Differentiable Soft Quantization: Bridging Full-Precision and Low-Bit Neural Networks

Ruihao Gong[1,2] Xianglong Liu[1*] Shenghu Jiang[1,2] Tianxiang Li[2,3]
Peng Hu[2] Jiazhen Lin [2] Fengwei Yu[2] Junjie Yan[2]
[1]State Key Laboratory of Software Development Environment, Beihang University
[2]SenseTime Group Limited [3]Beijing Institute of Technology
{gongruihao, xlliu, jiangshenghu}@nlsde.buaa.edu.cn
{litianxiang, hupeng, linjiazhen, yufengwei, yanjunjie}@sensetime.com

Abstract

Hardware-friendly network quantization (e.g., binary/uniform quantization) can efficiently accelerate the inference and meanwhile reduce memory consumption of the deep neural networks, which is crucial for model deployment on resource-limited devices like mobile phones. However, due to the discreteness of low-bit quantization, existing quantization methods often face the unstable training process and severe performance degradation. To address this problem, in this paper we propose Differentiable Soft Quantization (DSQ) to bridge the gap between the full-precision and low-bit networks. DSQ can automatically evolve during training to gradually approximate the standard quantization. Owing to its differentiable property, DSQ can help pursue the accurate gradients in backward propagation, and reduce the quantization loss in forward process with an appropriate clipping range. Extensive experiments over several popular network structures show that training low-bit neural networks with DSQ can consistently outperform state-of-the-art quantization methods. Besides, our first efficient implementation for deploying 2 to 4-bit DSQ on devices with ARM architecture achieves up to 1.7× speed up, compared with the open-source 8-bit high-performance inference framework NCNN [31].

1. Introduction

Deep convolution neural networks have achieved great success in many fields, such as computer vision, natural language processing, information retrieval, etc. However, the expensive memory and computation costs seriously impede their deployment on the widespread resource-limited devices, especially for real-time applications. To address this problem, the quantization technique has emerged as a promising network compression solution and achieved substantial progress in recent years. It can largely reduce the network storage and meanwhile accelerate the inference speed using different types of quantizers, mainly including binary/ternary [8, 9, 16, 23, 45, 25, 33], uniform [46, 28, 6, 26, 43, 18, 36, 42, 20, 7, 2, 39, 21] and non-uniform [44, 35, 13, 5, 41, 30, 38, 37, 3].

Limited by the specific hardware features like the instruction sets, most quantization methods can hardly accomplish the goal of network acceleration and may still heavily depend on the special hardware design and long-term hardware development. For example, a special inference engine EIE [12] has been developed to speed up the method in [13]. Fortunately, the recent studies have proved that both the binary and uniform quantization models enjoy the hardware-friendly property [18, 17, 11, 27], which enables us to accelerate the inference directly on off-the-shelf hardware with the efficient bit operation or integer-only arithmetic.

Despite the attractive benefits, when quantizing into extremely low bit, existing binary and uniform quantization models still face the severe performance degradation, due to the limited and discrete quantization levels [6]. First, based on the discrete quantized representation, the backward propagation can hardly access the accurate gradients, and thus has to resort to the appropriate approximation. In the literature, straight through estimation (STE) [4] has been widely used for approximation. But it ignores the influence of quantization, and when the data is quantized to extremely low bit, its error will be amplified, causing an obvious instability of optimization. Experiments and analysis in [26, 8, 24] have shown that the gradient error caused by quantization and STE greatly harms the accuracy of quantized models.

Besides, the quantization itself inevitably brings large

*corresponding author

deviations between the original data and their quantization values, and thus often causes the performance decrease. In practice, the quantization is usually completed by two operations: clipping and rounding. The former confines data to a smaller range, while the latter maps the original value to its nearest quantization point. Both operations contribute to the quantization loss. Therefore, to alleviate the performance degradation, it is also important to find an appropriate clipping range and make a balance between clipping and rounding [7, 20].

To solve the problems, in this paper we introduce **Differentiable Soft Quantization (DSQ)** to well approximate the standard binary and uniform quantization process (see the framework in Figure 1). DSQ employs a series of hyperbolic tangent functions to gradually approach the staircase function for low-bit quantization (e.g., sign for 1-bit case), and meanwhile keeps the smoothness for easy gradient calculation. We reformulate the DSQ function with respect to an approximation characteristic variable, and correspondingly develop an evolution training strategy to progressively learn the differential quantization function. During training, the approximation between DSQ and standard quantization can be controlled by the characteristic variable, which together with the clipping values can be automatically determined in the network. Our DSQ decreases deviations caused by extremely low-bit quantization, and thus makes the forward and backward process more consistent and stable in the training.

The specific design makes the DSQ own the following advantages compared to the state-of-the-art solutions:

- **Novel quantization**. We introduce a DSQ function to well approximate the standard binary and uniform quantization. The approximation of DSQ can be easily controlled in the evolution training way.
- **Easy convergence**. DSQ acts as a rectifier to gradually redistribute the data according to quantization points. Thus the backward propagation becomes more consistent with the forward pass, leading to an easier convergence with the accurate gradients.
- **Balanced loss**. With the help of DSQ, we can jointly determine the clipping range and approximation of the quantization, and thus balance the quantization loss including clipping error and rounding error.
- **High efficiency**. DSQ can be implemented based on our fast computation kernels, and its inference speed surpasses most open-source high performance inference frameworks.
- **Strong flexibility**. DSQ is compatible with the binary or uniform quantization methods, easy to deploy in state-of-the-art network structures and able to get further accuracy improvement.

