KCNN: Kernel-wise Quantization to Remarkably Decrease Multiplications in Convolutional Neural Network

Linghua Zeng, **Zhangcheng Wang** and **Xinmei Tian***

CAS Key Laboratory of Technology in Geo-Spatial Information Processing and Application Systems,
University of Science and Technology of China, China
zenglh@mail.ustc.edu.cn, wzc1@mail.ustc.edu.cn, xinmei@ustc.edu.cn

Abstract

Convolutional neural networks (CNNs) have demonstrated state-of-the-art performance in computer vision tasks. However, the high computational power demand of running devices of recent CNNs has hampered many of their applications. Recently, many methods have quantized the floating-point weights and activations to fixed-points or binary values to convert fractional arithmetic to integer or bit-wise arithmetic. However, since the distributions of values in CNNs are extremely complex, fixed-points or binary values lead to numerical information loss and cause performance degradation. On the other hand, convolution is composed of multiplications and accumulation, but the implementation of multiplications in hardware is more costly comparing with accumulation. We can preserve the rich information of floating-point values on dedicated low power devices by considerably decreasing the multiplications. In this paper, we quantize the floating-point weights in each kernel separately to multiple bit planes to remarkably decrease multiplications. We obtain a closed-form solution via an aggressive Lloyd algorithm and the fine-tuning is adopted to optimize the bit planes. Furthermore, we propose dual normalization to solve the pathological curvature problem during fine-tuning. Our quantized networks show negligible performance loss compared to their floating-point counterparts.

1 Introduction

Recently, convolutional neural networks have demonstrated state-of-the-art performance in many computer vision tasks [Goodfellow *et al.*, 2016]. Among them, the image classification task is particularly significant and fundamental. Many well-known network structures [Krizhevsky *et al.*, 2012; He *et al.*, 2016] have been trained on an image classification dataset [Russakovsky *et al.*, 2015] and then adapted to other tasks, such as object detection, image segmentation, and video recognition [Zhang *et al.*, 2018]. Although the classification accuracy of recent CNNs continues to improve, the high computational power demand of CNNs has hampered their adoption in a wide range of applications. They have become too cumbersome to be implemented on mobile devices with limited power budgets [Courbariaux *et al.*, 2015] or as part of web services that are sensitive to computational overhead [Han *et al.*, 2015]. In this work, we focus on decreasing computational power demand of running devices of CNNs on the image classification task. The accelerated network structures can be easily adapted to other tasks like object detection and video recognition.

Convolution operation consists of equal numbers of multiplications and additions which compose the accumulations. However, many previous methods [Tai *et al.*, 2015; Lin *et al.*, 2017] ignored the different complexity between multiplications and additions. Multiplication is much harder to be implemented on the hardware than addition [David *et al.*, 2007; Wolf, 2002]. For integer arithmetic, the multiplier needs $O(n^2)$ transistors, while the adder needs $O(n)$ transistors, where n is the number of bits. For fractional arithmetic, the situation is more complex but still a multiplier is considerably more complex than an adder. Therefore, the majority of the computational resources are consumed by multiplications.

In this paper, we only decrease the number of multiplications by quantizing the floating-point weights in each kernel separately to multiple bit planes. Since we quantize the weights in convolution operations to binary values (+1 and -1), we eliminate the multiplications and only conduct accumulation with sign reverse in convolution operations. BWN [Courbariaux *et al.*, 2015] and TWN [Zhu *et al.*, 2016] have similar motivation like us. However, they only use one bit plane which makes their method suffer from huge accuracy loss and be inefficient and impractical in industrial applications. Further more, it is generally accepted that the lower complexity of accumulations makes it possible to improve the accuracy at the cost of increasing accumulations. We quantize weights to multiple bit planes inspired by ABC [Lin *et al.*, 2017]. But unlike ABC, we only conduct quantization on weights, thus we still utilize fractional arithmetic which have advantages in handling large fluctuations or complex distributions of values. Following previous methods (e.g. BWN, TWN, ABC), we scale the resulting features by the assigned floating-point values after quantized convolution operations.

*Contact Author

This scaling operation requires far fewer multiplications than the original convolution operation. In addition, we find that the convolutions of different kernels in a convolutional layer are independent and unique. Quantizing each kernel separately is more accurate than previous works and thus results in a promotion in accuracy. Besides, to make our algorithm more elegant, we abandon the complex tricks which are widely used in previous works. These tricks need manually adjustment which require lots of experiments, such as weight clipping and weight shifting.

We modify the original Lloyd quantizer [Lloyd, 1982] in an aggressive manner to obtain a closed-form solution for our quantization problem. We minimize the squared error between the original and quantized weights. Each binary value is constrained to only two possible values (-1 and +1). The binary weights of the i-th kernel in a convolutional layer are all assigned a shared floating-point value. The solution for the binary values is the sign of the original weight. The solution for the floating-point value is the average of the absolute values of the original weights in the i-th kernel. The quantized weight in one bit plane is the binary value multiplied by the floating-point value. We subtract the quantized weight of one bit plane from the original weight and quantize the result again. This procedure creates one bit plane in one time and the number of bit planes equals to the times it runs.

Finally, we fine-tune the quantized network to reduce the accuracy loss. The original weights are kept and updated during fine-tuning since most of the operators is differentiable. We directly propagate the gradient of sign operator backward following the previous methods [Hubara *et al.*, 2016]. However, after quantization, the internal covariate shift problem and pathological curvature problem become serious, especially when the networks going deeper. To overcome this problem, we propose dual normalization that is inspired by batch normalization [Ioffe and Szegedy, 2015] and weight normalization [Salimans and Kingma, 2016]. We normalize both the weights and activations along channel axis and kernel axis respectively. The experiments show its efficiency.

