Quantization Networks

Jiwei Yang[1,2]*, Xu Shen[2], Jun Xing[3], Xinmei Tian[1]†, Houqiang Li[1],
Bing Deng[2], Jianqiang Huang[2], Xian-sheng Hua[2]‡
[1] Department of Electronic Engineering and Information Science,
University of Science and Technology of China
[2]Damo Academy, Alibaba Group
[3]Institute for Creative Technologies, University of Southern California
yjiwei@mail.ustc.edu.cn, junxnui@gmail.com, {xinmei,lihq}@ustc.edu.cn,
{shenxu.sx,dengbing.db,jianqiang.hjq,xiansheng.hxs}@alibaba-inc.com

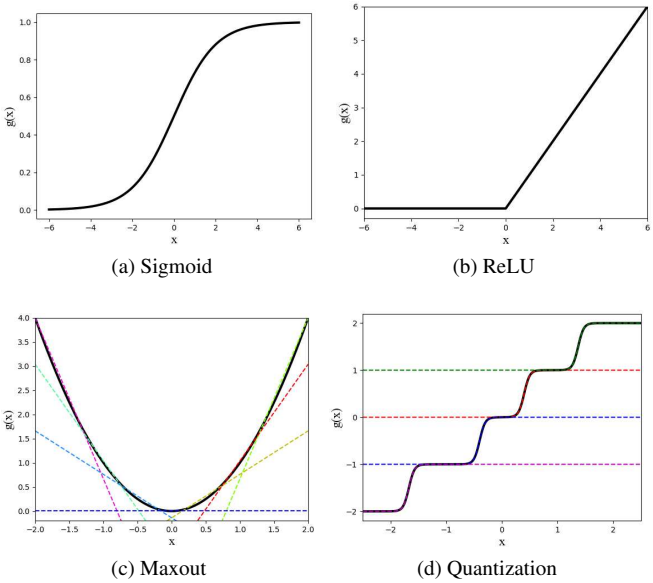
Abstract

Although deep neural networks are highly effective, their high computational and memory costs severely hinder their applications to portable devices. As a consequence, low-bit quantization, which converts a full-precision neural network into a low-bitwidth integer version, has been an active and promising research topic. Existing methods formulate the low-bit quantization of networks as an approximation or optimization problem. Approximation-based methods confront the gradient mismatch problem, while optimization-based methods are only suitable for quantizing weights and can introduce high computational cost during the training stage. In this paper, we provide a simple and uniform way for weights and activations quantization by formulating it as a differentiable non-linear function. The quantization function is represented as a linear combination of several Sigmoid functions with learnable biases and scales that could be learned in a lossless and end-to-end manner via continuous relaxation of the steepness of Sigmoid functions. Extensive experiments on image classification and object detection tasks show that our quantization networks outperform state-of-the-art methods. We believe that the proposed method will shed new lights on the interpretation of neural network quantization.

1. Introduction

Although deep neural networks (DNNs) have achieved huge success in various domains, their high computational and memory costs prohibit their deployment in scenarios where both computational and storage resources are limited. Thus, the democratization of deep learning hinges on the development of efficient DNNs. Various techniques have been proposed to lighten DNNs by either reducing the number of weights and connections or by quantizing the weights and activations to lower bits. As exemplified by ResNet [8], SqueezeNet [13] and MobileNet [11], numerous efforts have been devoted to designing networks with compact layers and architectures. Once trained, these networks can be further compressed with techniques such as network pruning [7], weight sharing [3] or matrix factorization [16].

Approaches for quantizing full-precision networks into low-bit networks can be roughly divided into two cate-



(a) Sigmoid (b) ReLU

(c) Maxout (d) Quantization

Figure 1: Non-linear functions used in neural networks.

*This work was done when the author was visiting Alibaba as a research intern.
†Corresponding author.
‡Corresponding author.

gories: approximation-based and optimization-based approaches. Methods in the first category approximate the full-precision (32-bit) values with discrete low-bit (*e.g.*, binary) values via step functions in the forward pass [27, 30, 33, 19, 34, 15, 21, 22, 1]. Because the gradients of such approximations are saturated, additional approximations in the backward process are needed. As a consequence, the use of different forward and backward approximations results in a gradient mismatch problem, which makes the optimization unstable. To avoid the approximation of gradients, some methods formulate the quantization of neural networks as a discretely constrained optimization problem, where the losses of the networks are incorporated [20, 10]. Unfortunately, optimization-based methods are only suitable for the quantization of weights. Moreover, the iterative solution of the optimization problem suffers from a high computational complexity during training.