2. Related Work

2.1. Network Quantization

Network quantization aims to obtain low-precision networks with high accuracy. One way to speed up low-precision networks is to utilize bit operation [16, 9, 8, 25, 45, 23, 13, 33]. They quantize weights or activations to {-1, +1} or {-1, 0, +1}. Because only two or three values can be used, training a binary or ternary model with high accuracy is very challenging. Another way to achieve acceleration is to uniformly convert weight and activations to fix-point representation. [18] has verified the feasibility of 8-bit uniform fix-point quantization. But lower bit quantization faces more challenges on accuracy. To address this problem, [7, 20] try to optimize the clipping value or quantization interval for the task specific loss in an end-to-end manner. [36, 46, 39] apply different techniques to find optimal bit-widths for each layer. [42] and [20] optimize the training process with incremental and progressive quantization. [29] and [16] adjust the network structure to adapt to quantization. [28] introduces knowledge distillation to improve quantized networks' performance. [41] learns a more flexibility quantizer with basis vector. Besides, there are other non-uniform methods like [30, 13, 3, 35, 37, 38, 44] which may need delicate hardware to get acceleration.

2.2. Efficient Deployment

With hardware-friendly network quantization, many efficient deployment frameworks are emerging. NVIDIA TensorRT [27] is a high-performance deep learning inference platform. It provides INT8 optimizations for deployments on GPU devices. Intel Caffe [11] is a fork of official Caffe [19] to improve performance on CPU, in particular Intel Xeon processors. Gemmlowp is a low-precision GEMM library in tensorflow [1] supporting ARM and Intel X86. NCNN [31] is Tencent's inference framework optimized for mobile platforms. These frameworks usually support 8-bit integer arithmetic, but does not do specific optimization for lower bit computation. To validate the efficiency of our quantization method, in this paper we implement 2-bit fast integer arithmetic with ARM NEON technology, which is an advanced SIMD architecture extension for the ARM Cortex-A series and Cortex-R52 processors.

3. Differentiable Soft Quantization

In this paper, we consider the standard 1-bit binary and multi-bit uniform quantization.

3.1. Preliminaries

For 1-bit binary quantization, the binary neural network (BNN) limits its activations and weights to either -1 or +1,

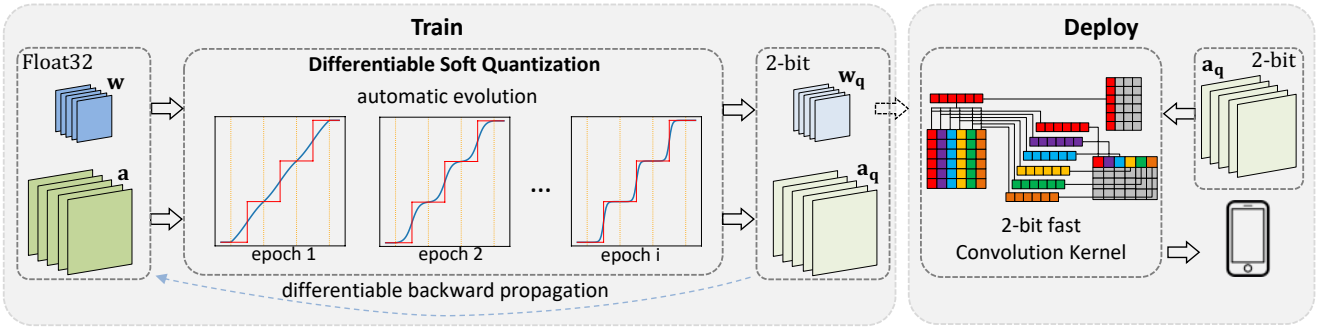


Figure 1. An overview of Differentiable Soft Quantization (DSQ), taking 2-bit uniform quantization as an example. During training, we apply piecewise DSQ to redistribute the data and make it automatically evolve in each epoch to behave more like uniform quantization. After training, the piecewise DSQ can completely convert to the hard uniform quantization by sign operation, ensuring an easy and efficient deployment on resource limited devices.

usually using the binary function:

$$Q_B(x) = \text{sgn}(x) = \begin{cases} +1, & \text{if } x \geq 0, \\ -1, & \text{otherwise.} \end{cases} \tag{1}$$

For multi-bit uniform quantization, given the bit width b and the floating-point activation/weight x following in the range (l, u), the complete quantization-dequantization process of uniform quantization can be defined as:

$$Q_U(x) = \text{round}(\frac{x}{\Delta})\Delta, \tag{2}$$

where the original range (l, u) is divided into $2^b - 1$ intervals $\mathcal{P}_i, i \in (0, 1, \ldots, 2^b - 1)$, and $\Delta = \frac{u-l}{2^b-1}$ is the interval length.

3.2. Quantization function

The derivative of binary/uniform quantization function is zero almost everywhere, which not only makes the training of the quantized network unstable, but also largely decreases the accuracy. To reduce the gap between the full-precision model and its quantized low-precision model, a differentiable asymptotic function is first introduced to approximately model the binary/uniform quantizer. This function handles the point x falling in different intervals $\mathcal{P}_i$:

$$\varphi(x) = s \tanh\left(k(x - m_i)\right), \quad \text{if } x \in \mathcal{P}_i, \tag{3}$$

with

$$m_i = l + (i + 0.5)\Delta \text{ and } s = \frac{1}{\tanh(0.5k\Delta)}. \tag{4}$$

The scaling parameter s guarantees that $\tanh$ functions of φ for the adjacent intervals can be smoothly connected (see Figure 2(a)). Owing to the highly symmetry of the $\tanh$ function, φ will be continuously differentiable everywhere. Besides, the coefficient k determines the shape of the asymptotic function. Namely, the larger k is, the more the asymptotic function behaves like the desired staircase function generated by uniform quantizer with multiple quantization levels.