Comparing with other quantization methods, our quantized networks achieve higher accuracy than theirs with the same number of bits, and the results are even close to those of their floating-point counterparts. Although sometimes our networks need more addition operations than the original network, adders are cheaper than multipliers in hardware, which guarantees promising application prospects for our method. Still, our networks require far less storage than the original networks. Contributions of this paper are summarized as follows:

(1) We propose a more accurate quantization algorithm that quantizes each kernel separately to multiple bit planes. We still utilize fractional arithmetic which has advantages in handling complex distributions of values and simultaneously eases the high demand of running devices.

(2) We find a closed-form solution for our quantization problem by modifying the Lloyd quantizer. It is compatible with fine-tuning which can further improve the performance.

(3) Moreover, we propose dual normalization, which solves the pathological curvature problem in our experiments. It does not require manual adjustments and thus will facilitate the adaptation to other methods.

2 Related Work

Previous works, which accelerated CNNs, could be divided into weights-based methods and quantization methods. They decreased computational power demand by modifying the convolutional layers.

Weights-based methods directly eliminate the weights in convolutional layers [Han *et al.*, 2015], such as matrix decomposition [Kim *et al.*, 2015], low-rank decomposition [Tai *et al.*, 2015], pruning [Han *et al.*, 2015], and sparsification [Ioannou *et al.*, 2017; Figurnov *et al.*, 2016]. EEC [Yang *et al.*, 2017] proposed an energy-aware pruning algorithm for CNNs that directly used the energy consumption of a CNN to guide the pruning process. NISP [Yu *et al.*, 2018] applied feature ranking techniques to measure the importance of each neuron in the final response layer, formulating network pruning as a binary integer optimization problem, and derived a closed-form solution to it for pruning neurons in earlier layers. Recently, some methods directly trained the computationally efficient CNNs with delicate structures, such as and ShuffleNet [Zhang *et al.*, 2018]. Weights-based methods achieve lower acceleration performance comparing with quantization methods. However, they are compatible with current devices, such as GPU, which guarantees excellent application prospects for them. Besides, weights-based methods and quantization methods utilize different kinds of redundancy [Zeng and Tian, 2018]. They are orthogonal with each other.

Quantization methods [Dong *et al.*, 2017] reduce the number of bits of weights and activations. They achieve great theoretical acceleration comparing with weights-based methods. However, quantization methods make various modifications on the framework of CNNs which needs hardware support. Hubara et al. (BNN) [Hubara *et al.*, 2016] used binary weights and activations to compute the parameter gradients to train the networks at training time. At running time, both weights and activations were binary and CNNs were accelerated by replacing fractional arithmetics with bit-wise arithmetics. Rastegar et al. (Xnor) [Rastegari *et al.*, 2016] quantized the filters and the inputs of the convolutional layers to binary. Lin et al. (ABC) [Lin *et al.*, 2017] approximated full-precision weights with linear combinations of multiple binary weight bases. Multiple binary activations were also employed to alleviate information loss.

Gupta et al. [Gupta *et al.*, 2015] showed that deep networks can be accelerated using low bit wide fixed-point number representation. Lin et al. [Lin *et al.*, 2016] proposed a quantizer designed for fixed point implementations of CNNs. Leng et al. [Leng *et al.*, 2018] modeled this problem as a discretely constrained optimization problem and borrowed the idea from Alternating Direction Method of Multipliers (ADMM). Wu et al. [Wu *et al.*, 2018] developed a new method to discretize both training and inference, where weights, activations, gradients and errors among layers are shifted and linearly constrained to low-bitwidth integers.

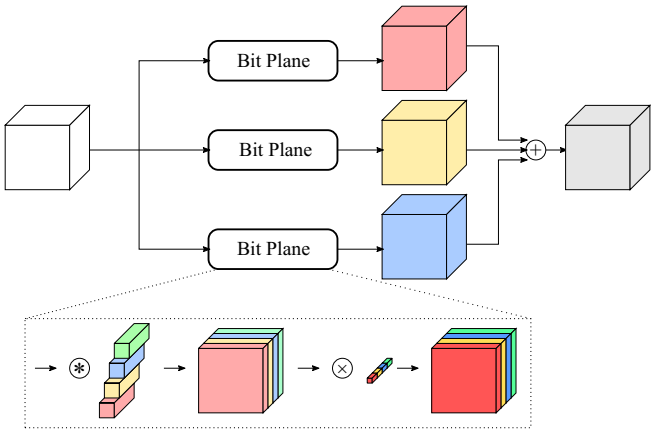


Figure 1: The structure of the quantized convolution process. Each Bit Plane has the structure shown in Equation 4. Here the $*$ represent binary convolution shown in Equation 2. The $\times$ represent multiplications with broadcasting as shown in Equation 4.

They further replaced batch normalization by a constant scaling layer and simplified other components that are arduous for integer implementation.

Courbariaux et al. (BWN) [Courbariaux *et al.*, 2015] accelerated the CNNs by quantizing only the weights to eliminate the multiplications. Li et al. (TWN) [Li *et al.*, 2016] constrained ternary weights to +1, 0 and -1 and the Euclidean distance between the full-precision weights and the ternary weights was minimized, up to a scaling factor. However, BWN and TWN suffered from huge accuracy loss. But we still follow the scheme of BWN and TWN, since it uses fractional arithmetic which is supported by current devices. This scheme is able to be realized with software development and further improved with the hardware support. Our method improve the performance of this scheme which makes it efficient and practical in industrial applications.

