], IR-Net [119]. The accuracy of the full-precision model is extracted from work [111].

![Figure 17: Top-1 accuracy of 1-bit (binary) and 2-bit (ternary) weight quantization methods on ResNet-18.](image-url)

When both weights and activations were quantized, some methods with 5-bit or 4-bit quantization achieved the accuracy higher than the full-precision model. When the bit-width was reduced to 3, 2, and 1, the accuracy of some methods like
PACT [103], LQ-Nets [116], and LSQ [171] was close to the full-precision model. Nevertheless, the accuracy commonly decreased and it was more significant in the deeper networks. To compare more clearly, Figure 18 shows top-1 accuracy of some binary quantization methods (XNOR-Net [22], ABC [111], Bi-RealNet [25], DGRL [161], RBNN [166] and SLB [168], LNS [140]) on ResNet-18 [8]. The accuracy of the full-precision model comes from [111].

![Figure 18: Top-1 accuracy of some binary methods (binarized weights and activations) on ResNet-18.](image)

8 CONCLUSION AND FUTURE WORKS

In this paper, we surveyed the previous quantization works in the image classification task. The basic and advanced concepts in DCNN quantization were described. The most important works and methods were introduced and advantages and challenges in this topic were discussed.

All numerical components in a neural network can be quantized and weights are the most common component for quantization. The methods which quantize both weights and activations have more compression rate and simple operations. The quantization of activations is more difficult than weights, which is due to: firstly, the wide range of activations, secondly, using a non-differentiable activation function and its estimation in the backward pass, and finally having different values in the inference time, unlike the weights.

The QAT and PTQ were investigated and concluded that the QAT approach normally leads to higher accuracy than the PTQ in the inference time. In QAT, the training of a quantized DNN is a challenge, which is different from the training of the full-precision network since the units are discrete. A quantized network requires extra iterations during training for convergence. Moreover, adaptive training to the quantized network is required for achieving an accurate model. For instance, learning rate and regularization can be different from the training in the full-precision network.

Some quantization approaches such as uniform and non-uniform were discussed and it was concluded that the non-uniform and especially POT quantization covers the distribution of full-precision values efficiently. For decreasing the quantization error, quantization levels must be allocated to the informative places and matched with the distribution of full-precision values. Using the scale factor helps shifting the quantization levels to the most informative parts of data.

Some previous methods which quantized both weights and activations in low bit-width achieved high accuracy on large-scale datasets like ImageNet. However, it is still challenging in the quantization with lower than 4-bit precision. The problem is more severe in the quantization of deeper networks. In the training of a quantized networks, STE is commonly used for
estimating the activation function and calculating the gradient in the backward pass. The noise due to gradient mismatch from an inaccurate estimation is amplified layer by layer from the end of the network to the initial layers. In the deeper networks, the amplification of the noise is more than shallow networks. This noise leads to divergence in the network training. On the other hand, since the number of parameters increases with the depth of the neural network, the range of parameters in the deeper networks is wider than in shallow ones, and the quantization is difficult. Therefore, Future works should consider the weights and activations quantization of deeper networks in low bit-width precision such as binary or ternary.

Mixed-precision is currently an interesting approach in quantization. The main challenge in the mixed-precision approach is exponential time complexity to find the optimum bit-width for each layer. accordingly, finding a solution for the optimum mixed-precision with the polynomial time complexity is desirable for future works.

