Exploring the Potential of Low-bit Training of Convolutional Neural Networks

Kai Zhong, Xuefei Ning, Guohao Dai, Zhenhua Zhu, Tianchen Zhao, Shulin Zeng, Yu Wang, *Fellow, IEEE,* and Huazhong Yang, *Fellow, IEEE*

Abstract*—Convolutional neural networks (CNNs) have been widely used in many tasks, but training CNNs is time-consuming and energy-hungry. Using the low-bit integer format has been proved promising for speeding up and improving the energy efficiency of CNN inference, while CNN training can hardly benefit from such a technique because of the following challenges: *(1) The integer data format cannot meet the requirements of the data dynamic range in training, resulting in the accuracy drop; (2) The floating-point data format keeps sizeable dynamic range with much more exponent bits, thus using it results in higher accumulation power than using the integer data format; (3) There are some specially designed data formats (e.g., with group-wise scaling) that have the potential to deal with the former two problems but common hardware platforms can not support them efficiently.

To tackle all these challenges and make the training phase of CNNs benefit from the low-bit format, we propose a low-bit training framework for convolutional neural networks to pursue a better trade-off between accuracy and energy efficiency. *(1) We adopt element-wise scaling to increase the dynamic range of data representation, which significantly reduces the quantization error; (2) Group-wise scaling with hardware friendly factor format is designed to reduce the element-wise exponent bits without degrading the accuracy; (3) We design the customized hardware unit that implements the low-bit tensor convolution arithmetic with our multi-level scaling data format.* Experiments show that our framework achieves a superior trade-off between the accuracy and the bit-width than previous low-bit training studies. For training various models on CIFAR-10, using 1-bit mantissa and 2-bit exponent is adequate to keep the accuracy loss within 1%. On larger datasets like ImageNet, using 4-bit mantissa and 2-bit exponent is adequate. Through the energy consumption simulation of the whole network, we can see that training a variety of models with our framework could achieve $4.9 \sim 10.2\times$ higher energy efficiency than full precision arithmetic.

***Index Terms*—Low-bit Training, Quantization, Convolutiuonal Neural Networks**

I. Introduction

Convolutional neural networks (CNNs) have achieved state-of-the-art performance in many computer vision tasks [1]–[3]. However, deep CNNs are both computation and storage-intensive. The training process could consume up to hundreds of ExaFLOPs of computations and tens of GBytes of storage [4], thus posing a tremendous challenge for training in resource-constrained environments. At present, GPU is commonly used to train CNNs, and it is energy-hungry. The power of a running GPU is about 250W, and it usually takes more than 10 GPU-days to train one CNN model on large practical datasets like ImageNet [5]. Therefore, reducing the energy consumption of the training process has raised interest in recent years. Quantization of CNNs has drawn significant attention since it can reduce both the storage and computational complexity. It is pointed out that 32-bit floating-point multiplication and addition units consume about $20 \sim 30\times$ more power than 8-bit fixed-point ones [6]. Also, using an 8-bit data format could save the energy consumption of memory access by roughly four times.

All authors were with the Department of Electronic and Computer Engineering, Tsinghua University, Beijing, China, and Beijing National Research Center for Information Science and Technology (BNRist) (e-mail: yu-wang@tsinghua.edu.cn).

TABLE I
The number of different operations in the training on ImageNet. Abbreviations: "Conv": convolution; "BN": batch nomalization; "FC": fully connected layer; "EW-Add": element-wise addition; "F": forward; "B": backward.

Op Name	Op Type	ResNet18	GoogleNet
Conv (F)	Mul&Add	1.88E+09	1.58E+09
Conv (B)	Mul&Add	4.22E+09	3.05E+09
BN	Mul&Add	3.06E+06	3.23E+06
FC	Mul&Add	5.12E+05	1.02E+06
EW-Add (F)	Add	7.53E+05	0
EW-Add (B)	Add	9.28E+05	0
SGD Update (B)	Mul&Add	1.15E+07	5.97E+06

Many studies [7]–[10] focus on amending the training process to acquire a reduced-precision model with higher inference efficiency. However, these methods rely on tuning from a full precision pre-trained model, which is costly or introduces more optimization operations into training for a better inference performance. Therefore, they are not suitable for efficient training. Besides the studies on improving inference efficiency, some studies focus on the training process. WAGE and FullINT [11], [12] implement fully fixed-point training with 8-bit and 16-bit integers to reduce the training cost. However, they fail to achieve an acceptable accuracy since the dynamic range of data in training is large, and the SGD algorithm needs a small quantization error to ensure convergence. **This contradiction between the large dynamic range requirement of the training algorithm and the small representation range of high-efficiency integer data format is the first challenge of low-bit training.**