3 Convolutional Layer Quantization

In this section, we detail the structure of our bit planes and show how we quantize convolution operations. In Section 3.1, we propose our kernel-wise quantization structure with bit planes. This structure is used for the deployment on devices. In Section 3.2, we illustrate how we obtain the bit planes from the floating-point weights. Based on the distilling theory [Hinton *et al.*, 2015], we firstly train convolutional networks and then quantize the trained networks. It is widely adopted by previous works [Courbariaux *et al.*, 2015; Lin *et al.*, 2017].

3.1 Kernel-wise Quantization

First, we discuss the case of one bit plane. Formally, the convolution process takes a stack of feature maps as input, which can be represented as a 3-D tensor denoted by $\boldsymbol{X} \in \mathbb{R}^{c_{in} \times h \times w}$, where h and w are the height and width, respectively, of the feature maps and c_{in} is the number of input channels. The weights in a convolutional layer with c_{out} kernels are denoted by $\boldsymbol{W} \in \mathbb{R}^{c_{out} \times c_{in} \times d \times d}$, where d is the kernel size. The output of the convolution, $\boldsymbol{Y} \in \mathbb{R}^{c_{out} \times h \times w}$, is

$$\boldsymbol{Y}_{k,i,j} = \sum_{v,m,n=1,1,1}^{c_{in},d,d} \boldsymbol{X}_{v,i+m-p,j+n-p} \boldsymbol{W}_{k,v,m,n}, \quad (1)$$

In this equation, we assume, without loss of generality, that the stride is one and that the padding size is $p = (d-1)/2$.

At testing time, for one bit plane for quantized convolution, each weight $\boldsymbol{W}_{k,v,m,n}$ in Equation 1 is quantized to either -1 or +1, denoted by $\boldsymbol{B}_{k,v,m,n} \in \{-1, +1\}^{c_{out} \times c_{in} \times d \times d}$. These two values are significantly advantageous from a hardware perspective. With this quantization, the multiplication in Equation 1 can be replaced with the sign operation. The sign of the activation $\boldsymbol{X}_{v,i+m-p,j+n-p}$ in Equation 1 is reversed when the weight $\boldsymbol{B}_{k,v,m,n}$ is -1 and remains unchanged when the weight is 1. The computational cost of the sign function is extremely low. Consequently, the convolution process degenerates to a process of accumulation:

$$\boldsymbol{Y}_{k,i,j} = \sum_{v,m,n=1,1,1}^{c_{in},d,d} f(\boldsymbol{X}_{v,i+m-p,j+n-p}, \boldsymbol{B}_{k,v,m,n}) \quad (2)$$

The definition of $f(x, y)$ is

$$f(x,y) = \begin{cases} x, & y = 1 \\ -x, & y = -1 \end{cases} \quad (3)$$

However, directly quantizing the weights to one bit plane will considerably reduce the accuracy of the network [Hubara *et al.*, 2016; Courbariaux *et al.*, 2015]. To overcome this problem, an intuitive solution is to utilize more bit planes to achieve more accurate quantization [Lin *et al.*, 2017]. Besides, in previous quantization works, the kernels of a single convolutional layer have been quantized as a whole. However, the convolution of each kernel is independent and unique. Quantizing all kernels in a convolutional layer as a whole is not an suitable approach. Instead, we quantize each kernel separately. The real-valued weight $\boldsymbol{W}_{k,v,m,n}$ is estimated using a linear combination of t bit planes $\boldsymbol{B}^1, \boldsymbol{B}^2, \ldots, \boldsymbol{B}^t \in \{-1, +1\}^{c_{out} \times c_{in} \times d \times d}$ and a floating-point vector $\boldsymbol{\alpha} \in R^{t \times c_{out}}$:

$$\begin{aligned} \boldsymbol{Y}_{k,i,j} &= \sum_{s=1}^{t} \boldsymbol{\alpha}_k^s \sum_{v,m,n=1,1,1}^{c_{in},d,d} f(\boldsymbol{X}_{v,i+m-p,j+n-p}, \boldsymbol{B}_{k,v,m,n}^s) \\ &= \sum_{s=1}^{t} \boldsymbol{\alpha}_k^s \boldsymbol{Y}_{k,i,j}^s \end{aligned} \quad (4)$$

The structure of the quantized convolutional layer that replaces the original convolutional layer according to Equation 4 is shown in Figure 1. We will present how to obtain $\boldsymbol{B}^1, \boldsymbol{B}^2, \ldots, \boldsymbol{B}^t$ and $\boldsymbol{\alpha}$ in Section 3.2.

Since we use multiple bit planes, our method need nearly t times addition operations than the original convolution operation. However, multiplication is much harder to be implemented on devices than addition is. Here, we analyze the case of one addition and one multiplication, without loss of generality. The carry adder needs $k_1 n$ transistors, where k_1 is

a coefficient and n is the number of bits in the original convolution. The array multiplier needs $k_2n^2 + k_3n$ transistors. Usually, $k_2 > k_1$, and $k_3 > k_1$. In the floating-point case, the number of transistors required is complicated to analyze, but the order of magnitude will remain consistent. The original convolution operation needs approximately $k_2n^2+(k_1+k_3)n$ transistors. The quantized convolution and scaling operations need approximately $\epsilon k_2n^2 + ((t + \epsilon)k_1 + \epsilon k_3)n$. Since ϵ is very small and $t \ll n, t \ll k_2$ usually holds, the quantized convolution and scaling operations need far fewer transistors than the original convolution operation does.