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. Advances in neural information processing systems, vol. 25, 1097–1105.
[2] Christian Szegedy, Wei Liu, Yangqing Jia, et al. 2015. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, 1–9.
[3] Alexis Conneau, Holger Schwenk, Loic Barrault, and Yann LeCun. 2016. Very deep convolutional networks for natural language processing. arXiv:1606.01781, vol. 2 http://arxiv.org/abs/1606.01781
[4] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Improving multi-task deep neural networks via knowledge distillation for natural language understanding. arXiv:1904.09482 http://arxiv.org/abs/1904.09482
[5] Xiaodong Liu, Yu Wang, Jianfeng Gao, et al. The microsoft toolkit of multi-task deep neural networks for natural language understanding. arXiv:2002.07972, 2020 http://arxiv.org/abs/2002.07972
[6] Geoffrey Hinton, Li Deng, Dong Yu, et al. 2012. Deep neural networks for acoustic modeling in speech recognition. The shared views of four research groups. IEEE Signal processing magazine, vol. 29, no. 6, 82–97.
[7] Ying Zhang, Mohammad Pezeshki, Philémon Brakel, et al. 2017. Towards end-to-end speech recognition with deep convolutional neural networks. arXiv:1701.02120 http://arxiv.org/abs/1701.02120
[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, 770–778.
[9] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, 7132-7141.
[10] Pierre Sermanet, David Eigen, Xiang Zhang. 2013. Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv:1312.6229 http://arxiv.org/abs/1312.6229
[11] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556 http://arxiv.org/abs/1409.1556
[12] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, et al. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and >650MB model size. arXiv:1602.07360 http://arxiv.org/abs/1602.07360
[13] Andrew G. Howard, Menglong Zhu, Bo Chen, et al. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861 http://arxiv.org/abs/1704.04861
[14] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE conference on computer vision and pattern recognition, 6848–6856.
[15] Kusyear Hwang and Wonjong Sung. 2014. Fixed-point feedforward deep neural network design using weights +1, 0, and −1. In 2014 IEEE Workshop on Signal Processing Systems (SiPS), 1–6.
[16] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. arXiv:1412.6115 http://arxiv.org/abs/1412.6115
[17] Song Han, Huizi Mao, and William J. Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv:1510.00149 http://arxiv.org/abs/1510.00149
[18] Yiwen Guo, Anbang Yao, and Yuqong Chen. 2016. Dynamic network surgery for efficient DNNs. In Proceedings of the 39th International Conference on Neural Information Processing Systems, 1387–1395.
[19] Pavlo Molchanov, Stephen Tyree, Tero Karras, et al. 2017. Pruning convolutional neural networks for resource efficient inference. In 5th International Conference on Learning Representations, ICLR 2017-Conference Track Proceedings 2019.
[20] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017, Thinet: A filter level pruning method for deep neural network compression. In Proceedings of the IEEE international conference on computer vision, 5058–5066.
[21] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, vol. 28, 3120-3131.

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

[23] Shuchang Zhou, Yuxin Wu, Zekun Ni, et al. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv:1606.06160 http://arxiv.org/abs/1606.06160

[24] Bohan Zhuang, Chunhua Shen, Mingkui Tan, et al. 2018. Towards Effective Low-Bitwidth Convolutional Neural Networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7920-7928.

[25] Zechun Liu, Wenhan Luo, Boyuan Wu, et al. 2020. Bit-real net: Binarizing deep network towards real-network performance. *International Journal of Computer Vision*, vol. 128, no. 1, 202-219.

[26] Tianshi Chen, Zidong Du, Ninghui Sun, et al. 2014. DiaNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News*, 42, no. 1: 269-284.

[27] Zidong Du, Robert Fasthuber, Tianshi Chen, et al. 2015. ShiflNao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 92-94.

[28] Xiaofan Zhang, Junsong Wang, Chao Zhu, et al. 2018. DNNBuilder: an automated tool for building high-performance DNN hardware accelerators for FPGAs. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1-8.

[29] Yong Yu, Tian Zhi, Xuad Zhou. 2019. BSHIFT: a low cost deep neural networks accelerator. *International Journal of Parallel Programming*, vol. 47, no. 3, 360-372.

[30] Luqiang Lu and Yun Liang. 2018. SpWA: An efficient sparse winograd convolutional neural networks accelerator on FPGAs. In *Proceedings of the 55th Annual Design Automation Conference*, 1-6.

[31] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, 292-308.

[32] Shubham Jain, Sumeet Kumar Gupta, and Anand Raghunathan. 2020. TiM-DNN: Terabyte In-Memory Accelerator for Deep Neural Networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

[33] Suhas Shivapakshi, Haridik Jain, Olaf Hellwich, and Friedel Gerfers. 2020. A Power Efficient Multi-Bit Accelerator for Memory Prohibitive Deep Neural Networks. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1-5.

[34] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. 2015. Net2net: Accelerating learning via knowledge transfer. arXiv:1511.05641 http://arxiv.org/abs/1511.05641

[35] Ping Luo, Zhenyao Zhu, Ziwei Liu, et al. 2016. Face model compression by distilling knowledge from neurons. In *Thirtyeth AAAI conference on artificial intelligence*.