Based on the asymptotic function φ, we can have our differentiable soft quantization (DSQ) function, approximating the uniform quantizer:

$$Q_S(x) = \begin{cases} l, & x < l, \\ u, & x > u, \\ l + \Delta\left(i + \frac{\varphi(x)+1}{2}\right), & x \in \mathcal{P}_i \end{cases} \tag{5}$$

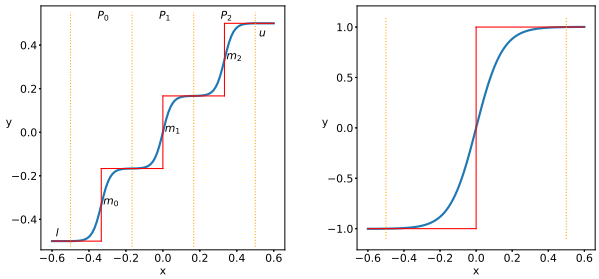
As shown in Figure 1, the curve of the asymptotic function gradually approaches the piecewise curve of the uniform quantizer, when k becomes larger. Thus in practice we can also utilize it to simulate the influence of real quantization in the forward pass and this function is well behaved for purposes of calculating gradients in the backward pass. From another view, DSQ acts as a rectifier, which align the data with its quantization points with small quantization error simply by the redistribution. Subsequently, during the backward propagation, the gradient can better reflect the correct direction of updating.

When φ composites with the sign function, DSQ can serve as a piecewise uniform quantizer (see Figure 2(a)). It is worth mentioning that when there is only one interval, we can also simulate the standard model binarization in this way (see Figure 2(b)). This means the binary quantization in (1) can also be treated as a special case of our soft quantization function.

3.3. Evolution to the standard quantization

With the DSQ function, we can easily find a differentiable substitute for standard quantization. However, how well the DSQ approximates the standard quantization can largely affect the behaviors of the quantized models. Namely, it is highly required we can adaptively choose the appropriate parameters of DSQ in the training process, and thus promise the controlled approximation according to the optimization goal of the quantized network.

To achieve this goal, it is important to find a characteristic variable to measure the approximation between DSQ

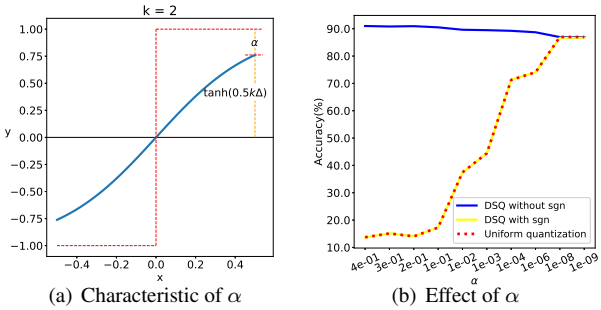


(a) Piecewise DSQ (b) Binary DSQ

Figure 2. Curves of differentiable soft quantization function for (a) uniform quantization, (b) binary quantization, a special case of (a).

and standard quantization. Figure 3(a) shows the curve of DSQ in one interval without scaling to -1 and $+1$. It is easy to prove that when the distance from the maximal point in the curve to the upper bound $+1$ is small enough, DSQ function can perfectly approximates the standard quantizer. Based on this observation, we introduce the characteristic variable α as follows:

$$\alpha = 1 - \tanh(0.5k\Delta) = 1 - \frac{1}{s}. \tag{6}$$



(a) Characteristic of α (b) Effect of α

Figure 3. The characteristic variable α in DSQ.

Furthermore, we can reformulate the DSQ function only with respect to the parameter α and Δ. Specifically, based on the above equation, we have

$$s = \frac{1}{1-\alpha}. \tag{7}$$

With the facts $\Delta = \frac{u-l}{2^b-1}$ and $\varphi(0.5\Delta) = 1$, we also have

$$k = \frac{1}{\Delta}\log(\frac{2}{\alpha} - 1). \tag{8}$$

Figure 3(b) illustrates the effect of α, where 2-bit quantized models are first trained using DSQ with respect to different α. The curves respectively show the inference accuracy performance of DSQ, DSQ appended with sign function, and standard uniform quantization. It is easy to see that our DSQ with sign function can perform quite consistently with the uniform quantization. Especially, when α is small, DSQ can well approximate the performance of the uniform quantization. This means an appropriate α will enble DSQ to help train a quantized model with high accuracy.