3.2 Solution to the Lloyd Algorithm

In computer science and electrical engineering, Lloyd's algorithm, also known as Voronoi iteration or relaxation, is used to find evenly spaced sets of points in subsets of Euclidean spaces and partitions of these subsets into well-shaped and uniformly sized convex cells [Lloyd, 1982]. In our algorithm, we want to find $\boldsymbol{B}^1, \boldsymbol{B}^2, \ldots, \boldsymbol{B}^t$ and $\boldsymbol{\alpha}$ in Equation 4. We solve the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \boldsymbol{B}^{1,\ldots,t}} J = & \sum_{o,v,m,n=1,1,1,1}^{c_{out},c_{in},d,d} (\boldsymbol{W}_{o,v,m,n} - \sum_{s=1}^{t} \boldsymbol{\alpha}_o^s \boldsymbol{B}_{o,v,m,n}^s)^2 \\ s.t. \quad & \boldsymbol{B}_{o,v,m,n}^s \in \{-1, +1\}^{c_{out} \times c_{in} \times d \times d} \end{aligned} \tag{5}$$

However, this problem is very hard to be solved. Inspired by differential pulse-code modulation (DPCM) [Cutler, 1952], we find the sub-optimal solution by sequentially solving the following problem:

$$\min_{\boldsymbol{\alpha}, \boldsymbol{B}^s} J = \sum_{o,v,m,n=1,1,1,1}^{c_{out},c_{in},d,d} (\boldsymbol{W}_{o,v,m,n}^{s-1} - \boldsymbol{\alpha}_o^s \boldsymbol{B}_{o,v,m,n}^s)^2 \tag{6}$$

where $s = 1, \ldots, t$ and

$$\boldsymbol{W}_{k,l,i,j}^s = \begin{cases} \boldsymbol{W}_{k,l,i,j}^{s-1} - \boldsymbol{\alpha}_k^s \boldsymbol{B}_{i,j,k,l}^s & s > 0 \\ \boldsymbol{W}_{k,l,i,j} & s = 0 \end{cases} \tag{7}$$

According to the solution of Lloyd's algorithm, a local minimum solution can be obtained for Equation 6. For a fixed $\boldsymbol{\alpha^s}$, we can obtain the numerical solution for B^s by computing the Voronoi diagram of the -1 and +1 sites:

$$\boldsymbol{B}_{k,l,i,j}^s = sign(\boldsymbol{W}_{k,l,i,j}^{s-1}) \tag{8}$$

The definition of $sign(x)$ is

$$sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases} \tag{9}$$

Here, weights of zero are quantized to +1. In practice, there are only a minute number of zero weights, which have a negligible influence on the final results of the whole network. Moreover, we adopt fine-tuning in our algorithm, which further reduces the number of zero weights.

However, we note that the solution for $\boldsymbol{B^s}$ is not related to $\boldsymbol{\alpha^s}$, which means that the solution for $\boldsymbol{B^s}$ remains consistent throughout all iterations. Thus, Equation 6 becomes

$$\min_{\boldsymbol{\alpha}, \boldsymbol{B}^s} J = \sum_{o,v,m,n=1,1,1,1}^{c_{out},c_{in},d,d} (|\boldsymbol{W}_{o,v,m,n}^{s-1}| - \boldsymbol{\alpha}_o^s)^2 \tag{10}$$

It is quite easy to find the solution:

$$\boldsymbol{\alpha}_k^s = \frac{1}{c_{in} \times h \times w} \sum_{v,m,n=1,1,1}^{c_{in},d,d} |\boldsymbol{W}_{k,v,m,n}^{s-1}| \tag{11}$$

Now, we obtain the sub-optimal solution for Equation 5. We conduct the quantization in an aggressive manner. Since it is a sub-optimal solution, we conduct fine-tuning after quantization which considerably reducing negative influences for unable to find the optimal solution.

4 Bit Plane Optimization

In this section, we introduce the techniques used in our fine-tuning stage. In Section 4.1, we introduce the dual normalization and analyze the possible reasons why it works. In Section 4.2, we describe the details for our quantized convolutional layer during fine-tuning.

4.1 Dual Normalization

In an effort to improve the convergence of the optimization procedure, we adopt dual normalization. It contains two separable procedures, batch normalization and weight normalization. The batch normalization is same as original batch normalization [Ioffe and Szegedy, 2015] which normalizes the v axis of $\boldsymbol{X}$ in Equation 1. The weight normalization skips the learnable parameters in batch normalization. Specifically, weight normalization normalizes the weights on the v axis of $\boldsymbol{W}$ in Equation 1. The corresponding axes of $\boldsymbol{X}$ and $\boldsymbol{W}$ are normalized, termed dual normalization.

We want to solve internal covariate shift problem and pathological curvature problem simultaneously. The distribution of each layer's inputs changes during training as the parameters of the previous layers change. It slows down training by requiring lower learning rates and careful parameter initialization, and makes it notoriously difficult to train models with saturated nonlinearities. It is called internal covariate shift problem. Normalizing the activations guarantees each layer's inputs have a fixed distribution, thus accelerates the convergence. The gradient descent algorithm is strongly influenced by the curvature of the objective. If the Hessian matrix of the objective at the optimum point has a low condition number, then gradient descent will get bogged down [Sutskever *et al.*, 2013]. This phenomenon is called the pathological curvature problem. There may be multiple equivalent ways of parameterizing the same model, and some of them are much easier to optimize than others [Salimans and Kingma, 2016]. From the intuitive perspective, weight normalization forces each input channel to be used in the convolution which makes the set of normalized weights easier to be optimized than others.