Intuitively, if we can formulate the quantization operation as a simple non-linear function similar to common activation functions (*e.g.*, Sigmoid [17], ReLU [25] and Maxout [6]), no approximation of gradients would be needed, and the quantization of any learnable parameters in the DNNs, including activations and weights, can be learned straightforwardly and efficiently. Inspired by this, we present a novel perspective for interpreting and implementing quantization in neural networks. Specifically, we formulate quantization as a differentiable non-linear mapping function, termed the quantization function. As shown in Fig. 1, the quantization function is formed as a linear combination of several Sigmoid functions with learnable biases and scales. In this way, the proposed quantization function can be learned in a lossless and end-to-end manner and works for any weights and activations in neural networks, thereby avoiding the gradient mismatch problem. As illustrated in Fig. 2, the quantization function can be trained via continuous relaxation of the steepness of the Sigmoid functions.

Our main contributions are summarized as follows:

- In contrast to existing low-bit quantization methods, we are the first to formulate quantization as a differentiable non-linear mapping function, thereby providing a *simple/straightforward* and *general/uniform* solution for any-bit weight and activation quantization without suffering the severe gradient mismatch problem.
- We implement a simple and effective form of quantization networks that can be learned in a lossless and end-to-end manner and that outperforms state-of-the-art quantization methods on both image classification and object detection tasks.

2. Related Work

In this paper, we propose formulating the quantization operation as a differentiable non-linear function. In this section, we give a brief review of both low-bit quantization methods and non-linear functions used in neural networks.

2.1. Low-Bit Quantization of Neural Networks

Approaches for quantizing full-precision networks into low-bit networks can be roughly divided into two categories: approximation-based and optimization-based approaches. The first approach is to approximate the 32-bit full-precision values with discrete low-bit values in the forward pass of the networks. BinaryConnect [4] directly optimizes the loss of the network with weights W replaced by sign(W), and it approximates the sign function with the "hard tanh" function in the backward process to avoid the zero-gradient problem. The binary weight network (BWN) [27] adds scale factors for the weights during binarization. Ternary weight network (TWN) [21] introduces ternary weights and achieves improved performance. Trained ternary quantization (TTQ) [34] proposes learning both ternary values and scaled gradients for 32-bit weights. DoReFa-Net [33] proposes quantizing 32-bit weights, activations and gradients using different bit widths. Gradients are approximated by a customized form based on the mean of the absolute values of the full-precision weights. In [30], weights, activations, gradients and errors are all approximated by low-bitwidth integers based on rounding and shifting operations. Jacob et al. [15] propose an affine mapping of integers to real numbers that allows inference to be performed using integer-only arithmetic. As discussed before, approximation-based methods use different forward and backward approximations, which causes a gradient mismatch problem. Friesen and Domingos [5] observe that setting targets for hard-threshold hidden units to minimize losses is a discrete optimization problem. Zhuang et al. [35] propose a two-stage approach to quantize the weights and activations in a two-step manner. Lin et al. [23] approximate full-precision weights with a linear combination of multiple binary weight bases. Zhang et al. [31] propose a flexible non-uniform quantization method to quantize both network weights and activations. Cai et al. [1] use several piece-wise backward approximators to overcome the gradient mismatch problem. Zhou et al. [32] propose a step-by-step decoupling operation to efficiently convert a pre-trained full-precision convolutional neural network (CNN) model into a low-precision version. As a specific quantization, HashNet [2] adopts a similar continuous relaxation to train the hash function, where a single tanh function is used for binarization. However, our training case (multi-bit quantization of both activations and weights in multiple layers) is substantially more complicated and challenging.

To avoid the gradient approximation problem, optimization-based quantization methods have recently been proposed. Such methods directly formulate the quantization of neural networks as a discretely constrained

optimization problem [20, 10]. Leng et al. [20] introduce convex linear constraints for the weights and solve the problem by the alternating direction method of multipliers (ADMM). Hou and Kwok [10] directly optimize the loss function w.r.t. the ternarized weights using a proximal Newton algorithm. However, these methods are only suitable for the quantization of weights and such iterative solutions suffer from high computational costs for training.

2.2. Non-Linear Functions in Neural Networks

In neural networks, the design of hidden units is distinguished by the choice of the non-linear activation function $g(x)$ for hidden units [12]. The simplest form of a neural network is the perceptron [28], where a unit step function is introduced to produce a binary output:

$$g(x) = \mathcal{A}(x) = \begin{cases} 1 & x \geq 0, \\ 0 & x < 0. \end{cases} \tag{1}$$

This form is similar to the binary quantization operation, *i.e.*, discretize the continuous inputs into binary values. However, the problem is that it is not immediately obvious how to learn the perceptron networks [26].

To solve this problem, the sigmoid activation function is adopted in the early form of feedforward neural networks:

$$g(x) = \sigma(x) = \frac{1}{1 + exp(-x)}, \tag{2}$$

which has smooth and non-zero gradients everywhere so that the sigmoid neurons can be learned via back-propagation. When the absolute value of x is very large, the output of a sigmoid function is close to a unit step function.