The floating-point format has a larger representation range than the fixed-point format with the same bit-width. FP8, HFP8, and S2FP8 [13]–[15] adopt 8-bit floating-point multiplications in convolution, in which more element-wise exponent bits are used to get a larger dynamic range. However, the precision of the effective number is lost, and they have to use a complex quantization format. On the other hand, the dynamic

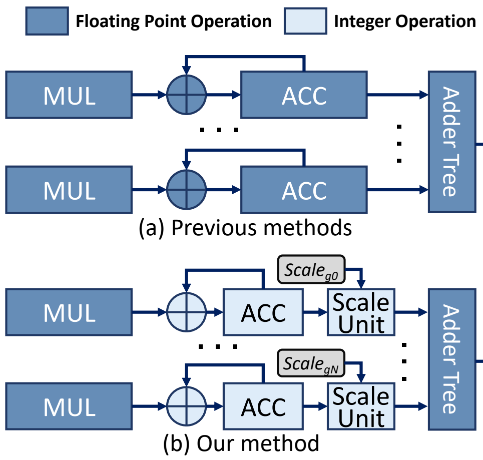

Fig. 1. The adder tree convolution hardware architecture. (a) Previous studies [14] use low-bit floating-point multiplication (FP MUL) (e.g., 8-bit), but single precision accumulations are still needed. (b) We not only makes MUL less than 8-bit, but also simplify the local accumulator.

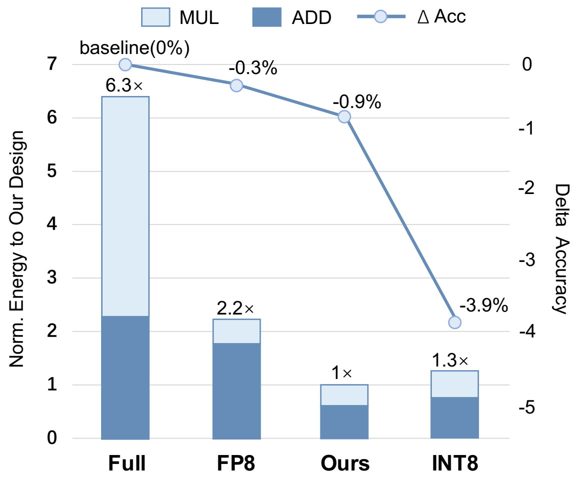

Fig. 2. The model accuracy drop (ResNet-18 on ImageNet) and energy consumption of calculating 3×3 convolutions with different arithmetic, nomalized to our design. FP8: [14]; Int8: [12].

range of intermediate accumulation results is too large. It can only be conducted in the floating-point format, which results in higher energy consumption than integer accumulation. The second challenge of low-bit training is **to realize low-cost multiplications and accumulations (MACs) for floating-point data formats with large dynamic ranges.**

In this work, we design a novel low-bit tensor format with multi-level scaling (MLS format) to maintain a high representation capability, which can be manipulated by our customized hardware design with high energy efficiency. In the MLS format: **1)** Element-wise scaling is used to increase the dynamic range of data representation, significantly reducing the quantization error. **2)** A specially designed group scaling factor is used to reduce the element-wise exponent bits with negligible overhead so that the accumulation can be simplified to integer accumulation without hurting the representational capability. Also, the group scaling could be conducted efficiently by shifting and additions instead of multiplications. Thus we can reduce the energy consumption of most computing unitswhile keeping the overall quantization error small to achieve accurate and efficient calculation.

Common hardware (e.g., GPU) is designed to support general floating-point arithmetic and can not efficiently support most of the existing low-bit tensor format. Furthermore, the systolic array architecture widely used in neural network accelerator treats convolution as general matrix multiplication and can not support group-wise scaling. **Hence, the third challenge of low-bit training is that common hardware does not support specially designed data formats with group-wise scaling.** To this end, we design **3)** the hardware of low-bit tensor convolution to support our training framework efficiently. Our computing unit consists of low-bit multiplication, integer intra-group accumulation, group-wise scale unit, and inter-group addition tree, as shown in Fig. 1 (b). Different from previous methods with similar architecture (Fig. 1 (a)), our multiplications have smaller bit-width, and the accumulations are conducted with integer arithmetic instead of floating-point one. Therefore, as shown in Fig. 2, our framework can significantly reduce the energy consumption of convolution operations compared with other frameworks.

