# Towards Accurate Low Bit-Width Quantization with Multiple Phase Adaptations

**Zhaoyi Yan,[1] Yemin Shi,[2] Yaowei Wang,[3] Mingkui Tan,[3]**
**Zheyang Li,[4] Wenming Tan,[4] Yonghong Tian[2]***
[1]School of ECE, Peking University Shenzhen Graduate School
[2]School of EECS, Peking University, [3]Pengcheng Laboratory, [4]Hikvision Research Institute

## Abstract

Low bit-width model quantization is highly desirable when deploying a deep neural network on mobile and edge devices. Quantization is an effective way to reduce the model size with low bit-width weight representation. However, the unacceptable accuracy drop hinders the development of this approach. One possible reason for this is that the weights in quantization intervals are directly assigned to the center. At the same time, some quantization applications are limited by the various of different network models. Accordingly, in this paper, we propose Multiple Phase Adaptations (MPA), a framework designed to address these two problems. Firstly, weights in the target interval are assigned to center by gradually spreading the quantization range. During the MPA process, the accuracy drop can be compensated for the unquantized parts. Moreover, as MPA does not introduce hyperparameters that depend on different models or bit-width, the framework can be conveniently applied to various models. Extensive experiments demonstrate that MPA achieves higher accuracy than most existing methods on classification tasks for AlexNet, VGG-16 and ResNet.

## Introduction

Over the last decade, we have witnessed the dramatic development of deep neural networks (DNNs) in many areas, the image classification (Krizhevsky, Sutskever, and Hinton 2012; He et al. 2016; Simonyan and Zisserman 2014), object detection (Girshick et al. 2014), and semantic segmentation (Long, Shelhamer, and Darrell 2015). With the aim of improving the network performance, the network models have become deeper and more complex. Unfortunately, the relationship between the number of parameters and the performance is not linear, meaning that there tends to be a lot of redundancy in these models. More critically, a well-trained DNN model is difficult to deploy on most resource-limited devices due to the network model's huge size. Deep neural network quantization is one of the common methods used for network model compression. Obviously, the fewer the bit-width we use, the worse accuracy we will get. For example, a bit-width of 4 means that there are only 16 values that can be used to represent more than 10 million weights. Even small errors from quantization may thus lead to a significant drop in performance.

(Miyashita, Lee, and Murmann 2016) use base-2 logarithmic representation to compress AlexNet (Krizhevsky, Sutskever, and Hinton 2012) parameters to 3-bit, resulting in a 1.4% drop in top-5 accuracy. Moreover, in (Ullrich, Meeds, and Welling 2017), a version of "soft weight-sharing" is used, with a resultant 2.02% accuracy drop of the ResNet (light) model on CIFAR-100. The problem here is how to compress the network model with acceptable performance loss. Researchers have developed some methods involving complex processes designed to reduce the loss; however, these processes bring with them additional parameters (hyperparameters). In INQ (Zhou et al. 2017), a group of global hyperparameters are handcrafted and pre-determined. In CLIPQ (Tung and Mori 2018), each layer has two distinct hyperparameters that differ from the other layers. SLQ (Xu et al. 2018) is similar, which owns global parameters for different bit-width representation. Although these methods obtain good results with well-selected hyperparameters, their practical application is limited by the variation of hyperparameters between various network models.

In general, there are two main challenges associated with model compression; the first is the trade-off between expected bit-width and model accuracy, while the second is the non-uniform hyperparameter selection. Therefore, an effective and easy-configure quantization framework, Multiple Phase Adaptations (MPA), is proposed in this paper to address the above problems. Unlike the rough quantization process, we offer a smoother method of conducting weight quantization. There are three main steps in our proposed framework: division, MPA, and fine-tuning. We can apply MPA on different models such as AlexNet, VGG16 and ResNet. For example, the quantized ResNet-18 model with bit-width of 4 even achieves better performance than the full-precision baseline. Other experiments also show better or similar results with the state-of-the-art methods. We further report the linear and exponential quantization results for potential inference acceleration.

In summary, our contributions to network quantization are as follows:

*Corresponding author: Yonghong Tian (yhtian@pku.edu.cn)

- Our approach divides quantization into multiple phases. In each phase, the range of quantized weights gradually increases; this has the advantage of making fine-tuning more effective and reducing the model's accuracy loss.
- For different models, we do not introduce hyperparameters that depend on different models or bit-width. So we can apply our approach on various models.
- Beyond the cluster method, we also try different division methods to explore their effectiveness. This is useful for further model acceleration.

## Related Work

In this section, we will present a brief review of the previous work on deep network compression methods.

**Low-rank factorization**. By using matrix or tensor decomposition like Singular Value Decomposition (SVD) (Denil et al. 2013) to estimate the parameters, we can achieve 3x compression rate. In (Lebedev et al. 2014), Canonical Polyadic (CP) uses nonlinear least squares to compute the decomposition. The work in (Yu et al. 2017) presents a better result that can compress a model by 10x. However, the decomposition is less useful for small kernel size, which hinders the development.