[36] Zheng Xu, Yen-Chang Hsu, and Jiawei Huang. 2017. Training shallow and thin networks for acceleration via knowledge distillation with conditional adversarial networks. arXiv:1709.00513 http://arxiv.org/abs/1709.00513

[37] Aarti Mishra and Debbie Marr. 2018. Apprentice: Using Knowledge Distillation Techniques To Improve Low-Precision Network Accuracy. In *International Conference on Learning Representations*.

[38] Jangho Kim, SeongUk Park, and Nojun Kwak. 2018. Paraphrasing complex network: network compression via factor transfer. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2765-2774.

[39] Raphael Tang, Yao Lu, Linqing Liu. 2019. Distilling task-specific knowledge from bert into simple neural networks. arXiv:1903.12136 http://arxiv.org/abs/1903.12136

[40] Clément Farabet, Cyril Poulet, Jefferson Y Han, and Yann LeCun. 2009. Cnp: An fpga-based processor for convolutional networks. In *2009 International Conference on Field Programmable Logic and Applications*, IEEE, 32-37.

[41] Yunji Chen, Tao Luo, Shaoli Liu, et al. 2014. Dadanna: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE, 609-622.

[42] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, et al. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. arXiv:1602.02830 http://arxiv.org/abs/1602.02830

[43] Wenlin Chen, James Wilson, Stephen Tyree, et al. 2015. Compressing neural networks with the hashing trick. In *International conference on machine learning*. PMLR, 2285-2294.

[44] Adam Polyak and Lior Wolf. 2015. Channel-level acceleration of deep face representations. *IEEE Access*, no. 3, 2163-2175.

[45] Yihui He, Yangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, 1389-1397.

[46] Xiuyu Yu, Tongliang Liu, Xunchao Wang, and Dacheng Tao. 2017. On Compressing Deep Models by Low Rank and Sparse Decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 67-76.

[47] Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Advances in neural information processing systems*, vol. 2, 598-605.

[48] Steven J. Nowlan and Geoffrey E. Hinton. 1992. Simplifying neural networks by soft weight-sharing. *Neural computation*, vol. 4, no. 4, 473-493.

[49] Gal Chechik, Isaac Meilisins, and Etan Ruppin. 1998. Synaptic pruning in development: a computational account. *Neural computation*, vol. 10, no. 7, 1759-1777.

[50] Babak Hassibi and David Stork. 1992. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*.
Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv:1710.09282*.

Ratko Pilipović, Patricia Bušić, and Vladimir Ružić. 2018. Compression of convolutional neural networks: A short survey. In *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, IEEE, 1-6.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2018. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, vol. 35, no. 1, 126-136.

Vadim Lebedev and Victor Lempitsky. 2018. Speeding-up convolutional neural networks: A survey. *Bulletin of the Polish Academy of Sciences. Technical Sciences*, vol. 66, no. 6.

Jian Cheng, Fei-song Wang, Gang Li, et al. 2018. Recent advances in efficient computation of deep convolutional neural networks. *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 1, 64-77.

Jian Cheng, Peisi-wong Wang, Gang Li, et al. 2018. A survey on acceleration of deep convolutional neural networks. *arXiv:1802.00939*.

Xin Long, Zongheng Ben, and Yan Liu. 2019. A survey of related research on theory and acceleration of deep neural networks. In *Journal of Physics: Conference Series*, vol. 1213, no. 5, p. 052003. IOP Publishing.

Lei Deng, Guo-pu Li, Song Han, et al. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, vol. 108, no. 4, 485-532.

Tejalal Choudhary, Vipul Mishra, Anurag Gooawami, and Jagannathan Sarangapani. 2020. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, 1-43.

Abhinav Goel, Caleb Tung, Yung-Hsang Lu, and George K. Thiruvathukal. 2020. A survey of methods for low-power deep learning and computer vision. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, IEEE, 1-6.

Manish Gupta and Puneet Agrawal. 2020. Compression of deep learning models for text: A survey. *arXiv:2008.05221*.

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

Haotong Qin, Ruihao Gong, Xianglong Liu, et al. 2020. Binary neural networks: A survey. *Pattern Recognition*, vol. 105, 107281.