To summarize, the contributions of this paper are:

1) This paper proposes the **MLS tensor format** to strike a superior balance between representation capability and energy efficiency. **The element-wise scaling** improves the dynamic range of data representation. Furthermore, using **the group-wise scaling** results in a low bit-width of element-wise exponents so that the intra-group accumulation can be conducted with integer accumulation for higher energy efficiency. We elaborate on the corresponding low-bit training framework and analyze that the MLS tensor format can be manipulated efficiently with our **low-bit tensor convolution arithmetic**.
2) Experimental results demonstrate the representational capability of the MLS format: For training ResNets, VGG-16, and GoogleNet on CIFAR-10, using 1-bit mantissa and 2-bit exponent for each element can achieve an accuracy loss within 1%. For training these models on ImageNet, 4-bit mantissa and 2-bit exponent are adequate to achieve an accuracy loss within 1%.
3) We conduct **hardware design of low-bit tensor convolution arithmetic with MLS format** and shows that our framework can indeed compute MACs in convolutions efficiently, without degrading the model accuracy. We implement Register Transfer Language (RTL) designs of accelerators with different arithmetic. The energy consumption simulation shows that training a variety of models with our framework could achieve up to $10.2\times$ and $1.2\times$ higher energy efficiency than training with 32-bit and 8-bit [14] floating-point arithmetic. Furthermore, we can achieve much higher accuracy than previous fixed-point training frameworks [11], [12].

The correspondences of the challenges in low-bit training and our contributions are summarized in Fig. 3. The rest of

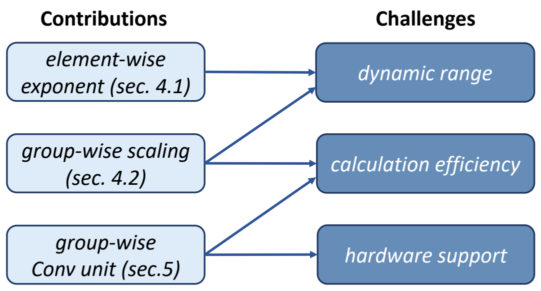
Contributions
Challenges
element-wise exponent (sec. 4.1)
group-wise scaling (sec. 4.2)
group-wise Conv unit (sec.5)
dynamic range
calculation efficiency
hardware support

Fig. 3. Contributions of this paper and challenges of low-bit training.

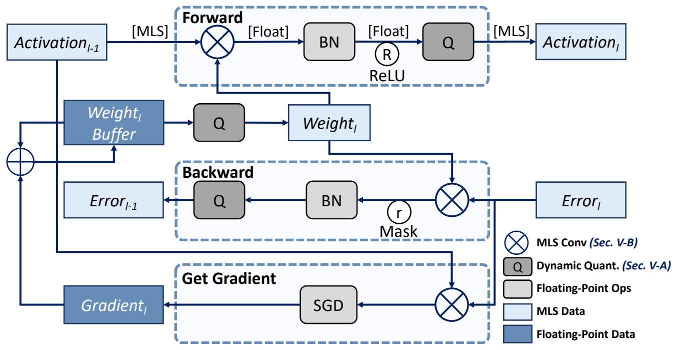

Fig. 4. Computation flow of our low-bit training framework.

this paper is organized as follows. Sec. II discusses the related work of low-bit training, and Sec. III gives basic knowledge on the training framework of CNNs. Then, Sec. IV explains why we proposed element-wise scaling and group-wise scaling in CNN low-bit training and summarizes them in MLS tensor format. The corresponding convolution arithmetic unit design is proposed in Sec. V. The dynamic quantization method and its overhead are discussed in Sec. V-A, and the experiment results are shown in Sec. VI. Finally, we conclude in Sec. VIII.