Currently, rectified linear units (ReLU) are more frequently used as the activation functions in deep neural networks, and a generalization of ReLU is Maxout:

$$g(x) = \max_j (a_j * x + c_j), j = 1, \ldots, k \tag{3}$$

where $\{a_j\}$ and $\{c_j\}$ are learned parameters. The form of Maxout indicates that a complex convex function can be approximated by a combination of k simple linear functions. We adopt a similar idea to formulate the quantization function.

3. Quantization Networks

The main idea of this work is to formulate the quantization operation as a differentiable non-linear function, which can be applied to any weights and activations in deep neural networks. We first present our novel interpretation of quantization from the perspective of non-linear functions, followed by the learning of the quantization networks.

3.1. Reformulation of Quantization

The quantization operation maps continuous inputs into discrete integer numbers, and a binary quantization operation can be seen as an unit step function. Inspired by the design of Maxout units (Eq. 3), quantizing continuous values into a set of integer numbers can be formulated as a combination of several binary quantizations. That said, the ideal low-bit quantization function could be represented as a combination of several unit step functions with specified biases and scales, as shown in Fig. 2(e):

$$y = \sum_{i=1}^{n} s_i \mathcal{A}(\beta x - b_i) - o, \tag{4}$$

where x is the full-precision weight/activation to be quantized, and y is the quantized integer set $\mathcal{Y}$ (*e.g.*, $\{-4, -2, -1, 0, 1, 2, 4\}$) with $n + 1$ (*e.g.*, 7) quantization intervals. $\mathcal{A}$ is the standard unit step function, with β being the overall scale factor of inputs, s_i and b_i being the scale and bias of each unit step function. In particular, β and b_i are parameters to be learned, and s_i is calculated by $s_i = \mathcal{Y}_{i+1} - \mathcal{Y}_i$. The global offset $o = \frac{1}{2}\sum_{i=1}^{n} s_i$ keeps the quantized output czero-centered. Once the target quantization integer set $\mathcal{Y}$ is given, $n = |\mathcal{Y}| - 1$, s_i and the offset o can be directly obtained.

3.2. Training and Inference with Quantization Networks

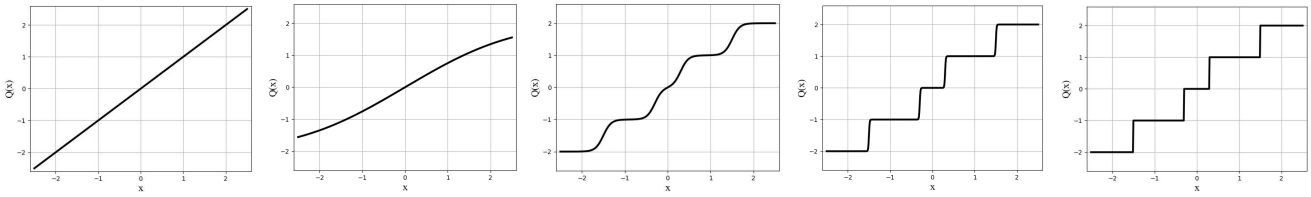
Since the ideal step function is not smooth, we adopt a continuous relaxation method to train it [2], by replacing each unit step function in the ideal quantization function (Eq. 4) with a sigmoid function. Such "soft" quantization function is differentiable (Fig. 2(c)), and can be learned in an end-to-end manner via back-propagation without suffering the gradient mismatching.

Since the ideal quantization function (Eq.4) is finally applied in the inference stage, we gradually narrow the gap between the ideal and soft quantization functions during the training stage. Motivated by the distilling idea in [9], we introduce a temperature factor T to the Sigmoid function:

$$\sigma(Tx) = \frac{1}{1 + exp(-Tx)}. \tag{5}$$

When the value of T gets larger, the gap between two quantization functions becomes smaller, but the learning capacity of the quantization networks is also lower due to the saturated gradient. Thus, during the training stage, we start with a small T to ensure a stable and effective learning, and gradually increase T w.r.t. the training epochs to finally approach the ideal quantization functions, as illustrated in Fig. 2.

Forward Propagation. For a set of full-precision weights or activations that need to be quantized $\mathcal{X} =$



(a) No Quantization (b) T=1 (c) T=11 (d) T=121 (e) Complete Quantization

Figure 2: The relaxation process of a quantization function during training, which goes from a straight line to steps as the temperature T increases.

$\{x_d, d = 1, \cdots, D\}$, the quantization function is applied to each x_d independently:

$$y_d = \mathcal{Q}(x_d) = \alpha(\sum_{i=1}^{n} s_i \sigma(T(\beta x_d - b_i)) - o), \quad (6)$$

where β and α are the scale factors of the input and output, respectively; b_i is the beginning of i-th quantization interval, and particularly, b_0 is set to $-\infty$.

Like any other non-linear activation functions, Eq. (6) provides a simple and uniform quantization operation, without introducing any changes to the original network structure.