Now we can see that the approximation power of DSQ actually depends on the appropriate α, which plays an important role in the optimization. To adaptively determine it, we introduce an evolution training strategy, which takes α as a variable to be optimized in the quantized network. In this way, we can adaptively adjust α and force DSQ to evolve into the standard quantizer during the training. Formally, we can formulate it as the network loss minimization problem with respect to the input x and output y at each layer:

$$\min_{\alpha} \mathcal{L}(\alpha; x, y) \quad \text{s.t. } ||\alpha||_2 < \lambda \tag{9}$$

According to the formulation, we can calculate the gradient of α in the backward process and then automatically adjust it with the L2 regularization constraint.

$$\frac{\partial y}{\partial \alpha} = \begin{cases} 0, & x < l, \\ 0, & x > u, \\ \frac{\partial Q_S(x)}{\partial \alpha}, & x \in P_i \end{cases} \tag{10}$$

3.4. Balancing clipping error and rounding error

Clipping and rounding together cause the quantization error. Usually, when the quantizer clips more, the clipping error increases and the rounding error decreases. Owing to the differentiable evolution from the soft quantizer to the standard quantizer, we can further analyze the connections between clipping error and rounding error in DSQ. Specifically, we can jointly optimize the lower bound and upper bound of clipping to pursue a balance between clipping error and rounding error.

$$\frac{\partial y}{\partial l} = \begin{cases} 1, & x < l, \\ 0, & x > u, \\ 1 + q\frac{\partial \Delta}{\partial l} + \frac{\Delta}{2}\frac{\partial \varphi(x)}{\partial l}, & x \in P_i \end{cases} \tag{11}$$

$$\frac{\partial y}{\partial u} = \begin{cases} 0, & x < l, \\ 1, & x > u, \\ q\frac{\partial \Delta}{\partial u} + \frac{\Delta}{2}\frac{\partial \varphi(x)}{\partial u}, & x \in P_i \end{cases} \tag{12}$$

where

$$q = i + \frac{1}{2}(\varphi(x) + 1). \tag{13}$$

From (11) and (12), we can conclude that the large outlier points are clipped by u and mainly contribute to the update of u, while the small ones are clipped by l and mainly contribute to the update of l. Data points falling in the middle range will influence derivative of both u and l. When clipping error dominates the whole quantization error, the outliers' gradients will be large and thus serve as the major power for weight updating. Otherwise, when rounding error dominates the error, points in the middle range will affect more in the process of backward propagation.

Now we can see our DSQ function has three key parameters: α, l and u, all of which can be optimized during training. By optimizing the clipping values jointly with the similarity factor α, we not only find an evolutional and differentiable approximation to the standard quantization function, but also balance the clipping error and rounding error, which together help bridge the accuracy gap between the full-precision and extremely low-bit quantized models.

3.5. Training and Deploying

In this paper DSQ function is proposed with the evolution training to optimize both the DSQ and network parameters, aiming to fine-tune a quantized network from full-precision network. The detailed fine-tuning procedures for the convolution network are listed in Algorithm 1.

Algorithm 1 Feed-Forward and Back-Propagation procedures for a convolution layer quantized with DSQ.

Input: the input activation $\mathbf{a}$

Parameters: weight $\mathbf{w}$, clip value $\mathbf{l_a}$, $\mathbf{u_a}$, $\mathbf{l_w}$, $\mathbf{u_w}$ and similarity factor $\alpha_\mathbf{a}$, $\alpha_\mathbf{w}$

Output: the output activation $\mathbf{o}$

Feed-Forward

Clip $\mathbf{a}$ with $\mathbf{l_a}$,$\mathbf{u_a}$ and clip $\mathbf{w}$ with $\mathbf{l_w}$, $\mathbf{u_w}$

Apply asymptotic function φ to activation and weight

$$\mathbf{a_{sq}} = \varphi_a(\mathbf{a})$$
$$\mathbf{w_{sq}} = \varphi_w(\mathbf{w})$$

Keep consistent with standard quantization

$$\mathbf{a_q} = \text{sgn}(\mathbf{a_{sq}})$$
$$\mathbf{w_q} = \text{sgn}(\mathbf{w_{sq}})$$

Dequantize $\mathbf{a_q}$ and $\mathbf{w_q}$

$$\hat{\mathbf{a}} = \mathbf{l_a} + \Delta_a(\mathbf{i} + \frac{\mathbf{a_q}+1}{2})$$
$$\hat{\mathbf{w}} = \mathbf{l_w} + \Delta_w(\mathbf{j} + \frac{\mathbf{w_q}+1}{2})$$

Calculate the output: $\mathbf{o} = \textit{Convolution}(\hat{\mathbf{w}}, \hat{\mathbf{a}})$

Backward-Propagation

Calculate the gradients(take $\mathbf{a}$ as an example)

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{a}}} \frac{\partial \hat{\mathbf{a}}}{\partial \mathbf{a_q}} \frac{\partial \mathbf{a_{sq}}}{\partial \mathbf{a}}$$
$$\frac{\partial \mathcal{L}}{\partial \alpha_\mathbf{a}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{a}}} \frac{\partial \hat{\mathbf{a}}}{\partial \mathbf{a_q}} \frac{\partial \mathbf{a_{sq}}}{\partial \alpha_\mathbf{a}}$$
$$\frac{\partial \mathcal{L}}{\partial \mathbf{l_a}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{a}}}\{1 + (\mathbf{i} + \frac{\mathbf{a_q}+1}{2})\frac{\partial \Delta_a}{\partial \mathbf{l_a}} + \frac{\Delta_a}{2}\frac{\partial \mathbf{a_{sq}}}{\partial \mathbf{l_a}}\}$$
$$\frac{\partial \mathcal{L}}{\partial \mathbf{u_a}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{a}}}\{(\mathbf{i} + \frac{\mathbf{a_q}+1}{2})\frac{\partial \Delta_a}{\partial \mathbf{u_a}} + \frac{\Delta_a}{2}\frac{\partial \mathbf{a_{sq}}}{\partial \mathbf{u_a}}\}$$

Parameters Update

Update all parameters with the learning rate η

For deploying on devices with limited computing resources, we also implement the low-bit computation kernels to accelerate the inference on ARM architecture. In the convolution networks, multiply and accumulation are the core operations of General Matrix Multiply (GEMM), which can be efficiently completed by the MLA instruction on ARM NEON. MLA multiplies two numbers in 8-bit registers and accumulates the result into an 8-bit register. In case that the accumulator is nearly overflowed, we can transfer the value to a 16-bit register by the SADDW instruction. Figure 4 shows the complete data flow of our GEMM kernel.

