Training High-Performance and Large-Scale Deep Neural Networks with Full 8-bit Integers

Yukuan Yang[a], Shuang Wu[a], Lei Deng[b*], Tianyi Yan[c], Yuan Xie[b], Guoqi Li[a*]
[a]Department of Precision Instrument, Center for Brain Inspired Computing Research, Tsinghua University, Beijing 100084, China.
[b]Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA.
[c]School of Life Science, Beijing Institute of Technology, Beijing 100084, China.

***Abstract*—Deep neural network (DNN) quantization converting floating-point (FP) data in the network to integers (INT) is an effective way to shrink the model size for memory saving and simplify the operations for compute acceleration. Recently, researches on DNN quantization develop from inference to training, laying a foundation for the online training on accelerators. However, existing schemes leaving batch normalization (BN) untouched during training are mostly incomplete quantization that still adopts high precision FP in some parts of the data paths. Currently, there is no solution that can use only low bit-width INT data during the whole training process of large-scale DNNs with acceptable accuracy. In this work, through decomposing all the computation steps in DNNs and fusing three special quantization functions to satisfy the different precision requirements, we propose a unified complete quantization framework termed as "WAGEUBN" to quantize DNNs involving all data paths including W (Weights), A (Activation), G (Gradient), E (Error), U (Update), and BN. Moreover, the Momentum optimizer is also quantized to realize a completely quantized framework. Experiments on ResNet18/34/50 models demonstrate that WAGEUBN can achieve competitive accuracy on ImageNet dataset. For the first time, the study of quantization in large-scale DNNs is advanced to the full 8-bit INT level. In this way, all the operations in the training and inference can be bit-wise operations, pushing towards faster processing speed, decreased memory cost, and higher energy efficiency. Our throughout quantization framework has great potential for future efficient portable devices with online learning ability.**

Keywords: Neural Network Quantization, 8-bit Training, Full Quantization, Online Learning Device

I. INTRODUCTION

Deep neural networks [1] have achieved state-of-art results in many fields like image processing [2], object detection [3], natural language processing [4], and robotics [5] through learning high-level features from a large amount of input data. However, due to the existence of a huge number of floating-point (FP) values and complex FP multiply-accumulate operations (MACs) in the process of network training and inference, the intensive memory overhead, large computational complexity, and high energy consumption impede the wide deployment of deep learning models. DNN quantization [6] which converts FP MACs to bit-wise operations is an effective way to reduce the memory and computation costs and improve the speed of deep learning accelerators.

With the deepening of research, DNN quantization gradually transfers from inference optimization (BWN [7], XNOR-Net [8], ADMM [9]) to training optimization (DoReFa [10], GXNOR-Net [11], FP8 [12]). Usually, the inference quantization focuses on the forward pass; while the training quantization further quantizes the backward pass and weight update. Recently, the training quantization becomes a hot topic in the network compression community. Whereas, there are still two major issues in existing schemes. The first issue lies in the incomplete quantization, including two aspects: partial quantization and FP dependency. Partial quantization means that only parts of dataflow, not all of them, are quantized (e.g. DoReFa [10], GXNOR-Net [11] and QBP2 [13]); FP dependency still remains FP values during the training process (e.g. MP [14] and FP8 [12]). The second issue is that the quantization of batch normalization (BN) [15] is ignored by most schemes (e.g. MP-INT [16] and FX Training [17]). BN is an essential layer for the training of DNNs by addressing the problem of internal covariate shift of each layer's inputs, especially as the network deepens, allowing a much higher learning rate and less careful weight initialization.

Compared with all the studies above, WAGE [18] is the most thorough work of DNNs quantization, which quantizes the data including W (Weights), A (Activation), G (Gradient), E (Error), U (Update) and replacing each BN layer with a constant scaling factor. WAGE has achieved competitive results on LeNet [19], VGG [20], and AlexNet [21], providing a good inspiration for this work. However, we find that WAGE is difficult to be applied in large-scale DNNs due to the absence of BN layers. Besides, it is known that the gradient descent optimizer such as Momentum [22] or Adam [23] increases the stability and even helps get rid of the local optimum, thus the speed and final performance are significantly improved. A complete quantization should cover the entire training process, including W, A, G, E, U, BN, and the optimizer. Regretfully, up to now, there is still no such solution that can achieve this complete quantization, especially on large-scale DNNs.

To address the issues and realize a completely quantized training of large-scale DNNs, we propose a unified quantization framework termed "WAGEUBN" to constrain W, A, G, E, U, BN, and the optimizer in the low-bit integer (INT) space. All computation steps and operands in DNNs are decomposed and quantized. According to various data distributions in DNN training, we fuse three quantization functions to satisfy the different precision requirements. Furthermore, we find that 8-bit errors lose too much information and cause non-convergence due to the insufficient data coverage. To solve this problem, we propose a new storage and computing method by introducing a flag bit to expand the data coverage. Under the WAGEUBN

*Corresponding authors. Email addresses: yyk17@mails.tsinghua.edu.cn (Y. Yang), wus15@mails.tsinghua.edu.cn (S. Wu), leideng@ucsb.edu (L. Deng), yantianyi@bit.edu.cn (T. Yan), Y. Xie (yuanxie@ece.ucsb.edu) and liguoqi@mail.tsinghua.edu.cn (G. Li).

framework, all operations in the training and inference of DNNs can be simple bit-wise operations. WAGEUBN shows competitive accuracy and much less memory cost, training time, and energy consumption on ResNet18/34/50 over ImageNet [24] dataset. This work provides a feasible idea for the architecture design of future efficient online learning devices. The contributions of this work are twofold, which are summarized as follows:

- We address the two issues existing in most quantization schemes via fully quantizing all the data paths, including W, A, G, E, U, BN, and the optimizer, greatly reducing the memory and compute costs. What's more, we constrain the data to INT8 for the first time, pushing the training quantization to a new bit level compared with the existing FP16, INT16, and FP8 solutions.
- Our quantization framework is validated in large-scale DNN models (ResNet18/34/50) over ImageNet dataset and achieves competitive accuracy with much fewer overheads, indicating great potential for future portable devices with online learning ability.

The organization of this paper is as follows: Section II introduces the related work of DNN quantization; Section III details the WAGEUBN framework; Section IV presents the experiment results of WAGEUBN and the corresponding analyses; Section V summarizes this work and delivers the conclusion.

II. Related Work

With the wide applications of DNNs, the related compression technologies have been proposed rapidly, among which the quantization plays an important role. The development of DNN quantization can be divided into two stages, inference quantization and training quantization, according to the different quantization objects.

Inference quantization: Inference quantization starts from constraining W into $\{-1,\ 1\}$ (BWN [7]), replacing complex FP MACs with simple accumulations. BNN [25] and XNOR-Net [8] further quantize both W and A, making the inference computation dominated by bit-wise operations. However, extremely low bit-width quantization usually leads to significant accuracy loss. For example, when the bit width comes to <4 bits, the accuracy degradation becomes obvious, especially for large-scale DNNs. Instead, the bit width of W and A for inference quantization can be reduced to 8 bits with little accuracy degradation. The study of inference quantization is sufficient for the deep learning inference accelerators. Whereas, this is not enough for efficient online learning accelerators because only the data in the forward pass are considered.

Training quantization: To further extend the quantization towards the training stage, DoReFa [10] trains DNNs with low bit-width W, A, and G, while leaving E and BN unprocessed. MP [14] and MP-INT [16] use FP16 and INT16 values, respectively, to constrain W, A, and G. Recently, FP8 [12] further pushes W, A, G, E, and U to 8, 8, 8, 8, and 16-bit FP values, respectively, still leaving BN untouched. QBP2 [13] replaces the conventional BN with range BN and constrains W, A, and E to INT8 values, while calculating G with FP MACS. Recently, WAGE [18] adopts a layer-wise scaling factor instead of using the BN layer and quantizes W, A, G, E, and U to 2, 8, 8, 8, and 8 bits, respectively. Despite its thorough quantization, WAGE is difficult to be applied to large-scale DNNs due to the absence of powerful BN layers. In summary, there still lacks a complete INT8 quantization framework for the training of large-scale DNNs with high accuracy.

III. WAGEUBN Framework

The main idea of WAGEUBN is to quantize all the data in DNN training to INT8 values. In this section, we detail the WAGEUBN framework implemented in large-scale DNN models. The organization of this section is as follows: Subsection III-A and Subsection III-B describe the notations and quantization functions, respectively; Subsection III-C explains the specific quantization schemes for W, A, G, E, U, BN, and the Momentum optimizer, respectively; Subsection III-D goes through the overall implementation of WAGEUBN, including in both forward and backward passes; Subsection III-E summarizes the whole process and shows the pseudo codes.

A. Notations

Before introducing the WAGEUBN quantization framework formally, we need to define some notations. Considering the l-th layer of DNNs, we divide the forward pass of DNNs into four steps as described in Figure 1 (BN is divided into two steps: Normalization & Q_{BN} and Scale & Offset).

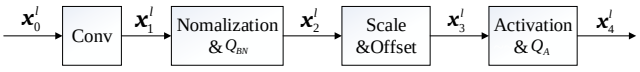


Fig. 1: Forward quantization of the l-th layer in DNNs.