Backward Propagation. During the training stage, we need to back-propagate the gradients of the loss ℓ through the quantization function, as well as compute the gradients with respect to the involved parameters:

$$\frac{\partial \ell}{\partial x_d} = \frac{\partial \ell}{\partial y_d} \cdot \sum_{i=1}^{n} \frac{T\beta}{\alpha s_i} g_d^i(\alpha s_i - g_d^i), \quad (7)$$

$$\frac{\partial \ell}{\partial \alpha} = \sum_{d=1}^{D} \frac{\partial \ell}{\partial y_d} \cdot \frac{1}{\alpha} y_d, \quad (8)$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{d=1}^{D} \frac{\partial \ell}{\partial y_d} \cdot \sum_{i=1}^{n} \frac{T x_d}{\alpha s_i} g_d^i(\alpha s_i - g_d^i), \quad (9)$$

$$\frac{\partial \ell}{\partial b_i} = \sum_{d=1}^{D} \frac{\partial \ell}{\partial y_d} \cdot \frac{-T}{\alpha s_i} g_d^i(\alpha s_i - g_d^i). \quad (10)$$

where $g_d^i = \sigma(T(\beta x_d - b_i))$, and the gradients of n, s_i and the offset o can be directly obtained by $\mathcal{Y}$. Our soft quantization function is a differentiable transformation that introduces quantized weights and activations into the network.

Training and Inference. To quantize a network, we specify a set of weights or activations and insert the quantization function for each of them according to Eq. (6). For example, a layer that previously receives x as an input becomes $\mathcal{Q}(x)$, and a module that previously uses W as parameters now becomes $\mathcal{Q}(W)$. Once the network has been trained, we replace the sigmoid function in Eq. (6) with the unit step function for inference:

$$y = \alpha(\sum_{i=1}^{n} s_i \mathcal{A}(\beta x - b_i) - o). \quad (11)$$

Algorithm 1 summarizes the procedure for training

Algorithm 1 Training quantization networks

Input: Network N with M modules $\mathcal{M}_{m=1}^{M}$ and their corresponding activations/inputs $\{\mathcal{X}^{(m)}\}_{m=1}^{M}$, trainable weights (or other parameters) $\{\Theta^{(m)}\}_{m=1}^{M}$, and corresponding target quantized integers set $\{\mathcal{Y}_{\mathcal{X}}^{(m)}, \mathcal{Y}_{\Theta}^{(m)}\}_{m=1}^{M}$.
Output: Quantized network for inference, N_Q^{inf}
$N_Q^{tr} \leftarrow N$ // Training quantization network
for $m \leftarrow 1$ to M **do**
  Infer $\{s_i^{(m)}, o^{(m)}\}$ and intialize $\{\alpha^{(m)}, \beta^{(m)}, b_i^{(m)}\}$ (Eq. 6) in the soft quantization function $\{\mathcal{Q}_{\mathcal{X}}^{(m)}, \mathcal{Q}_{\Theta}^{(m)}\}$ based on $\{\mathcal{Y}_{\mathcal{X}}^{(m)}, \mathcal{Y}_{\Theta}^{(m)}, \mathcal{X}^{(m)}, \Theta^{(m)}\}$.
  Apply the soft quantization function to each element x_d^m in $\mathcal{X}^{(m)}$ and each element θ_d^m in $\Theta^{(m)}$:
  $y_d^m = \mathcal{Q}^{(m)}_{\{\alpha_{\mathcal{X}}^{(m)}, \beta_{\mathcal{X}}^{(m)}, b_{\mathcal{X}}^{(m)}\}}(x_d^m)$,
  $\widehat{\theta}_d^m = \mathcal{Q}^{(m)}_{\{\alpha_{\Theta}^{(m)}, \beta_{\Theta}^{(m)}, b_{\Theta}^{(m)}\}}(\theta_d^m)$.
  Forward propagate module m with the quantized weights and activations.
end for
for $epoch \leftarrow 1$ to Max_Epochs **do**
  Train N_Q^{tr} to optimize the parameters $\Theta \cup \{\alpha_{\Theta}^{(m)}, \beta_{\Theta}^{(m)}, b_{\Theta}^{(m)}, \alpha_{\mathcal{X}}^{(m)}, \beta_{\mathcal{X}}^{(m)}, b_{\mathcal{X}}^{(m)}\}_{m=1}^{M}$ with gradually increased temperature T
end for
$N_Q^{inf} \leftarrow N_Q^{tr}$ // Inference quantization network with frozen parameters
for $m \leftarrow 1$ to M **do**
  Replace the soft quantization functions with Eq. (11) for inference.
end for