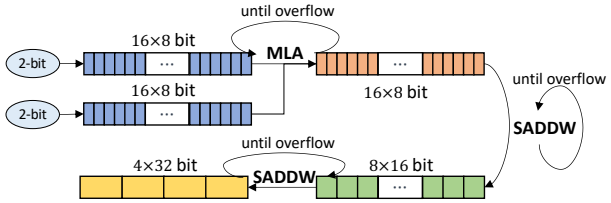


Figure 4. Data flow in our fast GEMM implementation.

Though SADDW takes extra computational cost than MLA, fortunately we can reduce the probability that the data transferring occurs. Given two b-bit signed numbers, we can call MLA instruction up to $\frac{2^7-1}{(-2^{b-1})^2}$ times until transferring the result to the 16-bit register by SADDW. The ratio of times for calling MLA and 16-bit register SADDW is listed in Table 1. It can be concluded that better acceleration performance will be achieved with lower quantized bits. In practice, our low-bit GEMM kernels can outperform the other open-source inference frameworks.

Table 1. Ratio of time for calling MLA (8-bit register) and SADDW (16-bit register) with respect to different number of quantized bits.

b	2	3	4
MLA/SADDW	31/1	7/1	1/1

4. Experiments

In this section, we conduct extensive experiments to demonstrate the effectiveness of the proposed DSQ, on two popular image classification datasets: CIFAR-10 [22] and ImageNet (ILSVRC12) [10]. The CIFAR-10 dataset consists of 50K training images and 10K testing images of size 32×32 with 10 classes. ImageNet ILSVRC12 contains about 1.2 million training images and 50K testing images with 1,000 classes.

4.1. Settings

DSQ function: We implement DSQ using Pytorch [32], as a flexible module that can be easily inserted to the binary or uniform quantization models. Because the DSQ function is differentiable, it can be implemented directly with Pytorch's automatic differentiation mechanism. There are two ways to quantize model to 1-bit, i.e., the binarization with $\{-1, +1\}$ and the uniform quantization. Our DSQ function is compatible with both approaches. When building up a quantized model, we simply insert DSQ function to all places that will be quantized, e.g., the inputs and weights of a convolution layer.

Network structures: We employ the widely-used network structures including VGG-Small [41], ResNet-20 for CIFAR-10, and ResNet-18, ResNet-34 [14], MobileNetV2 [34] for ImageNet. For binarized models, we adopt

parameter-free type-A shortcut as [15] and apply the activation function replacement introduced by [16]. All convolution and fully-connected layers except the first and the last one are quantized with DSQ.

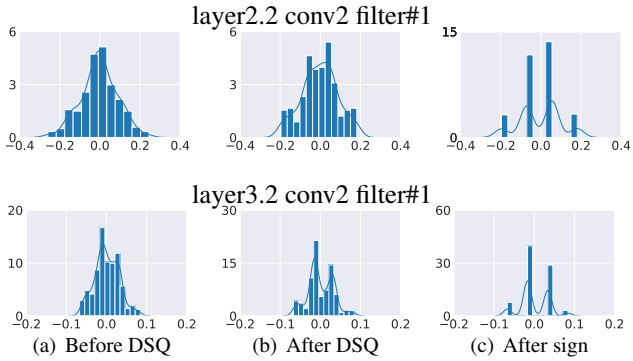
Initialization: For initialization, we try fine-tuning from a pre-train model and training from scratch. For hyper-parameters, we follow the rules described in the origin papers [7, 14, 15, 41]. For parameter α, we choose 0.2 as an initial value. For clipping value l and u, we try the following two strategies: moving average statistics and optimization by backward propagation.

4.2. Analysis of DSQ

First, we empirically analyze DSQ from the aspects of rectification, convergence, evolution, etc.

4.2.1 Rectification

An important effect of DSQ function is to redistribute the data and align them to the quantization values, which subsequently decrease the backward propagation error. To investigate this point, in Figure 5 we visualize the weight's distribution of ResNet-20 on CIFAR-10 before and after 2-bit DSQ function. From the figure, we can observe that the data distribution, originally similar to normal distribution (Figure 5(a)), appears with a few peaks in the histogram after DSQ's rectification (Figure 5(b)), and subsequently the data can be completely quantized to 4 quantization points after the sign function (Figure 5(c)). This observation proves that DSQ serves as a promising bridge between the origin full-precision model and low-bit quantized model, largely reducing the quantization loss in practice.



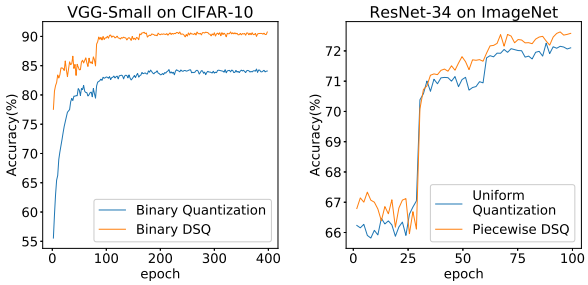
(a) Before DSQ (b) After DSQ (c) After sign

Figure 5. Data distribution before and after DSQ (2-bit).