**Parameter pruning**. Designed to remove redundant parameters that are not sensitive to the accuracy, pruning is a direct means of reducing complexity and accelerating computation. In (Hassibi and Stork 1993; LeCun, Denker, and Solla 1990), the Optimal Brain Damage and Optimal Brain Surgeon methods are applied to reduce connections by the Hessian of the loss function. (Han, Mao, and Dally 2015) introduces the idea of setting parameters to zero under thresholds, which is simple and effective. (Li et al. 2016) further propose pruning to prune entire network structures such as filters and channels. Traditional pruning is based on removing the small-weight connections.

**Quantization and binarization**. (Gong et al. 2014) use k-means clustering to apply vector quantization. (Han, Mao, and Dally 2015) present deep compression that updates weights together in the same cluster. Huffman coding is also adopted for further compression. However, the fine-tuning phase is time-consuming and makes it difficult to get convergence. Recently, (Xu et al. 2018) propose single and multiple level quantization to exploit the depth information in order to generate a low-bit compressed network. As for incremental quantization methods such as INQ (Zhou et al. 2017) and ELQ (Zhou et al. 2018), they achieve lossless accuracy results. CLIPQ (Tung and Mori 2018) combines quantization and pruning, but each layer has two distinct hyperparameters that differ from the other layers. (Zhang et al. 2018) propose learnable quantizers to solve the accuracy problem. Binarization is an extreme form of quantification. BinaryNet (Courbariaux et al. 2016) and XNORNet (Rastegari et al. 2016) are successful attempts; however the fatal drawback is the significant accuracy drop.

**Other methods**. Knowledge distillation (Hinton, Vinyals, and Dean 2015) makes use of knowledge transfer to shift knowledge from a large teacher model into a small one by learning the class distributions output via softened softmax. (Luo et al. 2016) represent the knowledge by using the neurons at the higher hidden layer, which preserves as much information as the label probabilities, but is more compact. One of the disadvantages of these methods is that they can only be applied to classification tasks with a softmax loss function. Some structures can also help with compression. (Howard et al. 2017) propose MobileNets, which uses depthwise separable convolutions to build lightweight deep neural networks. (Zhang et al. 2017) utilize point-wise group convolution and channel shuffle to reduce computation, thereby achieving a 13x-speed up on AlexNet.

## Notation and Preliminaries

Throughout the paper, we use $w^i$ to denote the weight value of the corresponding $i$-th layer of the model. The INQ divides the whole parameters into two parts; one is for quantized weights $F = \{\phi_F\}$, while the other set $R = \{\phi_R\}$ contains the intervals of fine-tune. The authors proposed accumulated portions of quantized weights at iterative steps to change the elements of the two parts. The iterative step is a real number in the range from 0 to 1 that expresses the proportion of quantized weights to total weights. Moreover, there are about four or five steps in the complete quantization process, meaning that this is a human designed discrete process. For AlexNet, VGG16 and GoogleNet, they use three different setting parameters. By contrast, in our method, weights can be divided into three sets. We add a new interval, $A = \{\phi_A\}$, to hold the weights to be quantized. In $A$, the range to conduct quantization is gradually expanded as the fine-tuning progresses. Thus, we do not need to compute the step value by ourselves; all that is required is to set the interval centers. We will discuss the different methods used to get centers in the following sections.

Once we have the interval center set $C^i$ of the i-th layer, we can divide the weights into $k$ intervals with $k+1$ endpoints, which can be calculated from the two adjacent centers described in Eq. (1) below. $c$ is the element in the interval center set $C$. For the condition $j = 0$ or $j = k$, we choose the extremum of this interval as the endpoint.

$$\begin{aligned} e_0^i &= min(w^i), e_k^i = max(w^i) \\ e_j^i &= (c_j^i + c_{j+1}^i)/2, j = 1, 2, \cdots, k-1 \end{aligned} \tag{1}$$

Based on the endpoints, the intervals can be defined as follows:

$$\begin{aligned} &\phi_1^i = \left\{w^i \in \left[e_0^i, e_1^i\right)\right\}, \phi_2^i = \left\{w^i \in \left[e_1^i, e_2^i\right)\right\}, \cdots, \\ &\phi_j^i = \left\{w^i \in \left[e_{j-1}^i, e_j^i\right)\right\}, \cdots, \phi_k^i = \left\{w^i \in \left[e_{k-1}^i, e_k^i\right]\right\} \end{aligned} \tag{2}$$

$$\phi_1^i \cup \phi_2^i \cup \cdots \cup \phi_k^i = w^i, \phi_1^i \cap \phi_2^i \cap \cdots \cap \phi_k^i = \emptyset \tag{3}$$

Note that there is an one-to-one correspondence between each interval and center. This means that the weights in interval $\phi_j^i$ will equal the center value $c_j^i$ following Multiple Phase Adaptations.

Moreover, in order to control the gradient of frozen weights, we use $M_F^i$ as the mask tensor, which has the same shape as the weights $w^i$.