Methods \ W/A	1/32	2/32	3(±2)/32	3(±4)/32	1/1	1/2
BinaryConnect [4]	35.4/61.0	-	-	-	27.9/50.42	-
BWN [27]	56.8/79.4	-	-	-	44.2/69.2	-
DoReFa [33]	53.9/76.3	-	-	-	39.5/-	47.7/-
TWN [21]	-	54.5/76.8	-	-	-	-
TTQ [34]	-	57.5/79.7	-	-	-	-
ADMM [20]	57.0/79.7	58.2/80.6	59.2/81.8	60.0/82.2	-	-
HWGQ [1]	-	-	-	-	-	52.7/76.3
TBN [29]	-	-	-	-	-	49.7/74.2
LQ-Net [31]	-	60.5/82.7	-	-	-	**55.7/78.8**
Ours	**58.8/81.7**	**60.9/83.2**	**61.5/83.5**	**61.9/83.6**	**47.9/72.5**	55.4/**78.8**

Table 1: Top-1 and Top-5 accuracies (%) of AlexNet on ImageNet classification. The performance of the full-precision model is **61.8/83.5**. "W" and "A" represent the quantization bits of the weights and activations, respectively.

quantization networks. For a full-precision network N with M modules, where a module can be either a convolutional layer or a fully connected layer, we denote all the activations to be quantized in the m-th module as $\mathcal{X}^{(m)}$, and all the weights to be quantized in the m-th module as $\Theta^{(m)}$. All elements in $\mathcal{X}^{(m)}$ share the same quantization function parameters $\{\alpha_{\mathcal{X}}^{(m)}, \beta_{\mathcal{X}}^{(m)}, \mathbf{b}_{\mathcal{X}}^{(m)}\}$. All elements in $\Theta^{(m)}$ share the same quantization function parameters $\{\alpha_{\Theta}^{(m)}, \beta_{\Theta}^{(m)}, \mathbf{b}_{\Theta}^{(m)}\}$. We apply the quantization function module by module, and train the network with a gradually increased temperature T.

4. Experiments

4.1. Image Classification

To evaluate our method, we compare with the state-of-the-art classification methods on ImageNet dataset (ILSVRC 2012). ImageNet consists of approximately 1.2 million training images from 1,000 categories and 50,000 validation images. We evaluate our method on AlexNet [18] (over-parameterized architectures) and ResNet-18/ResNet-50 [8] (compact-parameterized architectures). We report our classification performance using Top-1 and Top-5 accuracies with networks quantized to Binary({0, 1}, 1 bit), Ternary({-1, 0, 1}, 2 bits), {-2, -1, 0, 1, 2} (denoted as 3 bits(±2)), {-4, -2, -1, 0, 1, 2, 4 } (denoted as 3 bits(±4)), and {-15, -14, ⋯, -1, 0, 1, ⋯, 14, 15 } (5 bits). All the parameters are fine-tuned from pretrained full-precision models.

All the images from ImageNet are resized to 256 pixels for the smaller edge, followed by a random crop of 224×224. Each pixel value of the input images is subtracted by the mean values and divided by the variances. Random horizontal flipping is introduced for preprocessing. No other data augmentation tricks are used in the learning process. We choose a batch size of 256 during training. Similar to [27] and [21], the parameters of the first convolutional layer and the last fully connected layer for classification are not quantized. For testing, images are preprocessed in the same way.

For our quantization function Eq. (6), to ensure all the input full-precision values lie in the linear region of our quantization function, the input scale β is initialized to $\frac{5p}{4} \times \frac{1}{q}$, where p is the max absolute value of elements in $\mathcal{Y}$ and q is the max absolute value of the elements in $\{\mathcal{X}, \Theta\}$. Here activation set $\mathcal{X}$ consists of the activations of randomly sampled 1000 samples from the dataset. The output scale α is initialized to $\frac{1}{\beta}$, which keeps the magnitude of the inputs unchanged after quantization. We adopt a layer-wise quantization in this paper, *i.e.*, the weights/activations from the same layer share the same quantization function and the weights/activations from different layers use different quantization functions.

Weight quantization: For binary quantization, only 1 sigmoid function is needed: $n = 1$, $b = 0$, $s = 2$, and $o = 1$. For the quantization of other bits, we first group the full-precision inputs into $n + 1$ clusters via k-means clustering, then rank the centers of the clusters in an ascending order, $\{c_1, \ldots, c_{n+1}\}$. The biases are initialized by $b_i = \frac{c_i + c_{i+1}}{2}$.

Activation quantization: The outputs of the ReLU units are used for the activation quantization (Conv-BN-ReLU(-Pooling)-Quant). The o in Eq. (6) is set to 0 because all activations are non-negative. For binary quantization($\{0, 1\}$), only 1 sigmoid function is needed, *i.e.*, $n = 1$ and $s = 1$. For two-bit quantization of the activations ($\{0, 1, 2, 3\}$), $n = 3$ and $s_i = 1$. We randomly take 1000 samples from the dataset, and use the min/max activation values of the output in each layer for the initialization of q. And b_i is obtained by clustering as in weight quantization with this 1000 samples.