4.2.2 Convergence

State-of-the-art binary/uniform quantized networks often adopt the training strategy that directly applies quantization in forwarding process, but STE in backward process. They ignore the negative effect of the quantization loss on the gradient calculation, and thus often face the unstable training in most cases. We here show that training these models with our DSQ can significantly improve the ability to converge.

With the redistribution of DSQ, the numerical difference between quantized data and full-precision data is decreased, contributing to a more accurate backward. From another perspective, the introducing of DSQ can be viewed as an optimizer of STE, which can improve the ability to converge in process of optimization. Figure 6 compares the accuracy curves of validation with and without DSQ when using binarization in VGG-Small on CIFAR-10 and 3-bit uniform quantization in ResNet-34 on ImageNet. We can find that training with DSQ can achieve higher accuracy.

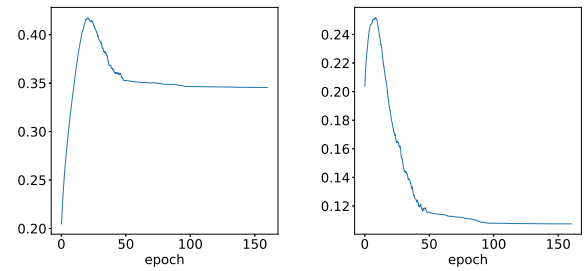


(a) VGG-Small (b) ResNet-34

Figure 6. Training quantized model with/without DSQ function.

4.2.3 Evolution

The automatic evolution of DSQ function is the key to the feasible approximation to the standard quantization, which is determined by the similarity factor α. In our experiments, for both weights and activations of ResNet-20 on CIFAR-10, α is initialized to 0.2, and to ensure the stability, we limit it to a reasonable range of $(0, 0.5)$ with $k \leq 1000$. Figure 7 (a) and (b) respectively plot the curves of α per step for activations and weights during the evolution training.



(a) α for activations (b) α for weights

Figure 7. Automatic evolution of α during training.

For both activations and weights, we can see that at the beginning of training, α will increase sharply. Considering when α becomes large, DSQ behaves more like an identity operation, this phenomenon implies that we should not quantize much at the beginning. After that, α decreases gradually and finally converges to a stable value, which makes DSQ approximate the standard quantization. In the whole training process, α is automatically adjusted according to the network loss, and thus enables the stronger flexibility of DSQ than the incremental quantization relying on manually adjusting.

Table 2. Learnt α for activations and weights of ResNet-20

Layer	Weight	Activation
layer1.0.conv1	0.1075	0.3455
layer1.0.conv2	0.0950	0.3101
layer1.1.conv1	0.0895	0.3046
layer1.1.conv2	0.0868	0.2526
layer1.2.conv1	0.1368	0.3621
layer1.2.conv2	0.0926	0.3401
layer2.0.conv1	0.1578	0.3903
layer2.0.conv2	0.1810	0.3785
layer2.0.downsample.0	**0.0828**	0.2887
layer2.1.conv1	0.1641	0.2722
layer2.1.conv2	0.1162	0.2663
layer2.2.conv1	0.1605	0.2690
layer2.2.conv2	0.1059	0.2440
layer3.0.conv1	0.2042	0.3993
layer3.0.conv2	**0.2779**	**0.4532**
layer3.0.downsample.0	0.0914	**0.2327**
layer3.1.conv1	0.2484	0.4241
layer3.1.conv2	0.2301	0.3918
layer3.2.conv1	0.1965	0.4238
layer3.2.conv2	0.0975	0.3277

Table 2 also reports the final optimal α of each layer. First, we find that usually α of weights is smaller than that of activations. This means that in the low-bit quantized network usually weights are more tolerant to the quantization, while activations are more sensitive, which conforms to the experiences and conclusions of previous studies [43, 7]. Second, different layers show different sensitivity to the quantization. For example, the downsampling convolution layers can be quantized much (a small α), while some layers such as layer3.0.conv2 are not suitable for quantization (a large α). This conclusion is very useful for understanding and improving the network quantization.

4.3. Ablation study

To further understand how DSQ works in practice, we also conduct the ablation study on both model binarization and uniform quantization. We specify α as 0.2 for network binarization experiments. From Table 3, we can see that even the naive DSQ with the fixed α brings stable improvement over the basic binarization strategy that directly applies sign function over ResNet-20 on CIFAR-10, e.g., nearly 2% performance gain for the 1W1A (1-bit quantization for both weight and activation) case.

In Table 4, we further study the effect of evolution training (learnt α) and balanced quantization error (learnt l, u) over the 2-bit uniform quantization of ResNet-20 on CIFAR-10. It is obvious that learning the adaptive α and the clipping values l, u respectively brings accuracy improvement. Besides, since DSQ can be conveniently inserted to any standard quantization method, we further investigate its performance over the state-of-the-art quantization method PACT [7]. We implement the PACT method and the results in Table 4 show that DSQ can further boost the performance of PACT, proving the flexibility and generality of DSQ.

Table 3. Ablation study on 1-bit binarized quantization.

Method	Bit-Width (W/A)	Accuracy (%)
FP	32/32	90.84
Binary	1/1	82.46
Binary DSQ	1/1	83.80
Piecewise DSQ	1/1	**84.11**
Binary	1/32	90.11
Binary DSQ	1/32	**90.24**
Piecewise DSQ	1/32	90.03

Table 4. Ablation study on 2-bit uniform quantization.

Method	Top-1 (%)	Top-5 (%)
Standard Quantization	86.63	99.35
Fixed α	86.95	99.50
Learnt α	87.25	99.49
Learnt α, l, u	**88.44**	**99.50**

Table 5. Ablation study on 2-bit PACT.