Specifically, we have

$$\begin{aligned}
\text{Input}: &\quad \boldsymbol{x}_0^l = \boldsymbol{x}_4^{l-1} \\
\text{Conv}: &\quad \boldsymbol{x}_1^l = \boldsymbol{W}_q^l \boldsymbol{x}_0^l \\
\text{Normalization\&Q}_{\text{BN}}: &\quad \boldsymbol{x}_2^l = Q_{BN}(\frac{\boldsymbol{x}_1^l - \mu_q^l}{\sigma_q^l}) \quad , \\
\text{Scale\&Offset}: &\quad \boldsymbol{x}_3^l = \boldsymbol{\gamma}_q^l \boldsymbol{x}_2^l + \boldsymbol{\beta}_q^l \\
\text{Activation\&Q}_{\text{A}}: &\quad \boldsymbol{x}_4^l = Q_A\left[relu(\boldsymbol{x}_3^l)\right]
\end{aligned} \tag{1}$$

where $\boldsymbol{x}_4^{l-1}$ is the output of the $(l-1)$-th layer and $\boldsymbol{x}_0^l$ is the input of the l-th layer; $\boldsymbol{W}_q^l$ is the quantized weight for convolution; Q_{BN} is the quantization function used for constraining $\boldsymbol{x}_2^l$ to low-bit INT values, which will be given in Equation (12); μ_q^l and σ_q^l are the quantized mean and standard deviation value of one mini-batch; $\boldsymbol{\gamma}_q^l$ and $\boldsymbol{\beta}_q^l$ are the quantized scale and bias used in the BN layer of DNNs; Q_A is the quantization function for activation which will be detailed in Equation (13); $relu$ is the activation function which is commonly used in DNNs. Noting that every step in the forward pass is quantized, $\boldsymbol{x}_0^l, \boldsymbol{x}_1^l, \boldsymbol{x}_2^l, \boldsymbol{x}_3^l, \boldsymbol{x}_4^l$ are all integers. In order to make the notations used in the paper consistent, we make the following rules: subscript q denotes the data which

Fig. 2: Backward quantization of the l-th layer in DNNs.

have been quantized to INT values and superscript l denote the layer index.

Different from most existing schemes, we define $\boldsymbol{e}$ and $\boldsymbol{g}$ respectively, where $\boldsymbol{e}$ represents the gradient of A (activation) which is used in the error backpropagation and $\boldsymbol{g}$ represents the gradient of W (weights) which is used in the weight update. Moreover, we quantize the BN layers, including both the forward and backward passes, which is not well touched in most prior work. Similar to the forward pass, we divide the backward pass of the l-th layer into five steps as shown in Figure 2. According to the derivative chain rules, we have

$$\begin{aligned}
\mathrm{Q_{E_1}}: \quad & \boldsymbol{e}_0^l = Q_{E_1}(\frac{\partial L}{\partial \boldsymbol{x}_4^l}) = Q_{E_1}(\boldsymbol{e}_4^{l+1}) \\
\text{Activation}: \quad & \boldsymbol{e}_1^l = \frac{\partial L}{\partial \boldsymbol{x}_3^l} = \frac{\partial L}{\partial \boldsymbol{x}_4^l} \odot \frac{\partial \boldsymbol{x}_4^l}{\partial \boldsymbol{x}_3^l} = \boldsymbol{e}_0^l \odot \frac{\partial \boldsymbol{x}_4^l}{\partial \boldsymbol{x}_3^l} \\
\text{Scale\&Offset}: \quad & \boldsymbol{e}_2^l = \frac{\partial L}{\partial \boldsymbol{x}_2^l} = \frac{\partial L}{\partial \boldsymbol{x}_3^l} \odot \frac{\partial \boldsymbol{x}_3^l}{\partial \boldsymbol{x}_2^l} = \boldsymbol{e}_1^l \odot \gamma_q^l \\
\mathrm{Norm\&Q_{E_2}}: \quad & \boldsymbol{e}_3^l = Q_{E_2}(\frac{\partial L}{\partial \boldsymbol{x}_1^l}) = Q_{E_2}(\frac{\partial L}{\partial \boldsymbol{x}_2^l} \odot \frac{\partial \boldsymbol{x}_2^l}{\partial \boldsymbol{x}_1^l}) \\
& \quad = Q_{E_2}(\boldsymbol{e}_2^l \odot \frac{\partial \boldsymbol{x}_2^l}{\partial \boldsymbol{x}_1^l}) \\
\text{Conv}: \quad & \boldsymbol{e}_4^l = \frac{\partial L}{\partial \boldsymbol{x}_0^l} = \frac{\partial \boldsymbol{x}_1^l}{\partial \boldsymbol{x}_0^l} \frac{\partial L}{\partial \boldsymbol{x}_1^l} = {W_q^l}^T \boldsymbol{e}_3^l
\end{aligned} \tag{2}$$

where L is the loss function, $\boldsymbol{e}_4^{l+1}$ represents the error from the $(l+1)$-th layer, and $\odot$ represents the Hadamard product. For vectors with the same dimension, such as $\boldsymbol{a} = (a_1, a_2, \cdots, a_n)$ and $\boldsymbol{b} = (b_1, b_2, \cdots, b_n)$, we have: $\boldsymbol{a} \odot \boldsymbol{b} = (a_1 b_1, a_2 b_2, \cdots, a_n b_n)$. Two quantization functions are used here: Q_{E_1} is the quantization function detailed as Equation (14) that converts high bit-width integers to low bit-width integers; Q_{E_2} detailed as Equation (16) is trying to convert FP values to low bit-width integers. ${W_q^l}^T$ is the transposed matrix of W_q^l, and $\partial \boldsymbol{x}_4^l / \partial \boldsymbol{x}_3^l$ represents the gradient of activation. When $relu$ is used as the activation function, $\partial \boldsymbol{x}_4^l / \partial \boldsymbol{x}_3^l$ is a tensor containing only 0 and 1 elements.

According to the definitions given above, the gradients of W, $\boldsymbol{\gamma}$, and $\boldsymbol{\beta}$ can be summarized as follows

$$\begin{aligned}
\boldsymbol{g}_W^l &= \frac{\partial L}{\partial W^l} = \frac{\partial L}{\partial \boldsymbol{x}_1^l} \frac{\partial \boldsymbol{x}_1^l}{\partial W^l} = \boldsymbol{e}_3^l {\boldsymbol{x}_0^l}^T \\
\boldsymbol{g}_{\boldsymbol{\gamma}}^l &= \frac{\partial L}{\partial \boldsymbol{\gamma}^l} = \frac{\partial L}{\partial \boldsymbol{x}_3^l} \odot \frac{\partial \boldsymbol{x}_3^l}{\partial \boldsymbol{\gamma}^l} = \boldsymbol{e}_1^l \odot \boldsymbol{x}_2^l \\
\boldsymbol{g}_{\boldsymbol{\beta}}^l &= \frac{\partial L}{\partial \boldsymbol{\beta}^l} = \frac{\partial L}{\partial \boldsymbol{x}_3^l} \odot \frac{\partial \boldsymbol{x}_3^l}{\partial \boldsymbol{\beta}^l} = \boldsymbol{e}_1^l.
\end{aligned} \tag{3}$$

To further reduce the bit width of G that will increase greatly after the multiplication, we have

$$\begin{aligned}
\boldsymbol{g}_{Wq}^l &= Q_{G_W}(g_W^l) \\
\boldsymbol{g}_{\boldsymbol{\gamma}q}^l &= Q_{G_{\boldsymbol{\gamma}}}(g_{\boldsymbol{\gamma}}^l) \\
\boldsymbol{g}_{\boldsymbol{\beta}q}^l &= Q_{G_{\boldsymbol{\beta}}}(g_{\boldsymbol{\beta}}^l)
\end{aligned} \tag{4}$$

where Q_{G_W}, $Q_{G_{\boldsymbol{\gamma}}}$, and $Q_{G_{\boldsymbol{\beta}}}$ are quantization functions for the gradient of W, $\boldsymbol{\gamma}$, and $\boldsymbol{\beta}$, respectively, which will be shown in Equation (17).

Some notations to be used below are also explained here. k_W, k_A, k_{G_W}, k_E (k_{E_1} and k_{E_2}), and k_{BN} are the bit width of W, A, G, E, and BN, respectively. k_{WU}, $k_{\boldsymbol{\gamma}U}$, and $k_{\boldsymbol{\beta}U}$ are the bit width of W, $\boldsymbol{\gamma}$, and $\boldsymbol{\beta}$ update, which are also the bit width of data stored in memory. $k_{\boldsymbol{\gamma}}$, $k_{\boldsymbol{\beta}}$, k_μ, and k_σ are the bit width of $\boldsymbol{\gamma}$, $\boldsymbol{\beta}$, μ, and σ, respectively, used in the BN layer. $k_{G\boldsymbol{\gamma}}$ and $k_{G\boldsymbol{\beta}}$ are the bit width of $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ gradient, respectively. k_{Mom} and k_{Acc} are the bit width of momentum coefficient (Mom) and accumulation (Acc), respectively, used in the Momentum optimizer. At last, k_{lr} is the bit width of the learning rate.

B. Quantization Functions

There are three quantization functions used in WAGEUBN. The direct-quantization function uses the nearest fixed-point values to represent the continuous values of W, A, and BN. The constant-quantization function for G is used to keep the bit width of U (update) fixed since G is directly related to U. Because U and the weights stored in memory have the same bit width, the bit width of weights stored in memory can be fixed, which is more hardware-friendly. The magnitude of E is very small, so the shift-quantization function reduces the bit width of E greatly compared with the direct-quantization function under the same precision.