Methods \ W/A	1/32	2/32	3(±2)/32	3(±4)/32	5/32	1/1	1/2	32/2
BWN [27]	60.8/83.0	-	-	-	-	51.2/73.2	-	-
TWN [21]	-	61.8/84.2	-	-	-	-	-	-
TTQ [34]	-	66.6/87.2	-	-	-	-	-	-
INQ [32]	-	66.0/87.1	-	68.1/88.4	69.0/89.1	-	-	-
ABC-Net [23]	-	-	-	-	68.3/87.9	42.7/67.6	-	-
HWGQ [1]	-	-	-	-	-	-	59.6/82.2	-
ADMM [20]	64.8/86.2	67.0/87.5	67.5/87.9	68.0/88.3	-	-	-	-
ICLR18 [5]	-	-	-	-	-	-	-	64.3/-
TBN [29]	-	-	-	-	-	-	55.6/79.0	-
LQ-Net [31]	-	68.0/88.0	-	69.3/88.8	-	-	62.6/84.3	-
Ours	**66.5/87.3**	**69.1/88.9**	**69.9/89.3**	**70.4/89.6**	**70.6/89.6**	**53.6/75.3**	**63.4/84.9**	**65.7/86.5**

Table 2: Top-1 and Top-5 accuracies (%) of ResNet-18 on ImageNet classification. The performance of the full-precision model are **70.3/89.5**.

Methods \ W/A	1/32	2/32	3(±2)/32	3(±4)/32	5/32
BWN [27]	68.7/-	-	-	-	-
TWN [21]	-	72.5/-	-	-	-
INQ [32]	-	-	-	-	74.8/-
LQ-Net [31]	-	75.1/92.3	-	-	-
Ours	**72.8/91.3**	**75.2/92.6**	**75.5/92.8**	**76.2/93.2**	**76.4/93.2**

Table 3: Top-1 and Top-5 accuracies (%) of ResNet-50 on ImageNet classification. The performance of the full-precision model are **76.4/93.2**.

The whole training process consists of 3 phases. Firstly, we disable the activation quantization and only train the quantization of the weights. Secondly, we fix the quantization of the weights and only train the quantization of the activations. Finally, we train the quantization of both weights and activations until the model converges. In practice, freezing $T = 1$ for the back-propagation of binary quantization achieves better performance.

AlexNet: This network consists of five convolutional layers and two fully connected layers. This network is the mostly widely used benchmark for the quantization of neural networks. As in [27, 21, 20], we use AlexNet coupled with batch normalization [14] layers. We update the model by stochastic gradient descent (SGD) with the momentum set to 0.9. The learning rate is initialized to 0.001 and decays by 0.1 at epochs 25 and 40, respectively. The model is trained for at most 55 epochs in total. The weight decay is set to $5e^{-4}$. The temperature T is set to 10 and increased linearly w.r.t. the number of training epochs, *i.e.*, $T = epoch \times 10$. The gradients are clipped with a maximum L2 norm of 5.

The results of different quantization methods are shown in Table 1, where $1/1$ means both weights and activations are binary quantized. As shown by the results, our quantization network outperforms state-of-the-art methods in both weight quantization and activation quantization.

ResNet: The most common baseline architectures, including AlexNet, VGG and GoogleNet, are all over-parameterized by design to achieve improved accuracy. Therefore, it is easy to obtain sizable compression of these architectures with a small accuracy degradation. A more meaningful benchmark would be to quantize the model architectures that already possess efficient parameters, *e.g.*, ResNet. We use the ResNet-18 and ResNet-50 networks proposed in [8] for evaluation.

The learning rate is decayed by 0.1 at epochs 30 and 45, and the model is trained for at most 55 epochs in total. The weight decay is set to $1e^{-4}$. The temperature T is set to 5 and increased linearly w.r.t the training epochs ($T = epoch \times 5$). The other settings are the same as those for AlexNet. The results of different quantization methods are shown in Table 2 and Table 3 for ResNet-18 and ResNet-50, respectively. We can see that the performance degradation of the quantized models is larger than that on AlexNet. This is reasonable because the parameters of the original model are more compact. Note that even in such

Methods \ W/A	2/32	3(±4)/32	3(±4)/8
ADMM [20]	76.2	77.6	-
Ours	**76.3**	**77.7**	**76.1**

Table 4: mAP (%) of SSD on Pascal VOC object detection. The performance of the full-precision model is **77.8**.

a compact model, our method still achieves lossless results with only 3 bits. In addition, as far as we know, we are the first to surpass the full-precision model on ResNet-18 with 3-bit weight quantization.

4.2. Object Detection

To evaluate our quantization network on object detection tasks, we test it on the popular SSD (single shot multibox detection) architecture [24]. The models are trained on the Pascal VOC 2007 and 2012 training datasets and tested on Pascal VOC 2007 test dataset. We follow the same settings in [24], and the input images are resized to 300×300. Except for the final convolutional layers with 1×1 kernels and the first convolution layer, the parameters of all other layers in the backbone VGG16 are quantized.