4.2 Forward and Backward Propagation in Fine-tuning

Here, we discuss the case of one convolutional layer. The algorithm is shown in Algorithm 1. We fine-tune the quantized convolution on the training data. In the fine-tuning process, the real-valued weights $\boldsymbol{W}$ of the original convolutional layer are preserved. The quantized weights $\boldsymbol{B}^1, \boldsymbol{B}^2, \ldots, \boldsymbol{B}^t$ and $\boldsymbol{\alpha}$

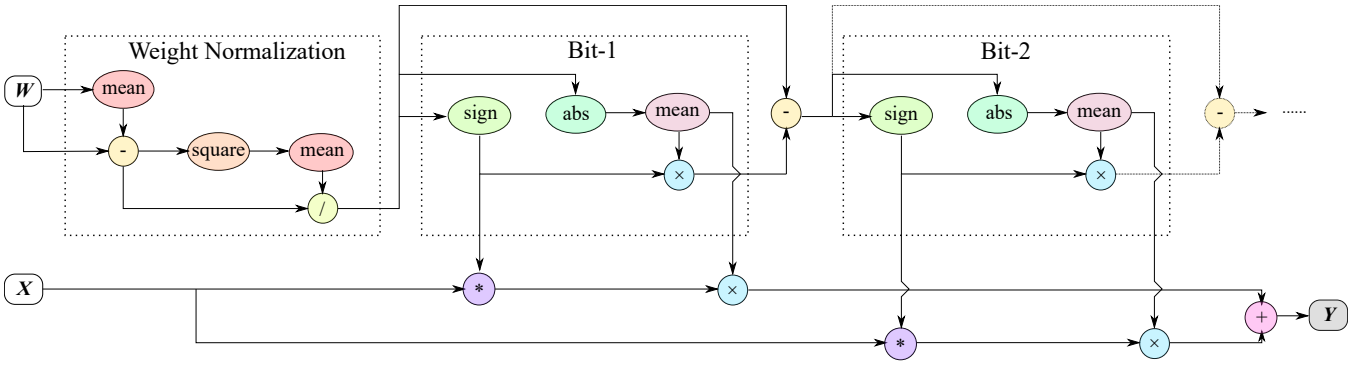


Figure 2: This figure lists the operators used in the fine-tuning. Here we show the 2-bit situation. We can use Tensorflow to implement it. Here the $\boldsymbol{W}$ represents the original weights of the convolutional layer. The $\boldsymbol{X}$ is the input feature maps and $\boldsymbol{Y}$ is the output of quantized convolution. The mean operator computes the average of input along one axis, which is the c_{in} axis in the block of Weight Normalization as marked in the graph and c_{out} axis in the block of each Bit of the rest part in the graph. The square operator computes the square for each value of the input. The sign operator is shown in Equation 9. The abs operator computes the absolute value for each value of input. The $*$ operator is shown in Equation 2. The $+$, $-$, $\times$ and $/$ represent addition, subtract, multiplication and division with broadcasting respectively.

Algorithm 1 Forward Quantization of One Convolutional Layer

Input: The input features from the last layer, $\boldsymbol{X}$; The real-valued weights of the original convolutional layer, $\boldsymbol{W}$; The number of bits used for quantization, t;
Output: The output of the quantized convolution, $\boldsymbol{Y}$

(optional) Normalize the activations $\boldsymbol{X}$ along kernel axis;
(optional) Normalize the weights $\boldsymbol{W}$ along channel axis;
$\boldsymbol{W}^0 = \boldsymbol{W}$;
for $i = [1, t]$ **do**
    Compute $\boldsymbol{B}^i$ with $\boldsymbol{W}^{i-1}$ according to Equation 8;
    Compute $\boldsymbol{\alpha}^i$ with $\boldsymbol{W}^{i-1}$ according to Equation 11;
    Compute $\boldsymbol{W}^i$ according to Equation 7;
end for
$\boldsymbol{Y} = 0$;
for $i = [1, t]$ **do**
    Compute $\boldsymbol{Y}^i$ with $\boldsymbol{X}$, $\boldsymbol{B}^i$ and $\boldsymbol{\alpha_i}$ according to Equation 4;
    $\boldsymbol{Y} = \boldsymbol{Y} + \boldsymbol{Y}^i$;
end for
return $\boldsymbol{Y}$;

are not kept since they are generated by $\boldsymbol{W}$. The operator graph of fine-tuning is shown in Figure 2. We aim to update the $\boldsymbol{W}$ with SGD [Goodfellow *et al.*, 2016] method.

However, all operators are differentiable except for sign operator. The derivation of the sign function is zero almost everywhere, making it apparently incompatible with backward propagation. Following Hubara et al. [Hubara *et al.*, 2016], we directly propagate the gradient without any modification. This is equivalent to using the following gradient:

$$\frac{\partial sign(x)}{\partial x} = 1 \tag{12}$$

This technique was first proposed by Hinton et al. [Hinton *et al.*, 2012] and has been utilized in numerous subsequent works [Hubara *et al.*, 2016; Lin *et al.*, 2017].

After deployment, $\boldsymbol{B}^1, \boldsymbol{B}^2, \ldots, \boldsymbol{B}^t$ and $\boldsymbol{\alpha}$ remain consistent. The quantization procedure is already complete before forward propagation is performed. In particularly, only Steps 9 to 14 in Algorithm 1 are performed, and the operators $*$ and $\times$ of the lower part of Figure 2 are active.

5 Experiments

To validate the effectiveness of our algorithm, we conducted experiments on ImageNet [Russakovsky *et al.*, 2015] dataset. The AlexNet [Krizhevsky *et al.*, 2012] structure is mostly used by weights-based methods. On the other hand, the ResNet-18 [He *et al.*, 2016] structure is mostly used by quantization methods. So we utilize two network structures to validate the effectiveness of our proposed method. The experiments were implemented with TensorFlow [Abadi *et al.*, 2016] at version 1.3.

5.1 Implementation Details

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [Russakovsky *et al.*, 2015] evaluates algorithms for object detection and image classification at a large scale. The training set, which is a subset of ImageNet, contains 1.2 million images which can be divided into 1000 categories. The validation set contains 50000 images. The data preprocessing generally follows the implementation of Tensorflow [Abadi *et al.*, 2016]:

(1) Converting the data type of image from uint8 to float32.