Method	Top-1 (%)	Top-5 (%)
PACT	88.24	99.60
PACT+DSQ	**90.11**	**99.71**

4.4. Comparison with State-of-the-arts

Next we comprehensively evaluate DSQ by comparing it with the existing state-of-the-art quantization methods.

Comparison on CIFAR-10: Table 6 lists the performance using different methods on CIFAR-10, respectively including BNN [16], XNOR-Net [33] over VGG-Small, and DoReFa-Net [43], LQ-Net [41] over ResNet-20. All methods quantize weights or activations to 1 bit. In all cases, our DSQ obtains the best performance for the two different network structures. More importantly, when using 1-bit activations and 1-bit weights (1/1), our method gets very significant improvement (i.e., 84.11% v.s. 79.30%) over the state-of-the-art DoReFa-Net [43]. Note that for VGG-Small, DSQ using 1-bit weights and activations can even obtain better performance than the full-precision model.

Comparison on ImageNet: For the large-scale case, we study the performance of DSQ over ResNet-18, ResNet-34 and MobileNetV2 on ImageNet. Table 7 shows a number of state-of-the-art quantization methods including BWN [33], HWGQ [5], TWN [23], PACT [7], LQ-Net [41], ABC-Net [25] and BCGD [40], with respect to different settings. From the table, we can observe that when only quantizing weights over ResNet-18, DSQ using 1 bit outperforms BWN and HWGQ by large margins and even surpasses TWN using 2 bits. Besides, over both ResNet-18 and ResNet-34, the accuracy of our method using 2-bit and 3-bit quantization is also consistenly higher than LQ-Net

Table 6. Comparison of 1-bit quantized models on CIFAR-10.

Model	Method	Bit-Width (W/A)	Accuracy (%)
VGG-Small	FP	32/32	91.65
	BNN [16]	1/1	89.90
	XNOR [33]	1/1	89.80
	Ours	1/1	**91.72**
ResNet-20	FP	32/32	90.78
	DoReFa [43]	1/1	79.30
	Ours	1/1	**84.11**
	DoReFa [43]	1/32	90.00
	LQ-Net [41]	1/32	90.10
	Ours	1/32	**90.24**

Table 7. Comparison of different quantized models on ImageNet.

Model	Method	Bit-Width (W/A)	Accuracy (%)
ResNet-18	FP	32/32	69.90
	BWN [33]	1/32	60.80
	HWGQ [5]	1/32	61.30
	TWN [23]	2/32	61.80
	Ours	1/32	**63.71**
	PACT [7]	2/2	64.40
	LQ-Net [41]	2/2	64.90
	Ours	2/2	**65.17**
	ABC-Net [25]	3/3	61.00
	PACT [7]	3/3	68.10
	LQ-Net [41]	3/3	68.20
	Ours	3/3	**68.66**
	BCGD [40]	4/4	67.36†
	Ours	4/4	**69.56**†
ResNet-34	FP	32/32	73.80
	LQ-Net [41]	2/2	69.80
	Ours	2/2	**70.02**
	ABC-Net [25]	3/3	66.70
	LQ-Net [41]	3/3	71.90
	Ours	3/3	**72.54**
	BCGD [40]	4/4	70.81†
	Ours	4/4	**72.76**†
Mobile-NetV2	FP	32/32	71.87
	PACT [7, 36]	4/4	61.40
	Ours	4/4	**64.80**

* The † represents the results of full quantization for activations and weights across all convolution layers.

and ABC-Net. We should point out that LQ-Net is a non-uniform quantization method. This means that our DSQ simultaneously enjoys efficient inference as the uniform methods, and competitive performance to the more complex non-uniform solutions. What's more, the accuracy of DSQ on efficient networks such as MobileNetV2 also significantly exceeds existing methods (e.g., 3.4% higher than PACT [7, 36] using 4-bit), which proves the protential of DSQ on hardware-friendly networks with a small amount of parameters.

4.5. Deploying Efficiency

Finally, we highlight the uniqueness of our DSQ that we support extremely low-bit (less than 4-bit) integer arithmetic based on GEMM kernels with ARM NEON 8-bit instructions, while existing open-source high performance inference frameworks (e.g., NCNN-8-bit [31]) usually only support 8-bit operations. In practice, the lower bit width doesn't mean a faster inference speed, mainly due to the overflow and tranferring among the registers as analyzed in Section 3.5. But fortunately, as Table 8 shows, our implementation can accelerate the inference even using the extreme lower bits. We also test the real speed of our implementation when quantizing ResNet-18 with DSQ on Raspberry Pi 3B, which has a 1.2 GHz 64-bit quad-core ARM Cortex-A53. As shown in Table 9, the inference speed using DSQ is much faster than that of NCNN.

Table 8. Time cost (ms) of the typical 3×3 convolution in ResNet using different number of bits (single thread).

input size	#output	4-bit	3-bit	2-bit
64x56x56	64	43.80	40.06	**38.11**
128x28x28	128	33.89	29.94	**28.15**
256x14x14	256	37.03	31.16	**29.20**
512x7x7	512	30.20	26.14	**25.43**

Table 9. Comparison of time cost of ResNet-18 on different inference frameworks with different bits (single thread).

	DSQ 2-bit	NCNN 8-bit [31]
time (ms)	**551.22**	935.51

* NCNN was tested with commit d263cd5 on 2019.3.15.