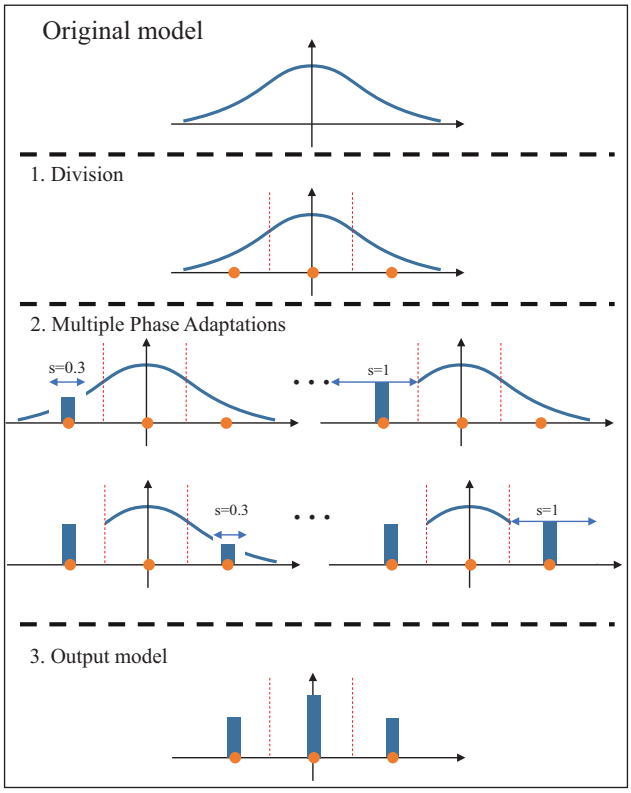
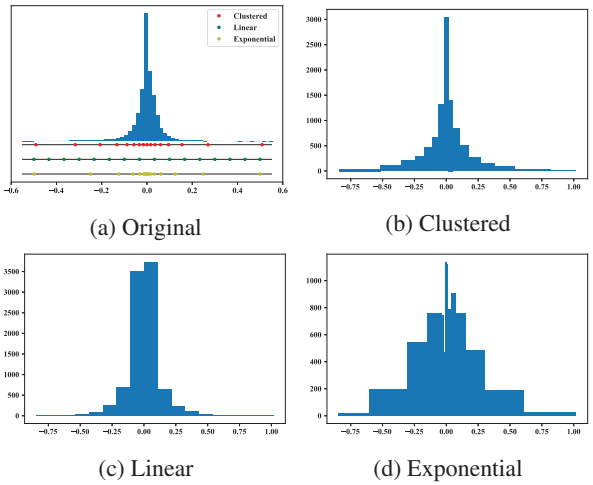



Figure 1: An overview of the Multiple Phase Adaptations Deep Quantization framework. The original model contains full-precision weights and is shown as the blue continuous curve. Depending on these weights, we divide them into three intervals (divided by dotted line), as part (1.) shows. We then begin to quantize the weights in the left part while fine-tuning others. As the training iteration progresses, the quantization range $s$ becomes larger and weights are gathered to the interval center (orange point). When $s = 1$, the whole left part is quantized and becomes discrete. If unquantized intervals remain, we repeat this operation on the next interval. After all intervals are well quantized, we get a compressed output model with little accuracy loss.

## Multiple Phase Adaptations for Compression

### Overview

Deep neural network quantization is one of the commonly used methods of network model compression. By using linear quantization, most models can be compressed from 32-bit to 8-bit with only a small accuracy drop. However, when we reduce the bit-width to 6-bit, the accuracy drop becomes unacceptable; each dismissed bit in the compressing process could potentially affect the model accuracy. It should be noted that the traditional quantization method is a rough process, as it simply uses the center value for the presentation of all parameters in the same interval, making it difficult to solve this problem. By contrast, we here present a smooth way to gather weights, which we refer to as Multiple Phase Adaptations.




(a) Original (b) Clustered

(c) Linear (d) Exponential

Figure 2: Four figures showing the original, clustered, linear and exponential weight distribution. Bar width represents the interval range, while bar height indicates the number of weights in this interval. In (a), the points below also illustrate the difference between division methods.

The entire algorithm comprises division, Multiple Phase Adaptations, and fine-tuning phases. First, divide all weights in each layer to get the quantization centers; thus, we can compute the adaptation and fine-tuning interval boundary for each step. Second, we apply different strategies in different intervals. Adaptation is the key point for quantization, as it will help the weights to approach the quantization center gradually with L1 weight loss. Once the weights are quantized to the interval center, they will be totally frozen, which means the weights cease backward propagation and update. Only one interval is conducting adaptation at any given step; at the same time, the other intervals are fine-tuning under the condition that the values are constrained to be outside of the frozen intervals. The overview is shonw as Figure 1. More details will be presented in the following sections.

### Multiple Phase Adaptations

In our paper, we use Multiple Phase Adaptations to overcome the shortcoming of rough quantization. Generally speaking, weights in the adaptation interval will equal the center value while fine-tuning other unfrozen intervals. Multiple Phase Adaptations are conducted interval by interval in descending order $(\bar{\phi_1}^i, \ldots, \bar{\phi_k}^i)$. The three sets are $A^i = \{\bar{\phi_1}^i\}$, $R^i = \{\bar{\phi_2}^i, \cdots, \bar{\phi_k}^i\}$, $F^i = \emptyset$, meaning that $\bar{\phi_1}^i$ is the adaptation interval while the others are fine-tuning intervals. Because no interval has been quantized, there is no frozen interval. For the set $A^i$, we apply L1 or L2 norm regularization to adapt the weights towards the center:

$$L_1^i = \sum \left\| w^i - c_j^i \right\|_1, w^i \in A^i \tag{4}$$

Moreover, we accumulate all the norm values of each layer to get the weight loss with $L_1^i$ or $L_2^i$. In our work, we mainly adopt $L_1^i$; this is because the L1 norm will result in sparser