Yunhui Guo. 2018. "A survey on methods and theories of quantized neural networks," *arXiv:1808.04752*.

Amir Gholami, Sehoon Kim, Zhen Dong, et al. 2021. A survey of quantization methods for efficient neural network inference. *arXiv:2103.13630*.

Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, et al. 1989. Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, vol. 37, no. 3, 328-339.

Sanna LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, vol. 86, no. 11, 2278-2324.

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, vol. 105, no. 12, 2295-2329.

Fei-Fei Li, Andrej Karpathy, and Justin Johnson. 2015. C231n: Convolutional neural networks for visual recognition. *University Lecture*.

Alex Berg, Jia Deng, and Fei-Fei Li. 2010. Large scale visual recognition challenge (ILSVRC), 2010. vol. 3.

Minje Kim and Paris Smaragdis. 2016. Bitwise neural networks. *arXiv:1603.01025*.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2018. Model compression and acceleration for deep neural networks using logarithmic data representation. *arXiv:1603.01025*.

Jungwook Choi, Zhiwang Wang, Swagath Venkataramani, et al. 2018. Pact: Parameterized clipping activation for quantized neural networks. *arXiv:1805.06085*.

Aast Mishra, Erikio Norvidad, Jeffrey J. Cook, and Debbie Marr. 2018. WRPN: Wide Reduced-Precision Networks. In *International Conference on Learning Representations*.

Wonyong Sung, Sungho Shin, and Kyuseon Hwang. 2017. Resiliency of deep neural networks under quantization. *arXiv:1511.06488*.

Fengli Li and Bin Liu. 2016. Ternary Weight Network. *arXiv:1605.04711*.

Chenzhou Zhu, Song Han, Huai Mao, and William J. Dally. 2016. Ternary convolutional neural networks. *arXiv:1612.01064*.

Edward H. Lee, Daisuke Miyashita, Elaina Chai, et al. 2017. Lognet: Energy-efficient neural networks using logarithmic computation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 5900-5904.

Wei Tang, Gang Hua, and Liang Wang. 2017. How to train a compact binary neural network with high accuracy? In *Thirty-First AAAI conference on artificial intelligence*.

Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. 2017. Deep learning with low precision by half-wave gaussian quantization. In *Proceedings*.
of the IEEE conference on computer vision and pattern recognition, 5918-5926.

[111] Xiaofan Liu, Cong Zhao, and Wei Pan. 2017. Towards accurate binary convolutional neural network. In Proceedings of the 31st International Conference on Neural Information Processing Systems, 344-352.

[112] Aojun Zhou, Anbang Yao, Yiwen Guo. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. arXiv:1702.03044 http://arxiv.org/abs/1702.03044

[113] Naveen Meltempudi, Abhishek Kundu, Dheeravatsa Mudigere, et al. 2017. Ternary neural networks with fine-grained quantization. arXiv:1705.01462 http://arxiv.org/abs/1705.01462

[114] Yuhui Xu, Yongzhong Wang, Zhou Zhang, et al. 2018. Deep neural network compression with single and multiple level quantization. In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1.

[115] Aojun Zhou, Anbang Yao, Kuan Wang, and Yuqong Chen. 2018. Explicit loss-error-aware quantization for low-bit deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, 9426-9435.

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

[117] Li Yuhang, Xin Dong, and Wei Pan. 2019. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. arXiv:1909.13144 http://arxiv.org/abs/1909.13144

[118] Niklits Molos, Ghouzhi Buodi Huorne, Casera Bannoo, et al. 2020. Bitpruning: Learning bitlengths for accurate quantization. arXiv:2002.03090 http://arxiv.org/abs/2002.03090

[119] Haotong Qin, Ruiaodong Gong, Xianglong Liu, et al. 2020. Forward and backward information retention for accurate binary neural networks. In 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, 2247-2256.

[120] Zhao Sijie, Tao Yue, and Xuemai Hu. 2021. Distribution-aware adaptive multi-bit quantization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 9281-9290.

[121] Ryan Razani, Gouery Morin, Eyyush Sari, and Sahid Partovi Nia. 2021. Adaptive binary-ternary quantization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 4613-4618.

[122] Ganesh Venkatesh, Erkko Nurvitadhi, and Debbie Marr. 2017. Accelerating deep convolutional networks using low-precision and sparsity. In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2861-2865.