(1) Direct-quantization function

The direct-quantization function simply approximates a continuous value to its nearest discrete state and is defined as

$$Q(\boldsymbol{x}, k) = \frac{round(\boldsymbol{x} \cdot 2^{k-1})}{2^{k-1}} \tag{5}$$

where k is the bit width, and $round(\cdot)$ rounds a number to its nearest INT value.

(2) Constant-quantization function

The intention of constant-quantization function is to normalize a tensor firstly, then limit it to INT, and finally maintain its magnitude. It is governed by

$$\begin{aligned}
R(\boldsymbol{x}) &= 2^{round(\log_2(max(|\boldsymbol{x}|)))} \\
Sr(x) &= \begin{cases} \lfloor x \rfloor, & P_x = x - \lfloor x \rfloor \\ \lceil x \rceil, & P_x = \lceil x \rceil - x \end{cases} \\
Norm(\boldsymbol{x}) &= \frac{\boldsymbol{x}}{R(\boldsymbol{x})} \\
Sd(\boldsymbol{x}) &= clip\left\{Sr\left[dr \cdot Norm(\boldsymbol{x})\right], -dr+1, dr-1\right\} \\
CQ(\boldsymbol{x}) &= \frac{Sd(\boldsymbol{x})}{2^{k_{GC}-1}}.
\end{aligned} \tag{6}$$

The illustration of constant-quantization function is described in Figure 3. Here, $R(\cdot)$ is used to project the maximum value of $\boldsymbol{x}$ to its nearest fixed-point value, which is prepared for normalization; $Sr(\cdot)$ is a stochastic rounding function used for converting a continuous float value to its nearby INT value in a probabilistic manner and P_x is the rounding

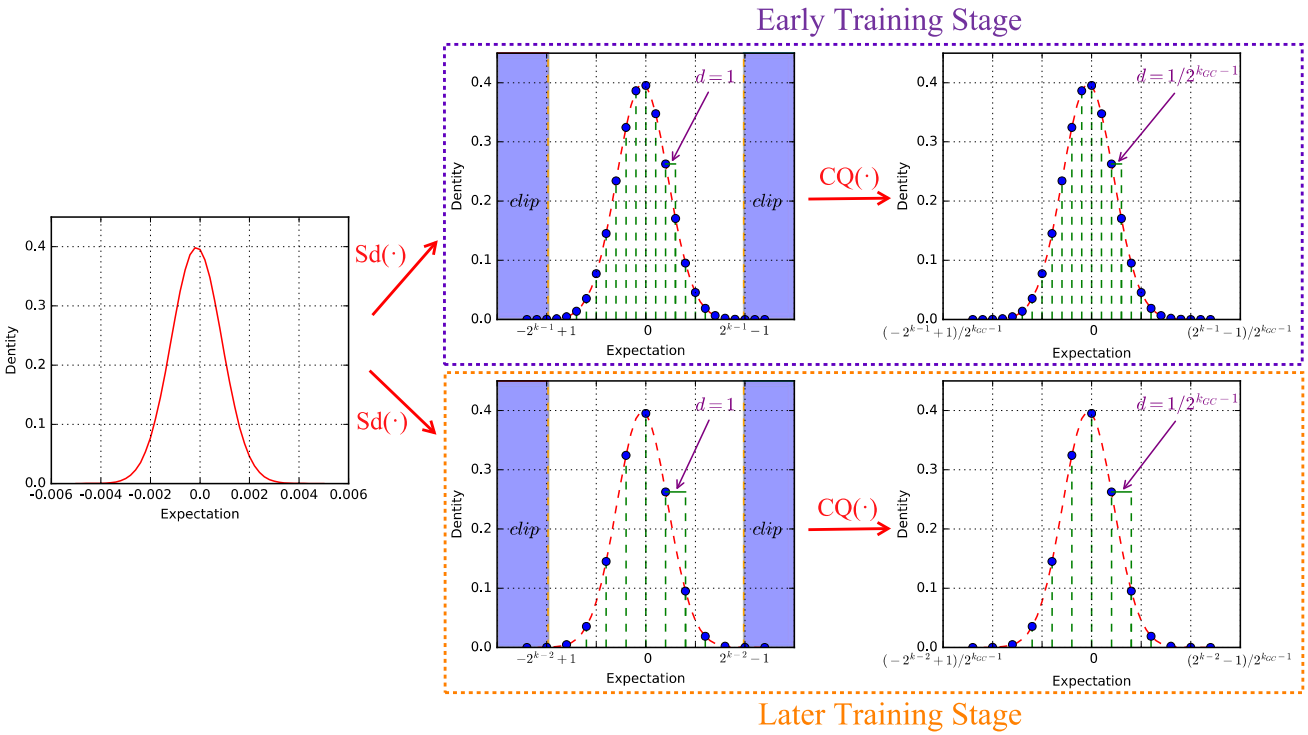


Fig. 3: Illustration of the constant-quantization function during training.

probability; $Norm(\cdot)$ denotes normalization and $clip(\cdot)$ is a saturation function limiting the data range;$Sd(\cdot)$ is to shift the distribution of $\boldsymbol{x}$ and limit $\boldsymbol{x}$ to INT values between $-dr+1$ and $dr-1$. Here $dr \in [2^{k-1}, 2^{k-2}, ..., 1]$ limits the data range after mapping and decreases as the training goes on, presenting the same effect as reducing the learning rate. For example, $Sd(\cdot)$ maps G to {-127, -126, $\cdots$, 126, 127} and {-63, -62, $\cdots$, 62, 63} in the early training stage ($k = 8$, $dr = 128$, epoch in $[0, 30]$) and later training stage ($k = 7$, $dr = 64$, epoch in $[30, 60]$), respectively. $CQ(\cdot)$ is utilized to maintain the magnitude order of data, where $2^{k_{GC}-1}$ is a constant scaling factor and k_{GC} is its bit width.

(3) Shift-quantization function

The shift-quantization function serves for the quantization of E and is defined as

$$d(k) = \frac{1}{2^{k-1}}$$

$$SQ(\boldsymbol{x}, k) = R(\boldsymbol{x}) \cdot clip\left\{Q\left[Norm(\boldsymbol{x}), k\right], -1 + d(k), 1 - d(k)\right\} \tag{7}$$

where $d(\cdot)$ is the minimum interval for a k-bit INT and $Q(\cdot)$ is the direct-quantization function defined in Equation (5).

The shift-quantization function normalizes E first, then converts E to fixed-point values, and finally uses a layer-wise scaling factor ($R(\cdot)$ defined in Equation (6)) to maintain the magnitude. The differences between the constant-quantization function and the shift-quantization function mainly exist in two points: First, the constant-quantization uses a constant to keep the magnitude for hardware friendliness while the shift-quantization uses a lay-wise scaling factor; Second, the constant-quantization contains a stochastic rounding process while the shift-quantization function does not.

C. Quantization Schemes in WAGEUBN

After introducing the quantization functions used in our WAGEUBN framework, we provide the detailed quantization schemes.

(1) Weight Quantization

Since weights are stored and used as fixed-point values, weights should be also initialized discretely. An initialization method proposed by MSRA [26] has been evidenced helpful for faster training. The initialization of weights can be formulated as follows

$$\begin{aligned} W^{'} &\sim N(0, \frac{1}{\sqrt{n_{in}}}) \\ W &= clip\left[Q(W^{'}, k_{WU}), -1 + d(k_{WU}), 1 - d(k_{WU})\right] \end{aligned} \tag{8}$$

where n_{in} is the layer's fan-in number, and k_{WU} is the bit width of weight update and the memory storage.

Because of the different bit width for weight storage and computation, it should be quantized from k_{WU} bits to k_W (the bit width of weights used for convolution) bits for convolution. In addition, we also limit the data range of W. Finally, the quantization function for W is

$$Q_W(\boldsymbol{x}) = clip\left[Q(\boldsymbol{x}, k_W), -1 + d(k_W), 1 - d(k_W)\right]. \tag{9}$$

(2) Batch Normalization Quantization

As aforementioned, BN plays an important role in training large-scale DNNs. WAGE [18] has proved that simple scaling layers are not enough to replace BN layers. Conventional BN layer can be divided into two steps as

$$\begin{aligned} \hat{\boldsymbol{x}} &= \frac{\boldsymbol{x} - \mu^l}{\sqrt{{\sigma^l}^2 + \epsilon}} \\ \boldsymbol{y} &= \boldsymbol{\gamma}^l \hat{\boldsymbol{x}} + \boldsymbol{\beta}^l \end{aligned} \tag{10}$$

where μ^l and σ^l are the mean and standard, respectively, deviation of $\boldsymbol{x}$ over one mini-batch in the l-th layer; ϵ is a small positive value added to σ to avoid the case of dividing by zero; $\boldsymbol{\gamma^l}$ and $\boldsymbol{\beta^l}$ are the scale and offset parameters, respectively.