II. Related Work

A. Post-Training Quantization

Earlier quantization methods [16] focus on the post-training quantization (PTQ) setting and quantize the pre-trained full precision model using the codebook generated by clustering or other criteria (e.g., SQNR [17], entropy [18]). POST [9] and HAWQ [10] select each channel's bit-width and clipping value through the analytical investigation. GEMMLOWP [19] proposes an integer convolution arithmetic for efficient inference. However, it is hard to be used in training because the scale of the output tensor should be known before calculation. MSFP [20] proposes a data format developed for the cloud-scale inference on custom hardware. These methods show that low-bit CNN models still have adequate representation ability. However, these methods aim to accelerate the inference of a pre-trained model and can not accelerate the training process.

B. Quantization-Aware Training

Quantization-aware training (QAT) considers quantization effects in the training process to further improve the accuracy of the quantized model. It is used for training binary [21] or ternary [22] networks. Although the follow-up studies [23], [24] have proposed new techniques to improve the accuracy, the extremely low bit-width still causes notable accuracy degradation. Other methods seek to retain the accuracy with relatively higher precision, e.g., 8-bit [7]. [25] develops a GPU-based training framework to get dynamic fixed-point models or Minifloat models. [8] parameterizes and trains the clipping value in the activation function to properly restrict the range of activation. These methods obtain quantized models that achieve better trade-off of accuracy and inference efficiency, but the training process is still full precision.

C. Low-Bit Training

To accelerate the training process, [26] proposes to use fixed-point arithmetic in both the forward and backward processes. [11], [12] implement full-integer training frameworks for integer-arithmetic machines. However, these methods cause notable accuracy degradation. So instead, [27] uses 8-bit and 16-bit integer arithmetic [19] and achieves better accuracy. Nevertheless, this arithmetic [19] is designed for accelerating inference and requires knowing the output scale before calculation. Therefore, although [27] quantizes the gradients in the backward process, it is not practical for actual training acceleration. To summarize, full-integer training frameworks have high energy efficiency but still suffer from considerable accuracy degradation when the bit-width is reduced to 8 bit.

Besides the studies on full-integer training frameworks, some studies propose new low-bit formats. BFloat [28] uses a 16-bit floating-point format that is more suitable for CNN training. Flexpoint [29] proposes the format that contains 16-bit mantissa and 5-bit tensor-shared exponent (scale), which is similar to the dynamic fixed-point format [30]. Recently, 8-bit floating-point formats [13]–[15] are used with chunk-based accumulation. However, to ensure a sufficient representation range, the exponent bit-width in their format is larger than 5, making the operations (especially the accumulation) using these formats inefficient. More recently, a radix-4 data format [31] is proposed along with two-stage quantization to realize 4-bit training, but the accuracy is not satisfying enough, and its computation is complex. In this work, the MLS tensor format is designed to have a small exponent bit-width. Thus, the accumulation can be conducted using fixed-point arithmetic while retaining the overall model accuracy.

III. Preliminary

A. Computation Flow for CNN Training

This paper denotes the filter parameters and feature map of convolution as weight and activation, respectively. In the back-propagation, the gradients of activation and weight are denoted as error and gradient, respectively. As shown in Fig. 4, generally, in a convolutional layer, convolution is followed by batch normalization (BN), nonlinear activation (e.g., ReLU), and other operations (e.g., pooling). As shown in Table I, the

MACs in convolutions are the majority of operations in CNN training. Hence, conducting MACs in convolutions with low-bit arithmetic is promising for improving energy efficiency. And retaining other operations (e.g., BN, weight update) using high bit-width helps stabilize training. Therefore, we conduct quantization before all three types of Conv (Conv of weight and activation, weight and error, activation and error). The input tensors of Conv are all in low-bit format, and the output tensor of Conv is floating-point since the convolution calculation will increase the data precision. Other operations such as BN are all performed on floating-point tensors.

B. Basic Formula of Convolution

Weight, activation, and error are all 4-dimension tensors in the training process. For the activation and error, the four dimensions are sample in batch (N), channel (C), feature map height (F_h), and feature map width (F_w). For weight, the four dimensions are output channel (C_o), input channel (C_i), kernel height (K_h), kernel width (K_w). We take Conv(Weight, Activation) ($Conv(\boldsymbol{W}, \boldsymbol{A})$) as the example to introduce the basic convolution formula, and the other two types of convolution can be implemented similarly. Denoting the input channel number as C and the kernel size as $K = K_h = K_w$, the original formula of convolution is

$$\begin{aligned}\boldsymbol{Z}[n, co, x, y] = Conv(\boldsymbol{W}, \boldsymbol{A}) = \sum_{ci=0}^{C-1}\sum_{i=0}^{K-1}\sum_{j=0}^{K-1} \\ \boldsymbol{W}[co, ci, i, j] \times \boldsymbol{A}[n, ci, x+i, y+j]\end{aligned} \quad (1)$$