**Algorithm 1** Training a $L$-layer network with $k$ cluster center weights Multiple Phase Adaptations Quantization

**Require:** Dataset inputs $X$ and target $Y$, the original weights $w$

{1. Division}

**for** $i = 1$ to $L$ **do**

  $C^i = kmeans(w^i)$ or other methods

  Get $\phi_j^i$ according to Eq. (1), (2)

  $\bar{c_j}^i = sort(|C^i|)$, its interval range changes to $\bar{\phi_j}^i$

  $A_0^i = \emptyset$, $R_0^i = w^i$, $F_0^i = \emptyset$

**end for**

**for** $j = 1$ to $k$ **do**

  $A_j^i = \{\bar{\phi_j}^i\}$, $R_j^i = w^i - A_{j-1}^i$, $F_j^i = F_{j-1}^i + A_{j-1}^i$

  **while** $s <= 1$ **or not** reach the required iteration **do**

    Get minibatch of inputs $x$ and target $y$ from $X, Y$

    **for** $i = 1$ to $L$ **do**

    {2.1 Multiple Phase Adaptations}

      Get range $A_r^i$ by Eq. (5), (6)

      Get quantized weights $w_a^i$ by Eq. (7)

      Get freeze mask $M_F^i$ by Eq. (8)

      Get $L_1^i$ regularization by Eq. (4)

    **end for**

    {2.2 Fine-tuning}

    Compute loss function $E(w)$ by Eq. (12)

    Update $w$ by Eq. (14)

    Apply constraint to weights by Eq. (15)

  **end while**

**end for**

weights, with the result that the parameters have an optimal value towards the center. Another benefit is that the L1 norm requires less in the way of computing resources.

Using regularization will make most weights locate around the center of the interval, but will not necessarily quantize them to the exact center. Thus, the weight assignment is necessary, which is traditionally conducted after regularization or fine-tuning, which leads to large losses following weight quantization; by contrast, Multiple Phase Adaptations offers a smoother way to quantize weights during the phase. We set a parameter $s$ to express the weight quantization process. $s = 0$ means that no weights are quantized to the center, while $s = 1$ indicates that all weights are assigned to the center; in other words, the quantization for this interval has been completed. In this paper, we use relative distance to define $s$, which we call the adaptation coefficient. We first find the distance between the center and end-points of the interval, then compute the left and right end-point distance $e_l$ and $e_r$ of the adaptation range by Eq. (5). An alternative definition of $s$ is the percentage of weights that have been quantized to the center. However, this definition is relatively complicated, as we need to sort the weights in each adaptations interval to find the endpoints. Accordingly, we can compute the range of $A_r^i$ by Eq. (6). As for the quantization operation, the weights in $A_r^i$ are assigned to the corresponding center value. This process is shown in Eq. (7).

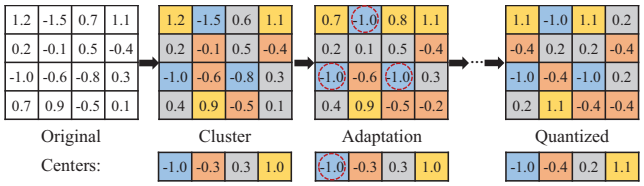



Figure 3: A toy example illustrating how adaptation operates on a layer with 16 weights. We first cluster the weights into four parts labeled with four colors. We then begin to quantize the blue part and re-train the other parts. (Note that weights may change the part that they used to belong to after the weight update.) Weight (4,4) is an example that changes its interval from gray to orange. Finally, we repeat the above operation until all weights are quantized.

$$e_l = s\left|c^i - e_j^i\right|, e_r = s\left|c^i - e_{j+1}^i\right| \tag{5}$$

$$A_r^i = \left[c^i - e_l, c^i + e_r\right] \tag{6}$$

$$w_a^i = \begin{cases} c^i, & if\ w^i \in A_r^i \\ w^i, & if\ w^i \notin A_r^i \end{cases} \tag{7}$$

In Eq. (5), $e_j^i$ and $e_{j+1}^i$ are two endpoints of the adaptation interval that meets the condition $e_j^i < e_{j+1}^i$. As the adaptation progresses, $s$ grows slowly. In practice, $s$ needs about two epochs to increase to 1.

Once the weights are quantized to the appropriate interval's center, they will never change again, a state we refer to as "frozen". This happens in both set $A_r^i$ and $F^i$. The difference between them is that $A_r^i$ is dynamic while $F^i$ is static in one interval. To handle this problem, we use a mask matrix $M_F^i$ to represent the $i^{th}$ layer's frozen status, which indicates whether the weights are frozen. $M_F^i$ is defined as:

$$M_F^i = \begin{cases} 0, if\ w^i \in \left\{F^i, A_r^i\right\} \\ 1, if\ w^i \notin \left\{F^i, A_r^i\right\} \end{cases} \tag{8}$$

As the Multiple Phase Adaptations process continues, the $A_r^i$ will finally equal $A^i$. When the set $A^i$ is totally frozen, it will be merged into $F^i$.