5. Conclusions

In this paper, we proposed the Differentiable Soft Quantization (DSQ) method to eliminate the accuracy gap between the full-precision networks and low-bit (binary/uniform) quantization networks. DSQ can evolve dynamically during end-to-end training to approximate standard quantization. Since it can reduce both the gradient deviation of backward propagation and the quantization loss in forward inference, state-of-the-art accuracy for various network structures can be promised. As a general module supporting both model binarization and uniform quantization, it also enjoys strong flexibility to improve the performance of different quantization methods, and high hardware-friendly efficiency based on the fast fixpoint GEMM kernels implementation.

Acknowledge

This work was supported by National Natural Science Foundation of China (61690202, 61872021), Fundamental Research Funds for Central Universities (YWF-19-BJ-J-271), Beijing Municipal Science and Technology Commission (Z171100000117022), and State Key Lab of Software Development Environment (SKLSDE-2018ZX-04).

References

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[2] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment. *arXiv preprint arXiv:1810.05723*, 2018.

[3] Chaim Baskin, Eli Schwartz, Evgenii Zheltonozhskii, Natan Liss, Raja Giryes, Alex M. Bronstein, and Avi Mendelson. Uniq: Uniform noise injection for non-uniform quantization of neural networks. *arXiv preprint arXiv:1804.10969*, 2018.

[4] Yoshua Bengio, Nicholas Lonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[5] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017.

[6] Jungwook Choi, Pierce I-Jen Chuang, Zhuo Wang, Swagath Venkataramani, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Bridging the accuracy gap for 2-bit quantized neural networks (qnn). *arXiv preprint arXiv:1807.06964*, 2018.

[7] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[8] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *arXiv preprint arXiv:1511.00363*, 2015.

[9] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2009.

[11] Jiong Gong, Haihao Shen, Guoming Zhang, Xiaoli Liu, Shane Li, Ge Jin, Niharika Maheshwari, Evarist Fomenko, and Eden Segal. Highly efficient 8-bit low precision inference of convolutional neural networks with intelcaffe. *Proceedings of the 1st on Reproducible Quality-Efficient Systems Tournament on Co-designing Pareto-efficient Deep Learning - ReQuEST 18*, 2018.

[12] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. Eie. *ACM SIGARCH Computer Architecture News*, 44(3):243254, Jun 2016.

[13] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *Lecture Notes in Computer Science*, page 630645, 2016.

[16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems 29*, pages 4107–4115. Curran Associates, Inc., 2016.

[17] Benoit Jacob et al. gemmlowp: a small self-contained low-precision gemm library.(2017), 2017.

[18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2018.

[19] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[20] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Youngjun Kwak, Jae-Joon Han, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. *arXiv preprint arXiv:1808.05779*, 2018.

[21] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[22] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, page 4, 2014.

[23] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[24] Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. Training quantized nets: A deeper understanding. *arXiv preprint arXiv:1706.02379*, 2017.

[25] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 345–353. Curran Associates, Inc., 2017.

[26] Jeffrey L. McKinstry, Steven K. Esser, Rathinakumar Appuswamy, Deepika Bablani, John V. Arthur, Izzet B. Yildiz, and Dharmendra S. Modha. Discovering low-precision networks close to full-precision networks for efficient embedded inference. *arXiv preprint arXiv:1809.04191*, 2018.

[27] Szymon Migacz. 8-bit inference with tensorrt. In *GPU Technology Conference*, 2017.

[28] Asit Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*, 2017.

[29] Asit Mishra, Eriko Nurvitadhi, Jeffrey J Cook, and Debbie Marr. Wrpn: Wide reduced-precision networks. *arXiv preprint arXiv:1709.01134*, 2017.

[30] Daisuke Miyashita, Edward H. Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016.

[31] nihui et al. Ncnn. https://github.com/Tencent/ncnn, 2017.

[32] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[33] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *Lecture Notes in Computer Science*, page 525542, 2016.

[34] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[35] Frederick Tung and Greg Mori. Clip-q: Deep network compression learning by in-parallel pruning-quantization. *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[36] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization. *arXiv preprint arXiv:1811.08886*, 2018.

[37] Peisong Wang, Qinghao Hu, Yifan Zhang, Chunjie Zhang, Yang Liu, and Jian Cheng. Two-step quantization for low-bit neural networks. *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[38] Yuhui Xu, Yongzhuang Wang, Aojun Zhou, Weiyao Lin, and Hongkai Xiong. Deep neural network compression with single and multiple level quantization. In *AAAI*, 2018.

[39] Amir Yazdanbakhsh, Ahmed T. Elthakeb, Prannoy Pilligundla, FatemehSadat Mireshghallah, and Hadi Esmaeilzadeh. Releq: An automatic reinforcement learning approach for deep quantization of neural networks. *arXiv preprint arXiv:1811.01704*, 2018.

[40] Penghang Yin, Shuai Zhang, Jiancheng Lyu, Stanley Osher, Yingyong Qi, and Jack Xin. Blended coarse gradient descent for full quantization of deep neural networks. *Research in the Mathematical Sciences*, 6(1):14, 2019.

[41] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[42] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

[43] 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.

[44] Shu-Chang Zhou, Yu-Zhi Wang, He Wen, Qin-Yao He, and Yu-Heng Zou. Balanced quantization: An effective and efficient approach to quantized neural networks. *Journal of Computer Science and Technology*, 32(4):667682, Jul 2017.

[45] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[46] Xiaotian Zhu, Wengang Zhou, and Houqiang Li. Adaptive layerwise quantization for deep neural network compression. *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2018.