[123] Tapani Raiko, Matthias Berglund, Guillaume Alain, and Laurent Dinh. 2014. Techniques for learning binary stochastic feedforward neural networks. arXiv:1406.2999 http://arxiv.org/abs/1406.2999

[124] Nayak Prateeth, David Zhang, and Sok Chai. Bit efficient quantization for deep neural networks. 2019. Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing, NeurIPS Edition (EMC2-NIPS) IEEE, 52-56.

[125] Sanghyun Seo and Juntae Kim. 2019. Efficient weights quantization of convolutional neural networks using kernel density estimation based non-uniform quantizer. Applied Sciences, vol. 9, no. 12, 2559.

[126] Jia Deng, Wei Dong, Richard Socher, et al. 2009. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, 248-255.

[127] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision, 1026-1034.

[128] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1985. Learning internal representations by error propagation. in Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations, California Univ San Diego La Jolla Inst for Cognitive Science.

[129] Frank Rosenblatt. 1957. The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory.

[130] Geoffrey Hinton. 2012. Neural networks for machine learning. Coursera, video lectures https://www.cs.toronto.edu/~hinton/coursera_lectures

[131] Yoshua Bengio, Nicolas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv:1308.3432 http://arxiv.org/abs/1308.3432

[132] Penghang Yin, Shuai Zhang, Jiachen Lyu, et al. 2020. Blended coarse gradient descent for full quantization of deep neural networks. Research in the Mathematical Sciences, vol. 6, no. 1, 1-14.

[133] Penghang Yin, Jiachen Lyu, Shuai Zhang, et al. 2018. Understanding Straight-Through Estimator in Training Activation Quantized Neural Nets. In International Conference on Learning Representations.

[134] Pengyu Cheng, Chang Liu, Chunchuan Li, et al. 2019. Straight-through estimator as projected Wasserstein gradient flow. arXiv:1910.02176 http://arxiv.org/abs/1910.02176

[135] Huanru Yi, Lin Du, Yiran Chen, and Hui Li. 2020. BQ: Exploring bit-level sparsity for mixed-precision neural network quantization. In International Conference on Learning Representations.

[136] Naamanon Maxim, Utku Durlu, Jongho Park, et al. 2018. On periodic functions as regularizers for quantization of neural networks. arXiv:1811.09862 http://arxiv.org/abs/1811.09862.

[137] Yu Bai, Yu-Xiang Wang, and Edo Liberty. 2018. ProxQuant: Quantized neural networks via proximal operators. In International Conference on Learning Representations.

[138] Wes Matthias, Sai Manoj Pudukottai Dimkarao, and Axel Jantsch. Weighted quantization-regularization in DNNs for weight memory minimization toward HW implementation. 2018. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 11, 2929-2939.

[139] Stefan Ulrich, Lukas Mauch, Kazuki Yoshiyama, et al. 2019. Differentiable quantization of deep neural networks. arXiv:1905.11452 vol. 2, no. 8,
[170] Jiwei Yang, Xu Shen, Jun Xing, et al. 2019. Quantization Networks. In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE Computer Society, 7300-7308.

[171] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, et al. 2019. Learned step size quantization. arXiv:1902.08153 http://arxiv.org/abs/1902.08153

[172] Markus Nagel, Rana A. Amjad, Mart V. Baalen, et al. 2020. Up or down? adaptive rounding for post-training quantization. In International Conference on Machine Learning, PMLR, 7197-7206.

[173] Qing Jin, Linjie Yang, and Zhenyu Luo. 2019. Towards efficient training for neural network quantization. arXiv:1912.10207 http://arxiv.org/abs/1912.10207

[174] Jun Fang, Ali Shafiee, Hamzah Abdel-Aziz, et al. 2020. Post-training piecewise linear quantization for deep neural networks. In European Conference on Computer Vision, Springer, Cham, 69-86.

[175] Hai Phan, Zechun Liu, Dang Huynh, et al. 2020. Binarizing mobilenet via evolution-based searching. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 13420-13429.

[176] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. 2020. Reactnet: Towards precise binary neural network with generalized activation functions. In European conference on computer vision, Springer, Cham, 143-159.

[177] Gao Huang, Zhuang Liu, Laurens V. D. Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, 4700-4708.