Under the WAGEUBN framework, the BN layer is also quantized. Through the operations described in Equation (11), all operands are quantized and all operations are bit-wise. Specifically, the quantization follows

$$\begin{aligned} \mu_q^l = Q_\mu(\mu^l), \quad \sigma_q^l = Q_\sigma(\sigma^l) \\ \hat{\boldsymbol{x}} = Q_{BN}(\frac{\boldsymbol{x} - \mu_q^l}{\sigma_q^l + \epsilon_q}) \\ \boldsymbol{\gamma}_q^l = Q_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}^l), \quad \boldsymbol{\beta}_q^l = Q_{\boldsymbol{\beta}}(\boldsymbol{\beta}^l) \\ \boldsymbol{y} = \boldsymbol{\gamma}_q^l \hat{\boldsymbol{x}} + \boldsymbol{\beta}_q^l \end{aligned} \tag{11}$$

where $Q_\mu, Q_\sigma, Q_{\boldsymbol{\gamma}}, Q_{\boldsymbol{\beta}}, Q_{BN}$ are the quantization functions converting the operands to fixed-point values defined as

$$\begin{aligned} Q_\mu(\boldsymbol{x}) =& Q(\boldsymbol{x}, k_\mu), Q_\sigma(\boldsymbol{x}) = Q(\boldsymbol{x}, k_\sigma) \\ Q_{\boldsymbol{\gamma}}(\boldsymbol{x}) =& Q(\boldsymbol{x}, k_{\boldsymbol{\gamma}}), Q_{\boldsymbol{\beta}}(\boldsymbol{x}) = Q(\boldsymbol{x}, k_{\boldsymbol{\beta}}) \\ Q_{BN}(\boldsymbol{x}) =& Q(\boldsymbol{x}, k_{BN}). \end{aligned} \tag{12}$$

And ϵ_q is a small fixed-point value, playing the same role as ϵ in Equation (10); $k_\mu, k_\sigma, k_{\boldsymbol{\gamma}}, k_{\boldsymbol{\beta}}, k_{BN}$ are the bit width of $\mu, \sigma, \boldsymbol{\gamma}, \boldsymbol{\beta}$ and $\hat{\boldsymbol{x}}$, respectively.

(3) Activation Quantization

After the convolution and BN layers in the forward pass, the bit width of operands increases due to the multiplication operation. To reduce the bit width and keep the input bit width of each layer consistent, we need to quantize the activations. Here, the quantization function for activations can be described as

$$Q_A(\boldsymbol{x}) = Q(\boldsymbol{x}, k_A) \tag{13}$$

where k_A is the bit width of activations.

(4) Error Quantization

In Equation (2), we have given the definition of E and quantized E. Through investigating the importance of error propagation in DNN training, we find that the quantization of E is very essential for the model convergence. If E is naively quantized using the direct-quantization function, it will require a large bit width of operands to realize the convergence of DNNs. Instead, we use the following shift-quantization function

$$Q_{E_1}(\boldsymbol{x}) = SQ(\boldsymbol{x}, k_{E_1}) \tag{14}$$

where $SQ(\cdot)$ is the shift-quantization function defined in Equation (7), and k_{E_1} is the bit width of $\boldsymbol{e}_0^l$ defined in Equation (2).

As mentioned above, we use Q_{E_1} and Q_{E_2} for the error quantization. However, the precision requirements of Q_{E_1} and Q_{E_2} vary a lot. Experiments show that $k_{E_1} = 8$ affects little on accuracy while $k_{E_2} \leq 8$ will cause the non-convergence of large-scale DNNs when using $SQ(\cdot)$ as the quantization function. $k_{E_2} = 16$ is a proper value for the training of DNNs with minimum accuracy degradation. More analyses will be given in Subsection IV-E. Here we will provide two versions of Q_{E_2}, the 16-bit and 8-bit versions. The 16-bit Q_{E_2} is defined as

$$Q_{E_2}(\boldsymbol{x}) = SQ(\boldsymbol{x}, k_{E_2}) \tag{15}$$

where k_{E_2} is the bit width of $\boldsymbol{e}_3^l$ defined in Equation (2).

Experiments have proved the data range covered by 8-bit Q_{E_2} ($k_{E_2} = 8$) is not sufficient to train DNNs. In order to expand the coverage of quantization function while still maintaining a low bit width, we introduce a layer-wise scaling factor Sc and a flag bit. Then, to distinguish it from Q_{E_2} defined in Equation (15), we name the quantization function Flag Q_{E_2} and the quantization process is governed by

$$\begin{aligned} Sc =& \frac{R(\boldsymbol{x})}{2^{k_{E_2}-1}} \\ Q_{E_2}(\boldsymbol{x}) =& \begin{cases} Sc \cdot clip\left[round(\frac{\boldsymbol{x}}{Sc}), min, max\right], |\frac{\boldsymbol{x}}{Sc}| \geq 1 \\ Sc \cdot Q(\frac{\boldsymbol{x}}{Sc}, k_{E_2}), |\frac{\boldsymbol{x}}{Sc}| < 1 \end{cases} \end{aligned} \tag{16}$$

where $k_{E_2} = 8$, $min = -2^{k_{E_2}} + 1$, and $max = 2^{k_{E_2}} - 1$.

(a)

(b)

Fig. 4: Data format of 9-bit integers.

By introducing a layer-wise scaling factor and a flag bit, we can expand the data coverage greatly. Details can be found in Figure 4. The flag bit is used to indicate whether the absolute value of x stored in memory is less than the layer-wise scaling factor (e.g., 0 represents $|x|<Sc$ and 1 represents $|x| \geq Sc$). The sign bit is used to denote the positive or negative direction of the value. The data bit follows the conventional binary format. According to the definition, the values stored in Figure 4(a) and 4(b) are $+Sc/128$ and $-127 \times Sc$ when $k_{E_2} = 8$, respectively. Therefore, the 9-bit data format can cover almost the same data range as the direct 15-bit quantization described in Equation (15). Since the flag bit is just used for judgment, the effective value for computation is still INT8.

(5) Gradient Quantization

The gradient is another important part in DNN training because it is directly related to the weight update. The rules for calculating and quantizing the gradients of W, $\boldsymbol{\gamma}$, and $\boldsymbol{\beta}$ are

described as Equation (3) and (4). Since $\boldsymbol{e_1^l}$, $\boldsymbol{e_3^l}$, $\boldsymbol{x_0^l}$, and $\boldsymbol{x_2^l}$ are all fixed-point values, the conventional FP MACs operations can be replaced with bit-wise operations during the process of calculating $\boldsymbol{g}_W^l$, $\boldsymbol{g}_{\boldsymbol{\gamma}}^l$, and $\boldsymbol{g}_{\boldsymbol{\beta}}^l$. The quantization functions are defined to further reduce the bit width of gradients and prepare for the next step of the optimizer. Specifically, we have

$$\begin{aligned} Q_{G_W}(\boldsymbol{x}) &= CQ(\boldsymbol{x}, k_{G_W}) \\ Q_{G_{\boldsymbol{\gamma}}}(\boldsymbol{x}) &= Q(\boldsymbol{x}, k_{G_{\boldsymbol{\gamma}}}) \\ Q_{G_{\boldsymbol{\beta}}}(\boldsymbol{x}) &= Q(\boldsymbol{x}, k_{G_{\boldsymbol{\beta}}}) \end{aligned} \tag{17}$$

where $CQ(\cdot)$ is the constant-quantization function defined in Equation (6); k_{G_W}, $k_{G_{\boldsymbol{\gamma}}}$, and $k_{G_{\boldsymbol{\beta}}}$ are the bit width of the gradient of W, $\boldsymbol{\gamma}$, and $\boldsymbol{\beta}$, respectively.

(6) Momentum Optimizer Quantization

Momentum optimizer is one of the most common optimizers used in DNN training, especially for classification tasks. For the i-th training step of the l-th layer, the conventional Momentum optimizer works as follows

$$Acc_i^l = Mom \cdot Acc_{i-1}^l + \boldsymbol{g}_i^l \tag{18}$$

where Acc_i^l and Acc_{i-1}^l are the accumulation in the i-th and $(i-1)$-th training step, respectively; Mom is a constant value used as a coefficient; $\boldsymbol{g}_i^l$ is the gradient of W, $\boldsymbol{\gamma}$, or $\boldsymbol{\beta}$.

Momentum optimizer under the WAGEUBN framework is trying to constrain all operands to fixed-point values. The process can be formulated as

$$\begin{aligned} Acc_i^l &= Mom \cdot Acc_{(i-1)q}^l + g_{iq}^l \\ Acc_{iq}^l &= Q_{Acc}(Acc_i^l) \end{aligned} \tag{19}$$

where $Acc_{(i-1)q}^l$ is the quantized accumulation in the $(i-1)$-th training step; g_{iq}^l is the quantized gradient of W, $\boldsymbol{\gamma}$, or $\boldsymbol{\beta}$; $Q_{Acc}(\cdot)$ is the quantization function defined as

$$Q_{Acc}(\boldsymbol{x}) = Q(\boldsymbol{x}, k_{Acc}). \tag{20}$$

To guarantee the consistency of bit width, we further set

$$k_{G_{\boldsymbol{\gamma}}} = k_{G_{\boldsymbol{\beta}}} = k_{GC} = k_{Mom} + k_{Acc} - 1. \tag{21}$$

(7) Update Quantization

The parameter update is the last step in the training of each mini-batch. Different from conventional DNNs where the learning rate can take any FP value, the learning rate under WAGEUBN must also be a fixed-point value and the bit width of update is directly related to the bit width of learning rate. The update under quantized Momentum optimizer can be described as

$$\begin{aligned} \Delta W &= lr \cdot Acc_i^l \\ W^l &= W^l - \Delta W \end{aligned} \tag{22}$$

where ΔW is the update of W with k_{WU} bits, and lr is the fixed-point learning rate with k_{lr} bits. The updates of $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are the same as in Equation (22). According to Equation (19), (21), and (22), we have

$$\begin{aligned} k_{WU} = k_{\boldsymbol{\gamma}U} = k_{\boldsymbol{\beta}U} &= k_{Mom} + k_{Acc} + k_{lr} - 2 \\ &= k_{GC} + k_{lr} - 1 \\ &= k_{G_{\boldsymbol{\gamma}}} + k_{lr} - 1 \\ &= k_{G_{\boldsymbol{\beta}}} + k_{lr} - 1. \end{aligned} \tag{23}$$

Through our evaluations, the precision of the update has the greatest impact on the accuracy of DNNs because it is the last step to constrain the parameters. Thus, we need to set a reasonable bit width for update to balance the model accuracy and memory cost.