We update the model by SGD with the momentum set to 0.9. The initial learning rate is set to $1e^{-5}$ for the quantized parameters and $1e^{-7}$ for the non-quantized parameters and then decayed by 0.1 at epochs 70 and 90. The models are trained for 100 epochs in total. The batch size is set to 16, and the weight decay is $5e^{-4}$. We increase the temperature T by 10 every epoch, *i.e.*, $T = epoch \times 10$. The gradients are clipped with maximum L2 norm of 20.

Since other baseline quantization methods did not report their performances on object detection tasks, we only compare our model with ADMM. As shown by the results in Table 4, our model is slightly better than ADMM. This result is very promising since our method is much simpler and substantially more general than ADMM.

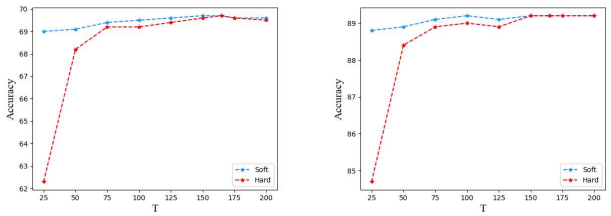
4.3. Ablation Experiments

In this section, we discuss the settings of our quantization network. All statistics are collected from the training process for Alexnet and ResNet-18 on ImageNet.

Configuration of Bias b. Usually, the quantized values are set linearly (*e.g.*, $\{-1, -\frac{k-1}{k}, \ldots, -\frac{1}{k}, 0, \frac{1}{k}, \ldots, \frac{k-1}{k}, 1\}$) or logarithmically (*e.g.*, $\{-1, -\frac{1}{2}, \ldots, -\frac{1}{2^{k-1}}, 0, \frac{1}{2^{k-1}}, \ldots, \frac{1}{2}, 1\}$) with a scale factor α [21, 30, 20, 33, 10, 15]. In this paper, we find that the distribution of the full-precision parameters of the pre-trained model roughly follows a Gaussian distribution, which indicates that quantizing weights into linear or logarithmic intervals may fail to preserve such distribution.

Quantization methods	W/A	Top-1	Top-5
linear	2/32	60.6	82.8
non-uniform	2/32	**60.9**	**83.2**
linear	3(±4)/32	60.7	83.0
non-uniform	3(±4)/32	**61.9**	**83.6**

Table 5: Ablation study concerning training the quantization of AlexNet on ImageNet classification: using linear vs. non-uniform quantization.



(a) Top-1 (b) Top-5

Figure 3: The gap between the training model and testing model along with the training process for ResNet-18 $\{-4, +4\}$. The gap between the training and testing model converges as the learning proceeds.

Thus, we adopt a non-uniform quantization (*e.g.*, K-means clustering) to counterbalance this, based on the $n + 1$ clustering centers for determining the quantization intervals $\{b_i\}$. The experimental results in Table 5 demonstrate the superior of the non-uniform quantization over the linear quantization. We also found that fixing the biases during training achieves better performance than learning an adaptive biases. Therefore, we freeze the biases after the initialization in all experiments.

Effect of Temperature. As discussed in Section 3, the temperature T controls the gap between the hard quantization function Eq. (11) in the inference stage and the soft quantization function Eq. (6) in the training stage. To investigate the effect of this gap on the performance of quantized networks, we compare the testing accuracy of the models (trained with different T) when soft and hard quantization functions are adopted, as shown in Fig. 3. We can see that as the temperature T increases, the difference between them is gradually reduced. Thus, gradually increasing the temperature T during training can achieve a good balance between model learning capacity and quantization gap.

Training from pre-trained model. In our training, the temperature parameter T is increased linearly w.r.t. the training epochs. When training from scratch, the temperature T may become quite large before the network is well-converged, and the saturated neurons will slow down the

Training methods	W/A	Top-1	Top-5
from scratch	3(±4)/32	55.3	78.8
from pre-trained	3(±4)/32	**70.4**	**89.6**

Table 6: Ablation study of training the quantization of ResNet-18 for ImageNet classification: from scratch vs. from a pre-trained model.

	Binary	Ternary	Full-precision
Time	1x	1.4x	45x
Space	1x	2x	32x

Table 7: Time-space complexity of final inference based on the VU9P FPGA evaluation. Each number indicates the ratio to the complexities of the binary network. Binary: 1-bit weights and 1-bit activations. Ternary: 2-bit weights and 2-bit activations.

network training process and cause the network to become stuck in local minima. According to Table 6, training from a pre-trained model can greatly improve the performance compared to training from scratch.

Time-space complexity of the final model for inference. Table 7 shows the time-space complexities of the final quantization networks for inference based on the VU9P FPGA evaluation. We can see that both the time and space complexities are significantly reduced resulting from the low-bit quantization of the neural networks.