We can see that every element in the output 4-dimension tensor is calculated by three loops of MACs. And three dimensions of input tensors are included in this accumulation. On common hardware platforms like TPU [32] and GPU, these tensors are processed with the “image to column” transformation. Then the convolution is calculated as a general matrix multiplication, in which grouping techniques cannot be used. However, in many customized CNN accelerators [33], [34], parallel PE units and addition tree architecture are used. The MACs can be grouped into intra-group ones and inter-group ones, makeing it possible for us to apply group-wise scaling. Next, we will show the advantages of group-wise scaling through data format design and hardware design.

IV. Mulit-level Scaling Low-bit Tensor Format

Using low-bit arithmetic in the training process is beneficial for energy efficiency. However, retaining a good accuracy in a low-bit fixed-point training process is challenging since the backpropagated gradients need high precision [26]. In this work, we design an MLS low-bit tensor format to retain the representational power of low-bit representations in CNNs. It consists of three levels of scaling factors: 1) Tensor-wise scaling factor; 2) Group-wise scaling factor; 3) Element-wise exponent. By incorporating the multi-level scaling technique, the element-wise bit-width can be largely reduced to boost the energy efficiency while the overall dynamic range is preserved.

This section gives the design details of the MLS low-bit tensor format, which is the core of our low-bit training framework. And in the next section Sec. V, we will elaborate on the framework and hardware design centering around the MLS format to demonstrate that the conversion and computation of the MLS-formatted data are energy efficient.

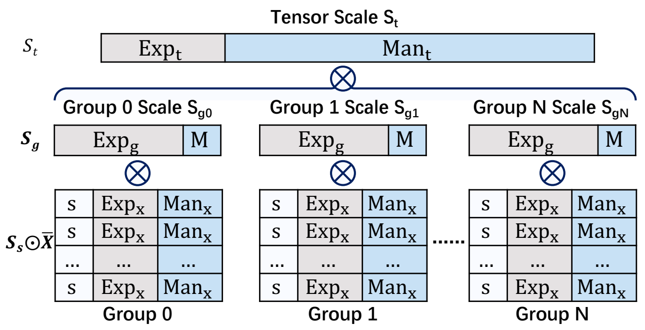

Fig. 5. The multi-level scaling (MLS) low-bit tensor format.

A. Overall Mapping Formula of the MLS Format

In a commonly used scheme [19], the mapping function from fixed-point representation and the floating-point values is $float = scale \times (Fix + Bias)$, in which $scale$ and $Bias$ are shared in one tensor. In training, however, since data distribution changes over time, one cannot simplify the $Bias$ calculation as they do. Thus, we adopt a quantization scheme without bias and extend the scaling factor to three levels for better representation ability. The resulting MLS tensor format is illustrated in Fig. 5. Denoting a 4-dimensional tensor that is the operand of Conv (weight, activation, or error) as $\boldsymbol{X}$, the mapping formula of the MLS tensor format is

$$\boldsymbol{X}[i, j, k, l] = \boldsymbol{S_s}[i, j, k, l] \times S_t \times \boldsymbol{S_g}[i, j] \times \bar{\boldsymbol{X}}[i, j, k, l] \quad (2)$$

where $[\cdot]$ denotes the indexing operation, $\boldsymbol{S_s}$ is a sign tensor (“s” in Fig. 5), S_t is a tensor-wise scaling factor, and $\boldsymbol{S_g}$ are group-wise scaling factors shared in each group. $\boldsymbol{S_g}$ and $\bar{\boldsymbol{X}}$ use the same data format, which we refer to as $\langle E, M\rangle$, a customized floating-point format with E-bit exponent and M-bit mantissa (no sign bit). A value in the format $\langle E, M\rangle$ is

$$\begin{aligned}float = I2F(Man, Exp) &= Frac \times 2^{-Exp} \\ &= \left(1 + \frac{Man}{2^M}\right) \times 2^{-Exp}\end{aligned} \quad (3)$$

where Man and Exp are the M-bit mantissa and E-bit exponent, and $Frac \in [1, 2)$ is a fraction.