D. Quantization Framework

Given the quantization details of W, A, G, E, U, BN, and the Momentum optimizer, the overall quantization framework is depicted in Figure 5. Under this framework, conventional FP MACs can be replaced with bit-wise operations. Here, the forward pass of the l-th layer in DNNs is divided into three parts: Conv (convolution), BN, and activation. $\boldsymbol{x}_0^l$, $\boldsymbol{x}_1^l$, $\boldsymbol{x}_2^l$, $\boldsymbol{x}_3^l$, and $\boldsymbol{x}_4^l$ are defined in Equation (1). The weights (W^l) are stored as k_{WU}-bit integers and then Q_W maps W^l to k_W-bit INT values (W_q^l) before convolution. After convolution, $MEAN\&Q_\mu$ and $STD\&Q_\sigma$ operations are used to calculate the mean and standard deviation of $\boldsymbol{x}_1^l$ in one mini-batch and then quantize them to k_μ and k_σ bits, respectively. The $BW\&Q_{BN}$ operation constrains $\boldsymbol{x}_2^l$ to k_{BN} bits in BN. Similar to W, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are stored as $k_{\boldsymbol{\gamma}U}$ and $k_{\boldsymbol{\beta}U}$-bit integers and used in $k_{\boldsymbol{\gamma}}$ and $k_{\boldsymbol{\beta}}$ bits ($\boldsymbol{\gamma}_q^l$ and $\boldsymbol{\beta}_q^l$) after the $Q_{\boldsymbol{\gamma}}$ and $Q_{\boldsymbol{\beta}}$ quantization, respectively. After the second step of BN, activation and quantization are implemented with the $ACT\&Q_A$ operation, reducing the increased bit width to k_A bits again and preparing inputs for the next layer.

The backward pass of the l-th layer is much more complicated than the forward pass, including error propagation, gradient of weight, gradient of BN, Momentum optimizer, and weight update. In the process of error propagation, $\boldsymbol{e}_0^l$, $\boldsymbol{e}_1^l$, $\boldsymbol{e}_2^l$, $\boldsymbol{e}_3^l$, and $\boldsymbol{e}_4^l$ are defined in Equation (2) and there are two locations needing quantization using Q_{E_1} and Q_{E_2}. Q_{E_1} reduces the bit width of $\boldsymbol{e}_4^{l+1}$ from $k_{E_2} + k_W - 1$ to k_{E_1}. ACT' is the derivative of activation function ($relu$) and Q_{E_2} is used to constrain $\boldsymbol{e}_3^l$ to k_{E_2} bits. In the phase of calculating the gradients of weights and BN, Q_{G_W}, $Q_{G_{\boldsymbol{\gamma}}}$, and $Q_{G_{\boldsymbol{\beta}}}$ are leveraged to reduce the increased bit width caused by the multiplication operations.

All parameters of the Momentum optimizer in the i-th training step are quantized. Different from the conventional learning rate with FP value, WAGEUBN requires a discrete learning rate so that the bit width of weight updates can be controlled. The updates of $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ in BN layers are also similar to the weight update, which are omitted in Figure 5 for simplicity.

E. Overall Algorithm

Given the framework of WAGEUBN, we summarize the entire quantization process and present the pseudo codes for both the forward and backward passes as shown in Algorithm 1 and 2, respectively.

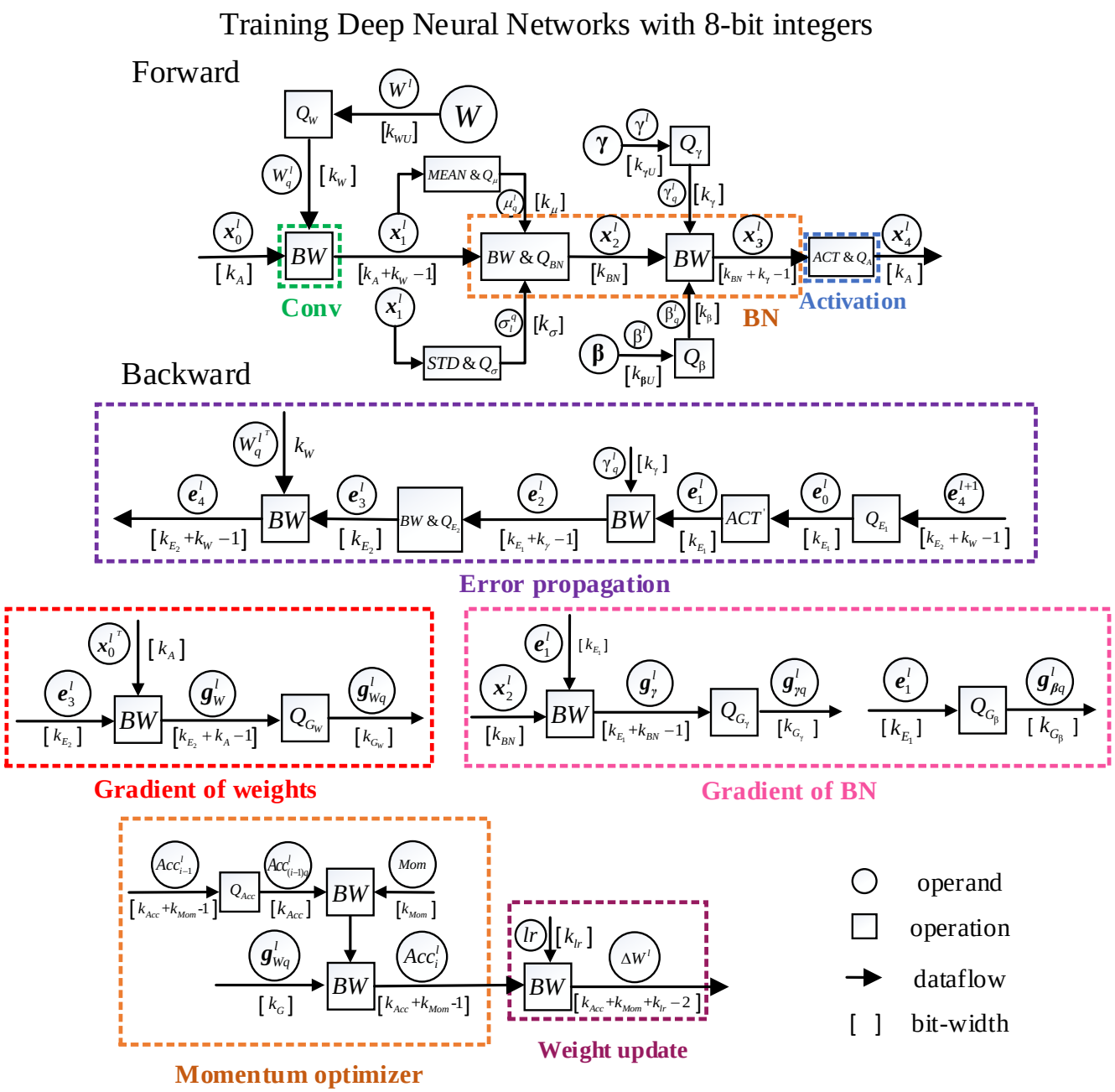


Fig. 5: Overview of the WAGEUBN quantization framework. "BW" denotes bit-wise operations.

IV. Results

A. Experimental Setup

To verify the effectiveness of the proposed quantization framework, we apply WAGEUBN on ResNet18/34/50 on ImageNet dataset. We provide two versions of WAGEUBN, one with full 8-bit INT where k_W, k_A, k_{G_W}, k_{E_1}, k_{E_2}, $k_{\boldsymbol{\gamma}}$, and $k_{\boldsymbol{\beta}}$ are equal to 8. The other version has 16-bit k_{E_2}. The only difference between the 16-bit E_2 version and the full 8-bit version exists in the quantization function Q_{E_2} (see Equation (15) and (16), respectively). $k_{G_{\boldsymbol{\gamma}}}$, $k_{G_{\boldsymbol{\beta}}}$, and k_{GC} are 15 and we set k_{mom}, k_{Acc}, k_{lr}, and k_{WU} to 3, 13, 10, and 24 respectively to satisfy Equation (21) and (23). In addition, we set k_{BN}, k_{μ}, and k_{σ} to 16. Since W, A, G, and E occupy the majority of memory and compute costs, their bit-width values are reduced as much as possible. Other parameters occupying much less resources can increase the bit width to maintain the accuracy, e.g., μ_q^l, σ_q^l, $g_{\boldsymbol{\gamma q}}^l$, and $g_{\boldsymbol{\beta q}}^l$ in BN layers.