Convergence of Temperature T. The training process is very stable w.r.t. different T (shown in Fig. 4). The approximation of the final "soft" quantization function to a "hard" step function is determined by the final temperature, which is controlled by the maximum training epoch ($T = epoch * 10$). The increasing speed of the temperature (*e.g.*, 10) controls the speed of convergence (or learning rate) from a "soft" to "hard" quantization (shown in Figure 4 in our paper), and it is consistent with the learning progress of the backbone model. Practically, for different backbone models, we can tune T in $\{5, 10, 20, 40\}$ based on the performance on validation set as for the learning rate for the DL models.

5. Conclusion

This work focused on interpreting and implementing the low-bit quantization of deep neural networks from the perspective of non-linear functions. Inspired by activation functions in DNNs, a soft quantization function is proposed and incorporated into deep neural networks as a new type of activation function. With this differentiable non-linear quantization function embedded, the quantization networks

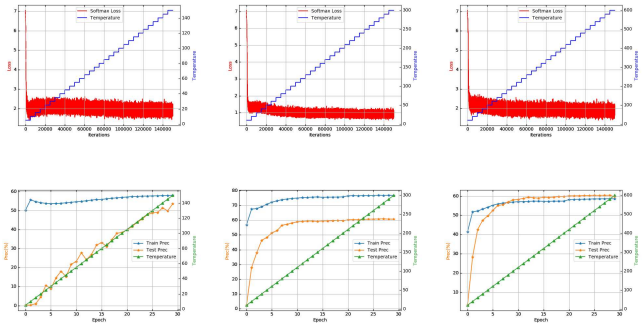


Figure 4: The training error curve and the training/validation accuracy curve for AlexNet quantization (left to right: $T = 5/10/20 * epoch$). Similar curves are observed for $T = 1/30/40 * epoch$; we do not show them here because of space limitation.

can be learned in an end-to-end manner. Our quantization method is both highly flexible and suitable for arbitrary-bit quantization and can be applied for the quantization of both weights and activations. Extensive experiments on image classification and object detection tasks have verified the effectiveness of the proposed method.

Acknowledgements

This work was supported in part by the National Key R&D Program of China under contract No. 2017YFB1002203 and NSFC No. 61872329.

References

[1] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. In *CVPR*, pages 5406–5414, 2017.

[2] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and S Yu Philip. Hashnet: Deep learning to hash by continuation. In *ICCV*, pages 5609–5618, 2017.

[3] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *ICML*, pages 2285–2294, 2015.

[4] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, pages 3123–3131, 2015.

[5] Abram L Friesen and Pedro Domingos. Deep learning as a mixed convex-combinatorial optimization problem. *arXiv preprint arXiv:1710.11573*, 2017.

[6] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C. Courville, and Yoshua Bengio. Maxout networks. In *ICML*, pages 1319–1327, 2013.

[7] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *NIPS*, pages 1135–1143, 2015.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

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

[10] Lu Hou and James T. Kwok. Loss-aware weight quantization of deep networks. In *ICLR*, 2018.

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

[12] Yoshua Bengio Ian Goodfellow and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

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

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.

[15] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, pages 2704–2713, 2018.

[16] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

[17] Joe Kilian and Hava T Siegelmann. On the power of sigmoid neural networks. In *COLT*, pages 137–143, 1993.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012.

[19] Abhisek Kundu, Kunal Banerjee, Naveen Mellempudi, Dheevatsa Mudigere, Dipankar Das, Bharat Kaul, and Pradeep Dubey. Ternary residual networks. *arXiv preprint arXiv:1707.04679*, 2017.

[20] Cong Leng, Hao Li, Shenghuo Zhu, and Rong Jin. Extremely low bit neural network: Squeeze the last bit out with admm. In *AAAI*, pages 3466–3473, 2018.

[21] Fengfu Li and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711v2*, 2016.

[22] Zefan Li, Bingbing Ni, Wenjun Zhang, Xiaokang Yang, and Wen Gao. Performance guaranteed network acceleration via high-order residual quantization. In *ICCV*, pages 2603–2611, 2017.

[23] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *NIPS*, pages 345–353, 2017.

[24] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In *ECCV*, pages 21–37, 2016.

[25] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[26] Michael A. Nielsen. Neural networks and deep learning. Determination Press, 2015.

[27] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, pages 525–542, 2016.

[28] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[29] Diwen Wan, Fumin Shen, Li Liu, Fan Zhu, Jie Qin, Ling Shao, and Heng Tao Shen. Tbn: Convolutional neural network with ternary inputs and binary weights. In *ECCV*, pages 315–332, 2018.

[30] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. In *ICLR*, 2018.

[31] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *ECCV*, pages 365–382, 2018.

[32] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. In *ICLR*, 2017.

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

[34] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. In *ICLR*, 2017.

[35] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Towards effective low-bitwidth convolutional neural networks. In *CVPR*, pages 7920–7928, 2018.