## Division

Division involves splitting weights into different intervals. For different tasks, there might be various ways to conduct this procedure. In our work, k-means is the first of these methods selected, because it is suitable for large samples; moreover, its criterion, minimizing within-cluster sum-of-squares, is similar to ours. The k-means aims to divide $n$ original weights in set $W = \{w_1, w_2, \ldots, w_n\}$ into $k$ disjoint clusters $C = \{c_1, c_2, \ldots, c_k\}$. The criterion is shown in Eq. (9). To decrease the clustering time, we here opt to use random sampling. This will also help to get the distribution of weights quickly and avoid the impact of outliers. After one interval is quantized, we can also update the other centers by k-means again.

$$\arg\min_C \sum_{l=1}^{k} \sum_{w \in c_l} \|w - c_l\|^2 \tag{9}$$

In addition to k-means, quantization with linear and exponential weight centers are also two possible solutions. First, take the average of the absolute of the minimum and maximum value, as the maximum of the centers. For linear quantization, we evenly sample centers between 0 and the max center value with $k/2$. For exponential quantization, we start from the max center value, then sample the centers by taking a half of the previous center value until it approaches to zero. Finally, the opposites of the sampled centers are also used as quantization centers. A comparison of the different ways used divide weights is presented in Figure 2.

Because the weights are stored in binary, it is common to make $k$ an integer power of two. Another key point is that the distribution of weights may vary greatly from layer to layer; thus, we need to conduct division for each layer. For the $i^{th}$ layer, the division set is shown as Eq. (10), where $c_j^i$ represents the $j^{th}$ center of $C^i$ and the centers are arranged in descending order.

$$C^i = \left\{c_1^i, c_2^i, \cdots, c_k^i\right\}, i = 1, 2, \cdots, L \tag{10}$$

It is next necessary to determine the order for intervals to conduct Multiple Phase Adaptations. In (Han et al. 2015), the authors simply set a threshold to prune the small-weight connections. Inspired by this approach, we take the absolute values of center $C^i$ and sort them again as $\bar{C^i}$ in Eq. (11). We begin to conduct adaptation in the $\bar{c_1}^i$ interval, which represents the most important weights in the layer. After sorting, the corresponding range of $\bar{c_j}^i$ also changes to $\bar{\phi_j}^i$, which meets the condition of $\bar{c_j}^i \in \bar{\phi_j}^i$.

$$\begin{aligned} \bar{C^i} =& sort\left(\left|C^i\right|\right) = sort\left(\left\{\left|c_1^i\right|, \left|c_2^i\right|, \cdots, \left|c_k^i\right|\right\}\right) \\ &= \left\{\bar{c_1}^i, \bar{c_2}^i, \cdots, \bar{c_j}^i, \cdots, \bar{c_k}^i\right\} \\ s.t.\ \ & \bar{c_1}^i > \bar{c_2}^i > \cdots > \bar{c_j}^i > \cdots > \bar{c_k}^i \end{aligned} \tag{11}$$

## Fine-tuning

In this paper, the fine-tuning and Multiple Phase Adaptations are performed simultaneously. Thus, the loss function is the sum of two parts, as Eq. (12) shows: one is for accuracy, while the other is for compression. Although the weights are clustered to different intervals, fine-tuning may change which interval they belong to; however, they cannot slip into the adaptation or frozen intervals because of the constraint. In Figure 3, some weights change color due to this.

$$\min\ E(w) = L(w; x, y) + \sum_{i=1}^{n} \lambda^i L_1^i(w) \tag{12}$$

In Eq. (12), $L(w; x, y)$ is the basic classification loss of the network, while $x, y$ denote the image input and target. $L_1^i(w)$ is the L1 norm loss for weights of the $i^{th}$ layer, and $\lambda^i$ is a factor of the $i^{th}$ layer that controls the importance of the layer's weight loss. Recently, models are going deeper and deeper, making the loss harder to propagate back to the shallower layers; thus, we can set a larger $\lambda^i$ for them. In this work, in addition to the basic weight loss $\lambda_0$, a weight loss decay factor $d_0$ in Eq. (13), is also adopted. For the optimization, we simply use the RMSprop algorithm, although others can also be applied. In order to freeze the quantized weights, $M_F^i$ is the gate used to control whether or not the weights conduct gradient propagation. The whole update process is shown in Eq. (14), where $\gamma$ is the learning rate.

$$\lambda^i = \lambda_0 (d_0)^i, 0 < d_0 \leq 1 \tag{13}$$

$$w \leftarrow w - \gamma \frac{\partial E}{\partial w} M_F^i \tag{14}$$

While fine-tuning, the value of weights may slip into the range of adaptations or frozen intervals, which will pollute the quantized weights. It is therefore necessary to implement a constraint for $w \in R$.

$$w_c = \begin{cases} w_{min}, & if\ w < w_{min} \\ w, & if\ w_{min} \leq w \leq w_{max} \\ w_{max}, & if\ w > w_{max} \end{cases} \tag{15}$$

where $w_{min}, w_{max}$ are the min and max value of set $R$. After fine-tuning, the distribution of weights will also change, meaning that the original center value cannot represent the new model. We may re-cluster the weights to get a better performance. This time, we only need to cluster the unquantized parts to update $\bar{c_j}^i$. Accordingly, in Figure 3, the centers also change after the update. The complete operations of MPA are summarized in Algorithm 1.