The first and last layers are believed to differ from the rest because of their interface with network inputs and outputs. The quantization of these two layers will cause significant accuracy degradation compared to hidden layers and they just consume few overheads due to the small number of neurons. Therefore, we do not quantize the first and last layers, as previous work did [27], [28].

Algorithm 1 Forward pass of l-th layer

Convolution:

$$\boldsymbol{x}_0^l \Leftarrow \boldsymbol{x}_4^{l-1},\ \boldsymbol{W}_q^l \Leftarrow Q_W(\boldsymbol{W}^l, k_W)$$
$$\boldsymbol{x}_1^l \Leftarrow \boldsymbol{W}_q^l \boldsymbol{x}_0^l$$

BN:

$$\mu_q^l \Leftarrow Q_\mu(\mu^l),\ \sigma_q^l \Leftarrow Q_\sigma(\sigma^l)$$
$$\boldsymbol{x}_2^l \Leftarrow Q_{BN}(\frac{\boldsymbol{x}_1^l - \mu_q^l}{\sigma_q^l})$$
$$\boldsymbol{\gamma}_q^l \Leftarrow Q_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}^l),\ \boldsymbol{\beta}_q^l \Leftarrow Q_{\boldsymbol{\beta}}(\boldsymbol{\beta}^l)$$
$$\boldsymbol{x}_3^l \Leftarrow \boldsymbol{\gamma}_q^l \boldsymbol{x}_2^l + \boldsymbol{\beta}_q^l$$

Activation:

$$\boldsymbol{x}_4^l \Leftarrow Q_A(relu(\boldsymbol{x}_3^l))$$

Algorithm 2 Backward pass of l-th layer

Error propagation:

$$\boldsymbol{e}_0^l \Leftarrow Q_{E_1}(\boldsymbol{e}_4^{l+1})$$
$$\boldsymbol{e}_1^l \Leftarrow \boldsymbol{e}_0^l \odot \frac{\partial \boldsymbol{x}_4^l}{\partial \boldsymbol{x}_3^l}$$
$$\boldsymbol{e}_2^l \Leftarrow \boldsymbol{e}_1^l \odot \boldsymbol{\gamma}_q^l$$
$$\boldsymbol{e}_3^l \Leftarrow Q_{E_2}(\boldsymbol{e}_2^l \odot \frac{\partial \boldsymbol{x}_2^l}{\partial \boldsymbol{x}_1^l})$$
$$\boldsymbol{e}_4^l \Leftarrow {W_q^l}^T \boldsymbol{e}_3^l$$

Gradient computation:

$$\boldsymbol{g}_W^l \Leftarrow \boldsymbol{e}_3^l {\boldsymbol{x}_0^l}^T$$
$$\boldsymbol{g}_{Wq}^l \Leftarrow Q_{G_W}(\boldsymbol{g}_W^l)$$
$$\boldsymbol{g}_{\boldsymbol{\gamma}}^l \Leftarrow \boldsymbol{e}_1^l \odot \boldsymbol{x}_2^l$$
$$\boldsymbol{g}_{\boldsymbol{\gamma}q}^l \Leftarrow Q_{G_{\boldsymbol{\gamma}}}(\boldsymbol{g}_{\boldsymbol{\gamma}}^l)$$
$$\boldsymbol{g}_{\boldsymbol{\beta}}^l \Leftarrow \boldsymbol{e}_1^l$$
$$\boldsymbol{g}_{\boldsymbol{\beta}q}^l \Leftarrow Q_{G_{\boldsymbol{\beta}}}(\boldsymbol{g}_{\beta}^l)$$

Momentum optimizer(i-th step):

$$Acc_i^l = Mom \cdot Acc_{(i-1)q}^l + \boldsymbol{g}_{iq}^l(\boldsymbol{g}_{Wq}^l, \boldsymbol{g}_{\boldsymbol{\gamma}q}^l, or\ \boldsymbol{g}_{\boldsymbol{\beta}q}^l)$$
$$Acc_{iq}^l = Q_{Acc}(Acc_i^l)$$

Weight updates(i-th step):

$$\Delta W = lr \cdot Acc_i^l$$
$$W^l = W^l - \Delta W$$

B. Training Curve

Figure 6 illustrates the accuracy comparison between vanilla DNNs (FP32), DNNs with 8-bit version of WAGEUBN, and DNNs with 16-bit version of WAGEUBN. The training curves show that there is little difference between vanilla DNNs and the ones under the WAGEUBN framework when the training epoch is less than 60, which reflects the effectiveness of our approach. As the epoch evolves, the accuracy gap begins to grow because the learning rate in vanilla DNNs is much lower than that in WAGEUBN, such as 5×10^{-6} v.s. 1.95×10^{-3} ($k_{lr} = 10$), thus the update of vanilla DNNs is more precise than that under the WAGEUBN framework. We can further improve the accuracy by reducing the learning rate, while the bit width values of learning rate and update need to increase accordingly at the expense of more overheads.

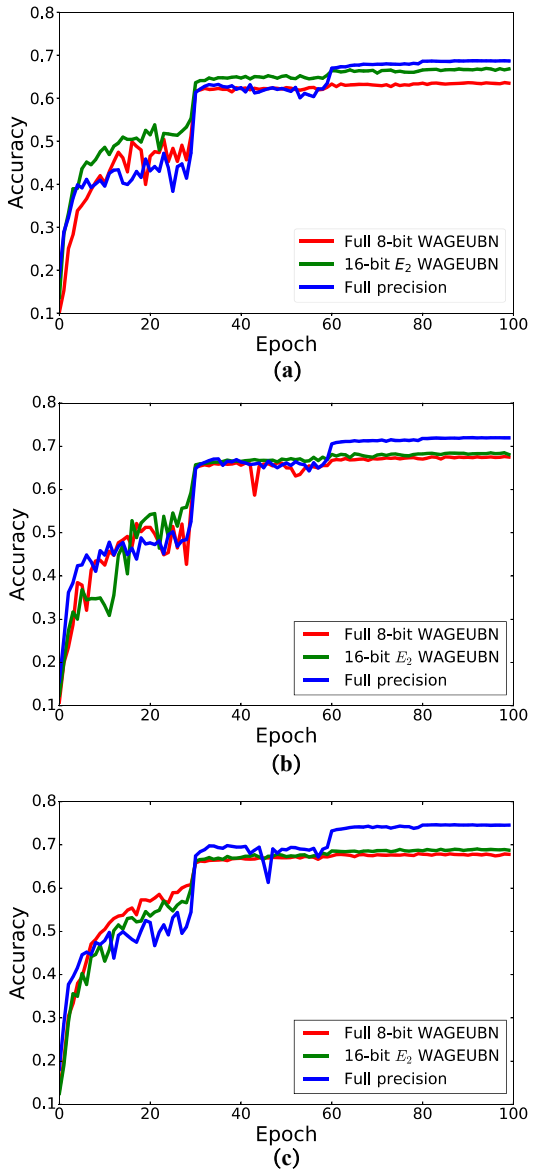


Fig. 6: Training curves under the WAGEUBN framework: (a) ResNet18; (b) ResNet34; (c) ResNet50.

Table I quantitatively presents the accuracy comparison between vanilla DNNs and WAGEUBN DNNs on ImageNet dataset. We have achieved the state-of-the-art accuracy on large-scale DNNs with full 8-bit INT quantization. The 16-bit E_2 WAGEUBN only loses 3.62% mean accuracy compared with the vanilla DNNs. Because the bit width of most data keeps the same between the full 8-bit WAGEUBN and the 16-bit E_2 WGAEUBN, the overhead difference between them is negligible. Compared with the vanilla DNNs, about $4\times$ memory size shrink, much faster processing speed, and much less energy and circuit area can be achieved under the proposed

TABLE I: Accuracy of vanilla DNNs and WAGEUBN DNNs on ImageNet dataset.

Network	k_W	k_A	k_{G_W}	k_{E_1}	k_{E_2}	k_{WU}	Accuracy Top-1/Top-5(%)
ResNet18	32	32	32	32	32	32	68.70/88.37
	8	8	8	8	16	24	66.92/87.42
	8	8	8	8	8	24	63.62/84.80
ResNet34	32	32	32	32	32	32	71.99/90.56
	8	8	8	8	16	24	68.50/87.96
	8	8	8	8	8	24	67.63/87.70
ResNet50	32	32	32	32	32	32	74.66/92.13
	8	8	8	8	16	24	69.07/88.45
	8	8	8	8	8	24	67.95/88.01

WAGEUBN framework.

C. Quantization Strategies for W, A, G, E, and BN

In our WAGEUBN framework, we use different quantization strategies for W, A, G, E, and BN, i.e. $Q(\cdot)$ for W, A, and BN; $CQ(\cdot)$ for G, and $SQ(\cdot)$ for E. Different quantization strategies are based on the data distribution, data sensitivity, and hardware friendliness. Figure 7 shows the distribution comparison between W, BN ($\boldsymbol{x}_2^l$ defined in Equation (1)), A, G (weight gradient), and E ($\boldsymbol{e}_0^l$, $\boldsymbol{e}_3^l$ defined in Equation (2)) before and after quantization.