(2) Each pixel is divided by 128 and subtracted 1.

(3) Resizing each image to 256×256.

(4) Randomly cropping a 227×227 patch.

(5) Flipping each patch with a probability of 0.5.

But the data processing on ResNet-18 [He *et al.*, 2016] uses multiple scales additionally. We thus randomly choose size 224, 256, 288, 320 and 352 to resize the image before cropping a 224×224 patch.

AlexNet contains 14.29MB floating-point weights belonging to convolutional layers. It costs 1.08G floating-point multiplications and additions for convolution. Equally, ResNet-18 contains 42.60MB floating-point weights and requires 1.81G floating-point multiplications and additions for convolution.

We firstly fine-tuned the pre-trained models from the model zoo before quantization. For AlexNet we fine-tuned the network for 40 epochs. The learning rate was initially set to 0.01, and decayed by a factor of 1/10 up to the 20th epoch and by another factor of 1/10 up to the 30th epoch. The L2-norm regularization with a factor of $5e^{-4}$ was adopted was adopted. The batch size was set to 256 in these experiments. For ResNet-18, the settings were similar except that the L2-norm regularization was adopted with a factor of $1e^{-4}$. We noticed that the networks had to be fine-tuned successively for several times to get the accuracy as reported by the authors.

During the fine-tuning process in Section 4.2, the momentum was set to 0.9. We fine-tuned the network for 20 epochs. The learning rate was initially set to 0.001 and decayed by a factor of 1/10 up to the 10th epoch and by another factor of 1/10 up to the 15th epoch. L2-norm regularization with a factor of $1e^{-4}$ was adopted. The batch size was set to 256 in these experiments.

5.2 Comparing with Previous Methods

The experimental results are shown in Table 1 and Table 2. Here, we adopt the same evaluation criteria as previous works [Figurnov *et al.*, 2016; Courbariaux *et al.*, 2015]. The column titled "Bits" reports the number of bits used for quantizing weights. The column titled "Top-1 Bef." reports the top-1 accuracy before compression, as described in previous works. The column titled "Top-1 Aft." presents the top-1 accuracy after compression. The column titled "Top-1 ↓" presents the top-1 accuracy loss after compression. The analogous results for the top-5 accuracy are reported similarly. The column titled "Mult.↓" shows the reduction in the number of multiplications. The column titled "Add.↓" similarly reports the addition reduction. The column titled "Weight↓" shows the reduction in the number of weights.

AlexNet

The experimental results for AlexNet are shown in Table 1. Comparing with the traditional weights-based methods, such as Pruning [Han *et al.*, 2015], Sparsification [Figurnov *et al.*, 2016], Decomposition [Kim *et al.*, 2015] and Low-rank [Tai *et al.*, 2015], our method which is labeled as KCNN achieves far higher multiplication reduction and somewhat higher weight reduction with a lower or comparable accuracy loss. It shows our advantages in decreasing multiplications. Although our method requires more additions, it still incurs a lower computational cost because floating-point multiplications are much harder to be implemented on devices than additions, as shown in Section 3.1. Comparing with recent weights-based methods, which are EEC [Yang *et al.*, 2017] and NISP [Yu *et al.*, 2018], we still have these advantages when more bits are used.

The weights-based methods are orthogonal with our method. So we choose to combine Low-rank method [Tai *et al.*, 2015] with our method because it achieves the highest compression rate with little accuracy loss. From the results which are labeled as KCNN + Low-rank, we find that after combining two methods, the Mult.↓ and accuracy slightly decrease but the Add.↓ and weight↓ considerably increase. Our method achieves extremely high Mult.↓ and Weight↓, and little Add.↓ with little accuracy loss. We also find that our method achieves better results than BWN [Courbariaux *et al.*, 2015] at one bit, because we quantize each kernel separately. It is also the same reason that we achieve better results than ABC [Lin *et al.*, 2017].

ResNet-18

The experimental results for ResNet-18 are shown in Table 2. Quantization methods achieve nearly the same multiplication, addition and weight reductions with the same number of bits for weights. We find that the results of ABC [Lin *et al.*, 2017] and our KCNN are better than Xnor [Rastegari *et al.*, 2016] and TWN [Li *et al.*, 2016] with one or two bits, while the results of ABC and our KCNN are close. So we specially analyze these two methods.

With one bit, we achieve only an accuracy loss comparable to that of ABC [Lin *et al.*, 2017]. We believe that there are two reasons related to this phenomenon:

(1) With one bit plane, our algorithm is equivalent to multiplying each output feature map by a factor of $\boldsymbol{\alpha}_i$. However, the batch normalization whitens each output feature map, which offsets our efforts. With two and three bits, our method begins to show its advantages.

(2) To make our algorithm more elegant, we abandon the complex tricks which are widely used in previous works. These tricks need manually adjustment which require lots of experiments, such as weight clipping and weight shifting. It causes performance degradations when weight bits are not enough.

We achieve an accuracy higher than that of ABC when more bits are used. With five bits, our method achieves a negligible accuracy loss that is better than that of ABC. In summary, our method achieves better results than the previous methods do.

5.3 Ablation Study

In this section, we analyze the influence of kernel-wise quantization and dual normalization on AlexNet and ResNet-18. The results are shown in Figure 3.

The results with kernel-wise quantization are better than those without. Quantizing each kernel separately is more accurate than quantizing the whole layer together, because each kernel of each bit plane owned an individual α parameter. At one bit, kernel-wise quantization significantly improve the performance. With more bits used, the gain gradually decreases, since the information in multiple bit planes becomes richer and more α parameters bring less information gain. But our method still outperforms other methods.