# Experiments

In this section, we evaluate our proposed Multiple Phase Adaptations Quantization method on the CIFAR-10 and ImageNet (ILSVRC-2012) classification benchmarks. CIFAR-10 consists of 10 classes with 6,000 images per class. Each image is a $32 \times 32$ color picture. There are 50,000 training images and 10,000 test images. As for ImageNet, which is widely adopted in the computer vision field, contains more than 120K images divided into 1,000 classes. Before using these images as input, some image preprocessing needs to be conducted. As in most experiments, we first normalize the images using the given mean and standard deviation value, then randomly crop images to $224 \times 224$ for VGG-16 and ResNet and $227 \times 227$ for AlexNet. Random image horizontal flip is adopted at the same time. Experiments are conducted on common CNN models, including AlexNet, VGG-16 and ResNet. The deep learning framework that we use is PyTorch.

## Implementation Details

First, we use the ResNet-18 in order to conduct quantization on CIFAR-10 to prove the effectiveness of our method. The CIFAR-10 original model is trained for 200 epochs and achieves 93.11% accuracy. After 16 epochs re-training with MPA, we convert the trained full-precision model to a 4-bit or 3-bit low-precision model. We then use ImageNet as the dataset. As for training epochs, we adopt two training strategies: a shorter one (in which 32 epochs are used to quantize weights) and a longer one (in which 64 epochs are used). AlexNet is one of the basic CNN structures. We use GPU

Table 1: Quantization accuracy results with short epoch training on ImageNet (strategy A)

| Network | Bit-width | Top-1 Acc (%) | Top-5 Acc (%) | Change in top-1/top-5 error |
|---|---|---|---|---|
| AlexNet | 32 | 56.5 | 79.0 | -/- |
| | 3 | 54.2 | 77.6 | -2.3/-1.4 |
| | 4 | 55.8 | 78.4 | -0.7/-0.6 |
| VGG-16 | 32 | 71.6 | 90.4 | -/- |
| | 3 | 70.8 | 89.9 | -0.8/-0.5 |
| | 4 | 71.4 | 90.3 | -0.2/-0.1 |
| ResNet18 | 32 | 69.8 | 89.0 | -/- |
| | Ternary | 67.2 | 87.4 | -2.6/-1.6 |
| | 3 | 68.9 | 88.0 | -0.9/-1.0 |
| | 4 | 69.2 | 88.3 | -0.6 /-0.7 |
| ResNet50 | 32 | 76.1 | 92.8 | -/- |
| | 3 | 74.8 | 92.2 | -1.3/-0.6 |
| | 4 | 75.9 | 92.7 | -0.2/-0.1 |

Table 2: Quantization accuracy results with long epoch training on ImageNet (strategy B)

| Network | Bit-width | Top-1 Acc (%) | Top-5 Acc (%) | Change in top-1/top-5 error |
|---|---|---|---|---|
| AlexNet | 32 | 56.5 | 79.0 | -/- |
| | 3 | 55.6 | 78.5 | -0.9/-0.5 |
| | 4 | 56.2 | 78.8 | -0.3/-0.2 |
| VGG-16 | 32 | 71.6 | 90.4 | -/- |
| | 3 | 71.4 | 90.0 | -0.2/-0.4 |
| | 4 | 71.7 | 90.7 | +0.1/+0.3 |
| ResNet18 | 32 | 69.8 | 89.0 | -/- |
| | Ternary | 69.2 | 88.5 | -0.6/-0.5 |
| | 3 | 69.7 | 88.8 | -0.1/-0.2 |
| | 4 | 70.2 | 89.3 | +0.4/+0.3 |
| ResNet50 | 32 | 76.1 | 92.8 | -/- |
| | 3 | 75.1 | 92.4 | -1.0/-0.4 |
| | 4 | 76.0 | 92.8 | -0.1/0.0 |

cards for the experiments with a batch size of 256. The RMSprop optimizer is adopted with an initial learning rate of $0.005$ and a momentum of $0.5$. Taking a bit-width of 4 as an example, the number epochs required to quantize one interval for the shorter and longer training strategies are 2 and 4 respectively. We set $\lambda_0 = 0.01$ and $d_0 = 0.95$. VGG-16 is similar to AlexNet, except that the number of convolutional layers increases to 13. Quantization is conducted with a learning rate of $0.001$ and other parameters unchanged. ResNet is widely used for feature extraction in many computer vision tasks. The main settings are the same overall; however, it is worth noting that we consider one block structure a whole part and those layers in one block share the same $\lambda^i$.

## Results Analysis

Results of all the ImageNet experiments are presented in Tables 1 and 2. AlexNet is a baseline test. In Table 2, top-1/top-5 accuracy decreases 0.3%/0.2%, 0.9%/0.5% with bit-widths of 4 and 3 respectively. Intuitively, the more we retrain, the better results we will get. Our experiments also prove this point.