According to the definition, the resolution of direct-quantization function is 2^{-7} when the bit width equals to 8 and there is no limitation on the data range. Because W, BN, and A in the inference stage directly affect the loss function and further influence the backpropagation, the quantization of W, BN, and A should be as precise as possible to avoid the loss fluctuation. This is guaranteed for the reason that the resolution of direct-quantization function is enough for W, BN, and A, which indicates that the direct-quantization function barely changes their data distributions.

The constant-quantization function has a resolution of 2^{-14} and the data range after quantization is about $[-2^{-7}, 2^{-7}]$ in the case of $k_{GC} = 15$ and $k = 8$. k will decrease as the training epoch goes on, causing the data range reduction. Figure 7 reveals that the constant-quantization function changes the data distribution of G greatly while the network accuracy has not declined much as a result. The reason behind this phenomenon is that it is the orientation rather than the magnitude of gradients that guides DNNs to converge. In the meantime, it is easy to ensure that the bit width of update can be fixed when k_{GC} is fixed, which is more hardware-friendly since the bit width of weights stored in memory can also be fixed during training.

The shift-quantization retains the magnitude order and omits the general values whose absolute value is less than 2^{-7} when $k = 8$. The 8-bit shift-quantization works well for the quantization of error after activation ($\boldsymbol{e}_1^l$ defined in Equation (2)). However, we find the shift-quantization is not enough for the quantization of errors between Conv and BN ($\boldsymbol{e}_3^l$ defined in Equation (2)). Therefore, the newly designed quantization function in Equation (16) named 8-bit Flag Q_{E_2} is utilized. Figure 7 shows that the distribution of E ($\boldsymbol{e}_3^l$) is almost the same before and after quantization, revealing the validity of the 8-bit Flag Q_{E_2} quantization function.

D. Accuracy Sensitivity Analysis

To compare the influences of W, A, G, E, and BN quantization individually, we quantize them to 8-bit INT separately with FP32 update. Taking $k_W = 8$ as an example, we quantize only W to 8-bit INT and leaving others (A, BN, G, E, and U) still kept in FP32. The quantization function for single data used here is the same as what Section III-C describes (Equation (16) is used for the the error quantization when $k_{E_2} = 8$).

The results of ResNet18 under the WAGEUBN framework with single data quantization is shown in Table II. The accuracy of single data quantization reflects the difficulty degree when quantizing W, A, G, E, and BN, separately. From the table, we can see that the quantization of E, especially $\boldsymbol{e}_3^l$ defined in Equation (2), makes the most impacts on accuracy. In addition, we find that the accuracy heavily fluctuates during training when $\boldsymbol{e}_3^l$ is constrained to 8-bit INT, which does not appear in the quantization of other data. To sum up, the E data, especially $\boldsymbol{e}_3^l$, demands the highest precision and is the most sensitive component under our WAGEUBN framework.

E. Analysis of the Error Quantization between Conv and BN

Error backpropagation is the foundation of DNN training. If the error of E caused by quantization is too large, the convergence of DNNs will be degraded. Especially, because the error quantization between convolution and BN ($\boldsymbol{e}_3^l$) is directly related to the weight update of the l-th layer, the impact of $\boldsymbol{e}_3^l$ quantization on the model accuracy is critical.

To further analyze the reason why 8-bit Q_{E_2} (defined in Equation (15) where $k_{E_2} = 8$) causes the non-convergence of DNNs and compare the distributions of $\boldsymbol{e}_3^l$ under 8-bit Q_{E_2}, 8-bit Flag Q_{E_2} (defined in Equation (16) where $k_{E_2} = 8$), and full precision, the data distribution of $\boldsymbol{e}_3^l$ of the first quantized layer on ResNet18 is shown in Figure 8. From the figure, we can see that the $\boldsymbol{e}_3^1$ distributions under 8-bit Q_{E_2} quantization

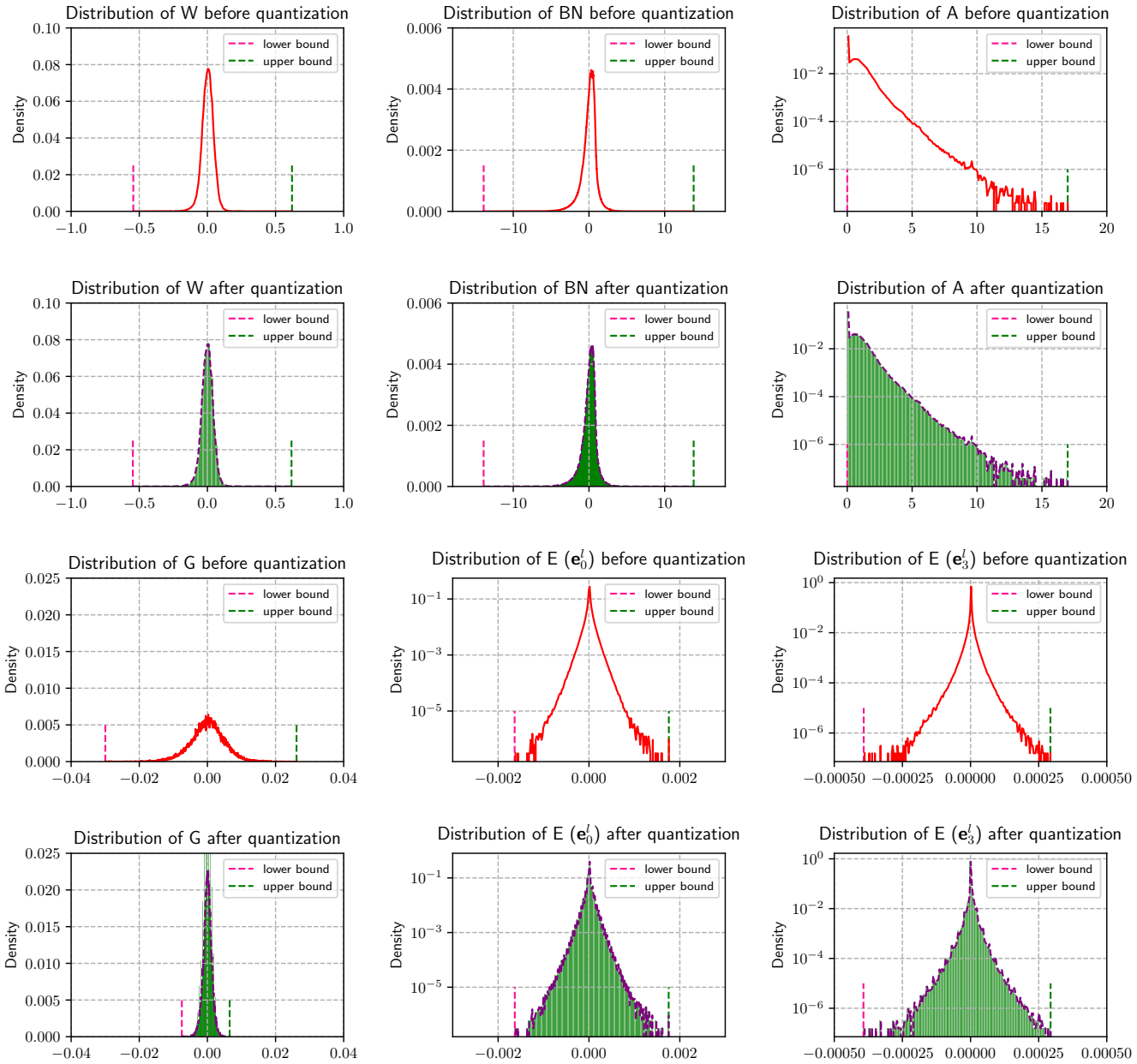


Fig. 7: Data distribution comparison between W, BN, A, G, and E before and after quantization.

TABLE II: Accuracy sensitivity under WAGEUBN with single data quantization on ResNet18.

Bit-width	$k_W = 8$	$k_{BN} = 8$	$k_A = 8$	$k_{G_W} = 8$	$k_{E_1} = 8$	$k_{E_2} = 8$
Accuracy Top-1/Top-5 (%)	67.98/88.02	68.01/87.96	67.74/87.89	67.88/87.89	67.88/87.92	67.08/87.44

and full precision differs a lot and those of 8-bit Flag Q_{E_2} quantization and full precision are almost the same. The major difference between the 8-bit Q_{E_2} quantization and full precision lies in the interval of $[-2^{-8}R(\boldsymbol{e}_3^1), 2^{-8}R(\boldsymbol{e}_3^1)]$ ($R(\cdot)$ is defined in Equation (6)), where the 8-bit Q_{E_2} quantization forces the data in this range to zero.