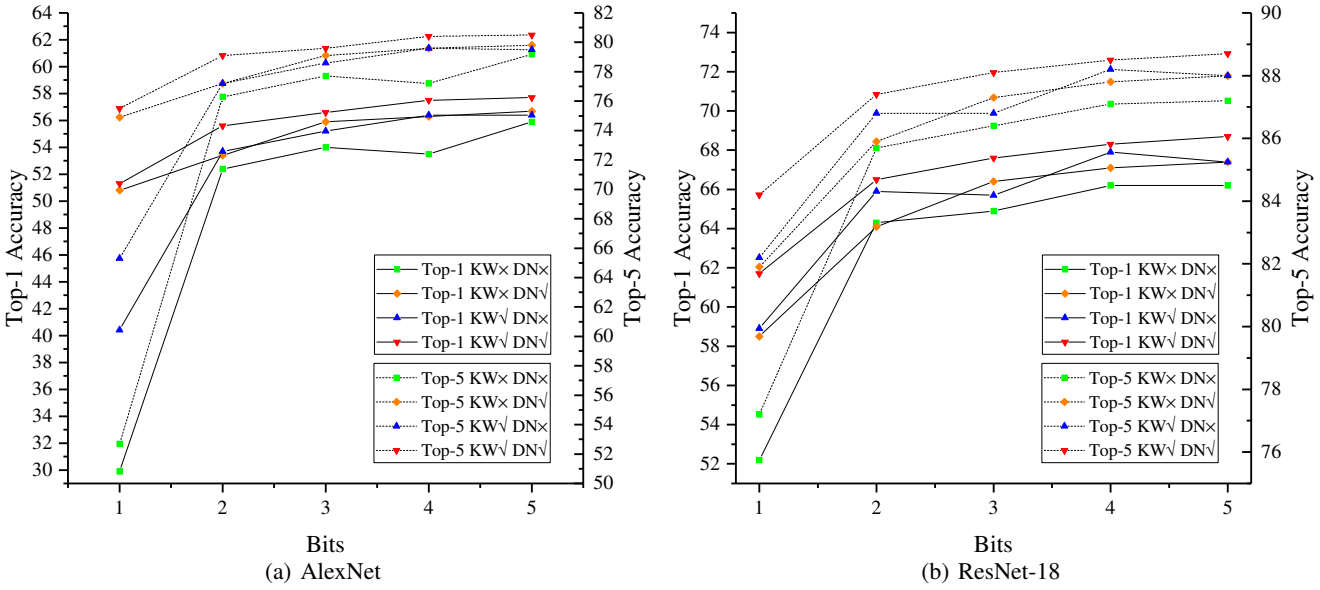
Then we analyze the influence of dual normalization. In general, the results with dual normalization are better than those without. The fluctuations of the results without dual normalization are more intense. We believe that the pathological curvature problem is more salient when dual normaliza-

Algorithm	Bits	Top-1 Bef.	Top-1 Aft.	Top-1 ↓	Top-5 Bef.	Top-5 Aft.	Top-5 ↓	Mult.↓	Add.↓	Weight↓
Pruning [Han *et al.*, 2015]	-	57.2%	57.2%	0.0%	80.3%	80.3%	0.0%	3×	3×	3×
Sparsification [Figurnov *et al.*, 2016]	-	-	-	-	80.4%	70.5%	9.9%	4.4×	4.4×	-
	-	-	-	-		74.3%	6.1%	3.5×	3.5×	-
	-	-	-	-		78.1%	2.3%	2.1×	2.1×	-
Low-rank [Tai *et al.*, 2015]	-	-	-	-	80.0%	79.6%	0.4%	5.27×	5.27×	5.00×
Decomposition [Kim *et al.*, 2015]	-	-	-	-	80.0%	78.3%	1.7%	2.67×	2.67×	5.46×
EEC [Yang *et al.*, 2017]	-	-	-	-	80.0%	79.5%	0.5%	6.66×	6.66×	11×
NISP [Yu *et al.*, 2018]	-	-	-	-	80.0%	80.0%	0.0%	2.5×	2.5×	2.1×
BWN* [Courbariaux *et al.*, 2015]	1	56.6%	29.9%	26.7%	80.0%	52.7%	37.3%	1656×	1.0×	30.6×
ABC* [Lin *et al.*, 2017]	2	56.6%	52.4%	4.2%	80.0%	76.3%	3.7%	828×	0.49×	15.81×
	3		54.0%	2.6%		77.7%	2.3%	552×	0.32×	10.54×
	4		53.5%	3.1%		77.2%	2.8%	414×	0.24×	7.90×
	5		55.9%	0.7%		79.2%	0.8%	331×	0.19×	6.32×
KCNN	1	56.6%	40.4%	16.2%	80.0%	65.3%	14.7%	1656×	1.0×	30.6×
	2		53.7%	2.9%		77.2%	2.8%	828×	0.49×	15.81×
	3		55.2%	1.4%		78.6%	1.4%	552×	0.32×	10.54×
	4		56.4%	0.2%		79.6%	0.4%	414×	0.24×	7.90×
	5		56.4%	0.2%		79.5%	0.5%	331×	0.19×	6.32×
KCNN + Low-rank	1	56.6%	37.9%	18.7%	80.0%	62.2%	17.8%	1434×	5.27×	149.6×
	2		51.8%	4.8%		75.7%	4.3%	717×	2.62×	74.8×
	3		53.6%	3.0%		77.1%	2.9%	478×	1.74×	49.8×
	4		55.6%	1.0%		78.8%	1.2%	358×	1.30×	37.4×
	5		56.2%	0.4%		79.5%	0.5%	286×	1.03×	29.9×

Table 1: The comparison between our proposed KCNN and previous methods on AlexNet.
*The BWN and ABC are realized by us.