For VGG-16, results for bit-widths of 4 and 3 exhibit a change of +0.1%/+0.3%, -0.2%/-0.4% in terms of top-1/top-5 accuracy respectively, while 3 bit-width quantization also maintains similar levels of accuracy. We find that the quantization model of VGG-16 results in accuracy improvement. This finding demonstrates that this kind of quantization does little harm to the original models so that fine-tuning can increase the accuracy. Another possible reason is that the weights become sparse, which improves the generalization performance for the validation image set.

The results show that our method is valid for residual structure. Moreover, ResNet has fewer weight parameters than VGG-16, but it achieves a better result. Thus, the number of weight parameters is not the determining factor of performance.

## Comparison with State-of-the-art Methods

Table 3 presents a comparison of this work with other state-of-the-art network quantization algorithms. In the interests of fairness, we compare the change of accuracy between Top-1 and Top-5. For some of the models and bit-widths experimental data is not available in the original paper. Accordingly, to facilitate comparison with our method, we use LQ-Net code to implement quantization. "A" indicates the short training epochs, while "B" denotes the long training epochs. Moreover, we only compare the results in the same

Table 3: Model comparison in Top-1/5 on ImageNet

| Network | Method | 3-bit (%) | 4-bit (%) |
|---|---|---|---|
| AlexNet | LogQuant | -/- | -/-1.4 |
| | LinearQuant | -/- | - /-0.7 |
| | LQ-Net-A | -6.8/-4.5 | -4.0/-3.7 |
| | LQ-Net-B | -1.6/-1.0 | -0.6/-0.3 |
| | **Ours-A** | -2.3/-1.4 | -0.7/-0.6 |
| | **Ours-B** | **-0.9/-0.5** | **-0.3/-0.2** |
| VGG-16 | LogQuant | -/-0.6 | -/0.0 |
| | LinearQuant | -/-6.8 | -/-0.4 |
| | SLQ | -0.2/-1.1 | **+2.6/+0.6** |
| | LQ-Net-A | -7.2/-5.1 | -4.2/-2.5 |
| | LQ-Net-B | -0.3/-0.5 | -0.1/+0.2 |
| | **Ours-A** | -0.8/-0.5 | -0.2/-0.1 |
| | **Ours-B** | **-0.2/-0.4** | +0.1/+0.3 |
| ResNet-18 | ABC-Net | -3.1/-2.5 | -/- |
| | INQ | -0.2/-0.3 | **+0.6**/+0.3 |
| | QIL | -0.3/-0.3 | +0.1/-0.1 |
| | LQ-Net-A | -6.2/-3.7 | -3.5/-2.4 |
| | LQ-Net-B | -0.4/-0.9 | -0.5/-0.3 |
| | LQ-Net | -0.3/-0.7 | -0.3/-0.4 |
| | **Ours-A** | -0.9/-1.0 | -0.6/-0.7 |
| | **Ours-B** | **-0.1/-0.2** | +0.4/**+0.3** |
| ResNet-50 | WEQ | -5.0/-2.2 | -2.0/-0.3 |
| | LQ-Net-A | -7.7/-4.5 | -2.3/-1.6 |
| | LQ-Net-B | -1.2/-0.7 | -0.3/-0.2 |
| | LQ-Net | -/- | **0.0**/-0.1 |
| | **Ours-A** | -1.3/-0.6 | -0.2/-0.1 |
| | **Ours-B** | **-1.0/-0.4** | -0.1/**0.0** |

bit-width.

For AlexNet, we obtain the best results. In particular, when utilizing strategy A and a bit-width of 4, we achieve much better accuracy than LQ-Net. This indicates that our method does not require much time to get an acceptable low bit-width model, which is also observed in the following experiments. For VGG-16, although we have the best result at 3 bit-width, our approach is outperformed by SLQ at a bit-width of 4; this is because we only use single layer information. As for ResNet-18, we use ternary weights to compare with INQ and ELQ. In Table 4, the symbol $\Delta$ means the decrease in Top-1 or 5. Our performance is better than ELQ, which is the state-of-art method, and we also use fewer hyperparameters (only universal parameters $\lambda_0$ and $d_0$). ResNet-50 is the most complex model, so a few methods have been tested on this with particularly low bit-width weights. Our models experience only -1.0%/-0,4%, -0.1%/0.0% accuracy loss using 3/4 widths on this model.

The reason why our approach achieves an advantage in accuracy is that only a portion of the weights are needed to be quantized in each interval. Thus, the model suffers only minimally from quantization, and will soon fine-tuned to the original accuracy or even better. In general, we meet the goal of low precision and high accuracy; moreover, in some aspects, we achieve the best overall results.

Table 4: Comparison of ternary ResNet-18 on ImageNet

| Method | Top-1(%) | Top-5(%) | $\Delta$ Top-1/5(%) |
|---|---|---|---|
| INQ | 66.0 | 87.1 | -2.3/-1.6 |
| ELQ | 67.5 | 88.1 | -0.8/-0.6 |
| Ours | 69.2 | 88.5 | **-0.6/-0.5** |

Table 5: Ablation experiments on ResNet-18

| Network | Method | Top-1 (%) | $\Delta$ (%) |
|---|---|---|---|
| CIFAR-10 | Original | 93.1 | - |
| | 3-bit | 91.6 | -1.5 |
| | 3-bit with MPA | 93.1 | 0.0 |
| | 4-bit | 92.3 | -0.8 |
| | 4-bit with MPA | 93.4 | +0.3 |
| ImageNet | Original | 69.8 | - |
| | 3-bit | 66.9 | -2.9 |
| | 3-bit with MPA | 69.7 | -0.1 |
| | 4-bit | 68.6 | -1.2 |
| | 4-bit with MPA | 70.2 | +0.3 |