The only difference between 8-bit Q_{E_2} and 8-bit Flag Q_{E_2} quantization functions lies in the data range. Theoretically, the covered data range of 8-bit Q_{E_2} and 8-bit Flag Q_{E_2} are about $[-R(\boldsymbol{e}_3^l), -2^{-8}R(\boldsymbol{e}_3^l)] \cup \{0\} \cup [2^{-8}R(\boldsymbol{e}_3^l), R(\boldsymbol{e}_3^l)]$ and $[-R(\boldsymbol{e}_3^l), -2^{-15}R(\boldsymbol{e}_3^l)] \cup \{0\} \cup [2^{-15}R(\boldsymbol{e}_3^l), R(\boldsymbol{e}_3^l)]$, respectively. Because the distribution of $\boldsymbol{e}_3^l$ is not uniform, the range covered by different quantization methods varies a lot. The data ratios of 8-bit Q_{E_2} and 8-bit Flag Q_{E_2} quantization functions covered by each layer of ResNet18 are illustrated as Figure 9. Although the larger values take greater impacts on the model accuracy in the process of error propagation, the smaller values also contain useful information and occupy the majority. Compared with 8-bit Flag Q_{E_2}, the data ratio 8-bit Q_{E_2} covers is too little because of the smaller data range. That

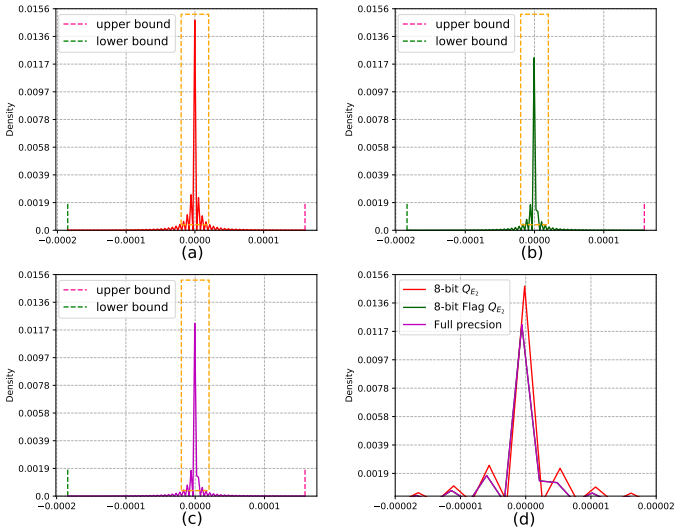


Fig. 8: Data distributions of e_3^1: (a) 8-bit Q_{E_2}, (b) 8-bit Flag Q_{E_2}, (c) full precision; (d) Data distribution comparison.

is to say, although the most important information contained by the larger values is retained, the information contained by the smaller values is ignored, resulting in the non-convergence of DNNs. In addition, there is also a rough trend that the data ratio decreases as the network becomes shallower, either in the 8-bit Q_{E_2} or 8-bit Flag Q_{E_2} quantization.

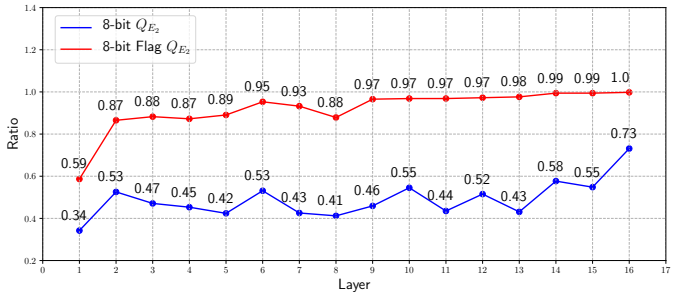
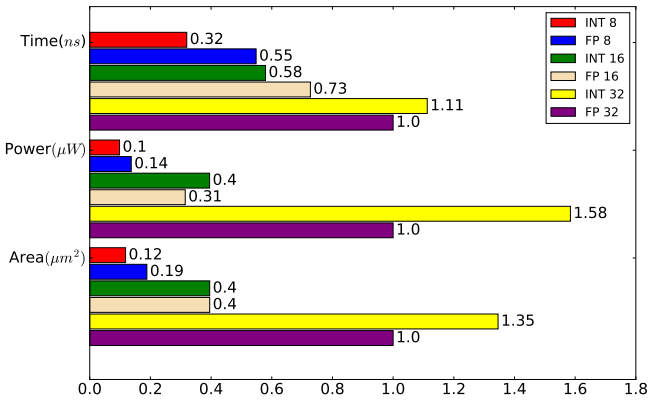


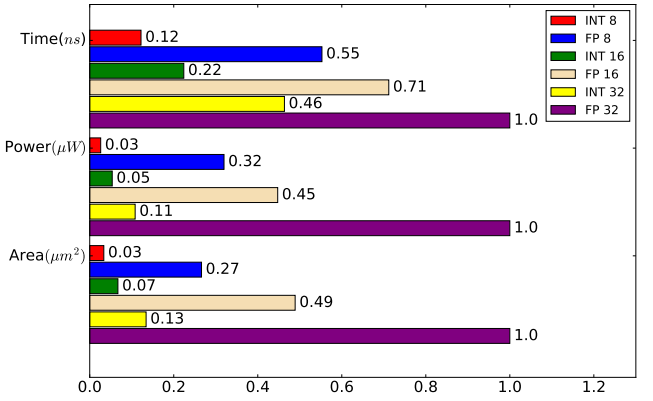
Fig. 9: Data ratios of 8-bit Q_{E_2} and 8-bit Flag Q_{E_2} quantization methods covered by each layer of ResNet18.

F. Cost Discussion

Although it is recognized that DNN quantization can greatly reduce memory and compute costs, resulting in lower energy consumption, quantitative analysis is rarely seen in recent research. In order to compare the full INT8 quantization with other precision solutions (FP32, INT32, FP16, INT16, and FP8) more clearly, we have simulated the processing speed, power consumption, and circuit area for single multiplication and accumulation operation on FPGA platform. Figure 10 shows the results. With FP32 as the baseline, taking the multiplication operation as an example, INT8 can perform $>3\times$ faster in speed, $10\times$ lower in power, and $9\times$ smaller in circuit area. Similarly, compared with FP32, the speed of INT8 accumulation is about $9\times$ faster, and the energy consumption and circuit area are reduced by $>30\times$. In addition, the INT8 multiplication and accumulation operations are more advantageous than other data type operations, whether it is FP8, INT16, FP16 or INT32. In conclusion, the proposed full INT8 quantization has great advantages in hardware overheads, whether in terms of processing speed, power consumption, and circuit area.



(a)



(b)

Fig. 10: Comparison of time, power, and area of single multiplication and accumulation operation under different quantization precision: (a) multiplication, (b) accumulation.

V. CONCLUSIONS

We propose a unified framework termed as "WAGEUBN" to achieve a complete quantization of large-scale DNNs in both training and inference with competitive accuracy. We are the first to quantize DNNs over all data paths and promote DNN quantization to the full INT8 level. In this way, all the operations can be replaced with bit-wise operations, causing significant improvements of memory overhead, processing speed, circuit area, and energy consumption. Extensive experiments evidence the effectiveness and efficiency of WAGEUBN. This work provides a feasible solution for the online training acceleration of large-scale and high-performance DNNs and further shows the great potential for the applications in future efficient portable devices with online learning ability. Future works

could transfer to the design of computing architecture, memory hierarchy, interconnection infrastructure, and mapping tool to enable the specialized machine learning chip.

ACKNOWLEDGMENT

The work was supported partially by the National Natural Science Foundation of China (No. 61603209, 61876215).

REFERENCES

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[2] Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In *International Conference on Neural Information Processing Systems*, 2012.

[3] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[4] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[5] Bo Zhang, Luping Shi, and Sen Song. Creating more intelligent robots through brain-inspired computing. *special supplement: Brain-inspired intelligent robotics: The intersection of robotics and neuroscience sciences*, pages 4–9, Science Vol. 354, Issue 6318, pp.1445 (2016).

[6] Wonyong Sung, Sungho Shin, and Kyuyeon Hwang. Resiliency of deep neural networks under quantization. *arXiv preprint arXiv:1511.06488*, 2015.

[7] 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.

[8] 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.

[9] Cong Leng, Zesheng Dou, Hao Li, Shenghuo Zhu, and Rong Jin. Extremely low bit neural network: Squeeze the last bit out with admm. 2018.

[10] 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.

[11] Lei Deng, Peng Jiao, Jing Pei, Zhenzhi Wu, and Guoqi Li. Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. *Neural Networks*, 100:49–58, 2018.

[12] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *Advances in neural information processing systems*, pages 7675–7684, 2018.

[13] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. In *Advances in Neural Information Processing Systems*, pages 5145–5153, 2018.

[14] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[16] Dipankar Das, Naveen Mellempudi, Dheevatsa Mudigere, Dhiraj Kalamkar, Sasikanth Avancha, Kunal Banerjee, Srinivas Sridharan, Karthik Vaidyanathan, Bharat Kaul, Evangelos Georganas, et al. Mixed precision training of convolutional neural networks using integer operations. *arXiv preprint arXiv:1802.00930*, 2018.

[17] Charbel Sakr and Naresh Shanbhag. Per-tensor fixed-point quantization of the back-propagation algorithm. *arXiv preprint arXiv:1812.11732*, 2018.

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

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[21] 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.

[22] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[24] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[25] 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.

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

[27] Diwen Wan, Fumin Shen, Li Liu, Fan Zhu, Jie Qin, Ling Shao, and Heng Tao Shen. Tbn: Convolutional neural network with ternary inputs and binary weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 315–332, 2018.

[28] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. Learning low precision deep neural networks through regularization. *arXiv preprint arXiv:1809.00095*, 2018.