Algorithm	Bits	Top-1 Bef.	Top-1 Aft.	Top-1 ↓	Top-5 Bef.	Top-5 Aft.	Top-5 ↓	Mult.↓	Add.↓	Weight↓
BWN [Courbariaux *et al.*, 2015]	1	69.3%	60.8%	8.5%	89.2%	83.0%	6.2%	730×	1×	31.5×
TWN [Li *et al.*, 2016]	2	-	61.8%	-	-	84.2%	-	730×	1×	31.5×
ABC [Lin *et al.*, 2017]	1	69.3%	62.8%	6.5%	89.2%	84.4%	4.8%	730×	1×	31.5×
	2		63.7%	5.6%		85.2%	4.0%	365×	0.49×	15.7×
	3		66.2%	3.1%		86.7%	2.5%	243×	0.32×	10.5×
	5		68.3%	1.0%		87.9%	1.3%	146×	0.19×	6.3×
KCNN	1	69.2%	61.7%	7.5%	89.0%	84.2%	4.8%	730×	1×	31.5×
	2		66.5%	2.7%		87.4%	1.6%	365×	0.49×	15.7×
	3		67.6%	1.6%		88.1%	0.9%	243×	0.32×	10.5×
	4		68.3%	0.9%		88.5%	0.5%	182×	0.24×	7.8×
	5		68.7%	0.5%		88.7%	0.3%	146×	0.19×	6.3×

Table 2: The comparison between our proposed KCNN and previous methods on ResNet-18.



(a) AlexNet (b) ResNet-18

Figure 3: The influence of kernel-wise quantization dual normalization in our proposed KCNN on AlexNet and ResNet-18. Here the KW and DN represent kernel-wise quantization and dual normalization respectively. The $\surd$ and $\times$ represent with and without respectively.

tion is not used. This problem prevents the optimization procedure from reaching a good local minimum and makes the algorithm highly rely on initialization. The dual normalization procedure can considerably compensate for the effect of the pathological curvature problem and thus facilitate the convergence process. The gain with dual normalization is more significant on ResNet-18 than AlexNet, because the ResNet-18 is deeper in which the pathological curvature problem is more salient.

6 Conclusion

In this paper, we reduce the number of multiplications in CNNs by quantizing the floating-point weights in the covolutional layer to multiple bit planes with binary weights (+1 and -1) and corresponding floating-point scale values. We quantize each kernel separately with aggressive Lloyd's algorithm. Fine-tuning is adopted to optimize the weights after quantization. To facilitate convergence during fine-tuning, we propose a dual normalization procedure, which considerably improves the convergence performance. Our designed networks show only negligible performance loss compared with their floating-point counterparts. Comparisons with previous weights-based algorithms show that our proposed method achieves better results than previous state-of-the-art methods in terms of multiplication reduction. Our method also performs better than other quantization methods with the same number of weight bits. After combined with weights-based methods, our method is able to reduce the multiplications without causing additions increasing.

Acknowledgments

This work was supported by NSFC projects 61872329 and 61572451.

References

[Abadi *et al.*, 2016] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[Courbariaux *et al.*, 2015] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[Cutler, 1952] Cassius C Cutler. Differential quantization of communication signals, July 29 1952. US Patent 2,605,361.

[David *et al.*, 2007] Jean Pierre David, Kassem Kalach, and Nicolas Tittley. Hardware complexity of modular multiplication and exponentiation. *IEEE Transactions on Computers*, 56(10):1308–1319, 2007.

[Dong *et al.*, 2017] Yinpeng Dong, Renkun Ni, Jianguo Li, Yurong Chen, Jun Zhu, and Hang Su. Learning accurate low-bit deep neural networks with stochastic quantization. *arXiv preprint arXiv:1708.01001*, 2017.

[Figurnov *et al.*, 2016] Mikhail Figurnov, Aizhan Ibraimova, Dmitry P Vetrov, and Pushmeet Kohli. Perforatedcnns: Acceleration through elimination of redundant convolutions. In *Advances in neural information processing systems*, pages 947–955, 2016.

[Goodfellow *et al.*, 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[Gupta *et al.*, 2015] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.

[Han *et al.*, 2015] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[Hinton *et al.*, 2012] Geoffrey Hinton, Nitsh Srivastava, and Kevin Swersky. Neural networks for machine learning. *Coursera, video lectures*, 264, 2012.

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

[Hubara *et al.*, 2016] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

[Ioannou *et al.*, 2017] Yani Ioannou, Duncan Robertson, Roberto Cipolla, and Antonio Criminisi. Deep roots: Improving cnn efficiency with hierarchical filter groups. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1231–1240, 2017.

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

[Kim *et al.*, 2015] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[Leng *et al.*, 2018] Cong Leng, Zesheng Dou, Hao Li, Shenghuo Zhu, and Rong Jin. Extremely low bit neural network: Squeeze the last bit out with admm. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

[Lin *et al.*, 2016] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

[Lin *et al.*, 2017] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, pages 344–352, 2017.

[Lloyd, 1982] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[Rastegari *et al.*, 2016] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[Russakovsky *et al.*, 2015] Olga Russakovsky, Jia Deng, Hao Su, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[Salimans and Kingma, 2016] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

[Sutskever *et al.*, 2013] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[Tai *et al.*, 2015] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

[Wolf, 2002] Wayne Wolf. *Modern VLSI design: system-on-chip design*. Pearson Education, 2002.

[Wu *et al.*, 2018] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680*, 2018.

[Yang *et al.*, 2017] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5687–5695, 2017.

[Yu *et al.*, 2018] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[Zeng and Tian, 2018] Linghua Zeng and Xinmei Tian. Accelerating convolutional neural networks by removing interspatial and interkernel redundancies. *IEEE transactions on cybernetics*, 2018.

[Zhang *et al.*, 2018] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

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