Table 6: Comparison of division methods on ImageNet

| Bit-width | Method | Top-1 (%) | Top-5 (%) |
|---|---|---|---|
| 32 | ref | 69.8 | 89.0 |
| 3 | C | 69.7 | 88.8 |
| | L | 67.6 | 87.3 |
| | E | 69.5 | 88.8 |
| 4 | C | 70.2 | 89.3 |
| | L | 68.9 | 88.0 |
| | E | 69.7 | 88.9 |

## Ablation Experiments

We use ResNet-18 to conduct quantization on CIFAR-10 and ImageNet in order to prove the effectiveness of our method. Methods without MPA mean that $s = 1$ when quantization is being conducted. The results of ablation experiments results on two datasets are presented in Table 5. We can see from the results that an accuracy increase is achieved for both bit-widths compared with the naive method.

## Division Method Comparison

During the inference, linear and exponential centers can be beneficial for acceleration. For example, if we use exponential weight centers, we can use a shift operation instead of multiplication. Thus, it is advisable to quantize models with these division methods. Results for linear and exponential quantization are also shown in Table 6. "C" is the original cluster method, while "L" means linear and "E" denotes an exponential method of conducting division. We find that there is little accuracy drop for the exponential quantization. Different ways of determining the intervals may have different weight histograms. Both clustering and exponential methods have more centers near zero; as a result, the exponential centers can balance compression and acceleration.

## Conclusions

In this paper, a novel model quantization framework (MPA) is proposed for model compression, which includes division, Multiple Phase Adaptations, and fine-tuning. Through using these steps, our method can minimize damage to the model and compensate for more accuracy loss during fine-tuning. Moreover, MPA does not introduce any model-or bit-width-specific hyperparameter, meaning that we can apply MPA on various network models. Due to these well-designed phases and operations, we obtain state-of-the-art-level ternary, 3- and 4-bit-width results on different models without substantial accuracy loss. In addition, linear and exponential center experiments are also conducted, which indicates the potential of network acceleration. In our future work, we will apply Multiple Phase Adaptations from CNN to RNN or LSTM models. Furthermore, our quantized CNN models will be migrated to resource-limited devices to investigate computation and power efficiency.

## Acknowledgments

This work is partially supported by grants from the National Key R&D Program of China under grant 2017YFB1002400, the National Natural Science Foundation of China under contract No. U1611461 and No. 61825101, and Program for Guangdong Introducing Innovative and Entrepreneurial Teams 2017ZT07X183.

## References

Courbariaux, M.; Hubara, I.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*.

Denil, M.; Shakibi, B.; Dinh, L.; Ranzato, M.; and De Freitas, N. 2013. Predicting parameters in deep learning. In *NIPS*, 2148–2156.

Girshick, R.; Donahue, J.; Darrell, T.; and Malik, J. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 580–587.

Gong, Y.; Liu, L.; Yang, M.; and Bourdev, L. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *NIPS*, 1135–1143.

Han, S.; Mao, H.; and Dally, W. J. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Hassibi, B., and Stork, D. G. 1993. Second order derivatives for network pruning: Optimal brain surgeon. In *NIPS*, 164–171.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*, 1097–1105.

Lebedev, V.; Ganin, Y.; Rakhuba, M.; Oseledets, I.; and Lempitsky, V. 2014. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*.

LeCun, Y.; Denker, J. S.; and Solla, S. A. 1990. Optimal brain damage. In *NIPS*, 598–605.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

Long, J.; Shelhamer, E.; and Darrell, T. 2015. Fully convolutional networks for semantic segmentation. In *CVPR*, 3431–3440.

Luo, P.; Zhu, Z.; Liu, Z.; Wang, X.; and Tang, X. 2016. Face model compression by distilling knowledge from neurons. In *AAAI*, 3560–3566.

Miyashita, D.; Lee, E. H.; and Murmann, B. 2016. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 525–542. Springer.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Tung, F., and Mori, G. 2018. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *CVPR*, 7873–7882.

Ullrich, K.; Meeds, E.; and Welling, M. 2017. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*.

Xu, Y.; Wang, Y.; Zhou, A.; Lin, W.; and Xiong, H. 2018. Deep neural network compression with single and multiple level quantization. *arXiv preprint arXiv:1803.03289*.

Yu, X.; Liu, T.; Wang, X.; and Tao, D. 2017. On compressing deep models by low rank and sparse decomposition. In *CVPR*, 7370–7379.

Zhang, X.; Zhou, X.; Lin, M.; and Sun, J. 2017. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*.

Zhang, D.; Yang, J.; Ye, D.; and Hua, G. 2018. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *ECCV*, 365–382.

Zhou, A.; Yao, A.; Guo, Y.; Xu, L.; and Chen, Y. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*.

Zhou, A.; Yao, A.; Wang, K.; and Chen, Y. 2018. Explicit loss-error-aware quantization for low-bit deep neural networks. In *CVPR*, 9426–9435.