B. Group-wise Scaling

The dynamic ranges of weight, activation, and error are large in training, but these values are not evenly distributed. The values in different groups have distinct dynamic ranges, as shown in Fig. 6. The blue line shows the max value in each group when activation and error are grouped by channel or sample. If we use the overall maximum value (green lines in Fig. 6) as the overall scaling, many small elements will be swamped. Furthermore, there are usually more than half groups in which all elements are smaller than half of the

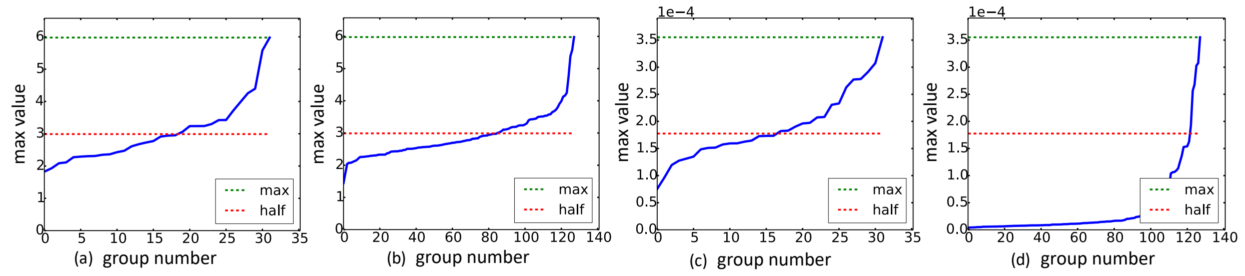

Fig. 6. Maximum value of each group of activation (a, b) and error (c, d). (a)(c): Grouped by channel; (b)(d): Grouped by sample.

overall maximum (red line). Thus, to fully exploit the bit-width, it is natural to use group-wise scaling factors. Our work considers three grouping dimensions: 1) the sample dimension of activation and error tensors or the output channel dimension of weight tensors, 2) the channel dimension of tensors, 3) the sample and the channel dimension simultaneously.

Naive floating-point group-wise scaling in previous studies [21] cannot bring actual hardware acceleration. Since when the values of different groups are accumulated, the floating-point scaling factors need to be multiplied with elements, which involves floating-point multiplications. To facilitate a hardware-friendly low-bit training framework, we propose a special scaling format in which the floating-point group-wise scaling is separate into tensor-wise and group-wise scaling factors. The first level **tensor-wise scaling factor** S_t is a standard floating-point number ($\langle E_t, M_t\rangle = \langle 8, 23\rangle$). Considering the actual hardware implementation cost, we propose two hardware-friendly schemes for the **group-wise scaling factor** $\boldsymbol{S_g}$, which can be denoted as $\langle E_g, 0\rangle$, and $\langle E_g, 1\rangle$. The scaling factor in $\langle E_g, 0\rangle$ format is a power of two, which can be implemented as shifting. From Eq. 3, a $S_g = I2F(Man_g, Exp_g)$ value in the $\langle E_g, 1\rangle$ format can be written as

$$\begin{aligned} S_g &= \left(1 + \frac{Man_g}{2}\right) \times 2^{-Exp_g} \\ &= \begin{cases} 2^{-Exp_g} + 2^{-Exp_g - 1} & Man_g = 1 \\ 2^{-Exp_g} & Man_g = 0 \end{cases} \end{aligned} \tag{4}$$

which is a sum of two shiftings and can be implemented with low hardware overhead. We will show how the convolution arithmetic of MLS tensors benefits from this special format of group-wise scaling factors in Sec. V-B (Eq. 8).

C. Element-wise Scaling

The third level scaling factor $S_x = I2F(0, Exp_x) = 2^{-Exp_x}$ is the **element-wise exponent** in $\bar{X} = S_x(1+\frac{Man_x}{2})$, and we can see that the elements of $\bar{\boldsymbol{X}}$ in Eq. 2 are in a $\langle E_x, M_x\rangle$ format. The specific values of E_x and M_x determine the cost of the MAC operation, which will be discussed in Sec. V-B. Compared with integer data format ($E_x = 0$), adding element-wise exponent helps achieve a balance in the dynamic range and precision of representation. Furthermore, by using group-wise scaling, the bit-width of $\bar{\boldsymbol{X}}$ can be largely reduced.

Algorithm 1: The low-bit training framework

```
Input: $L$: number of layers; $\boldsymbol{W^{1:L}}$: current float
       weights; $\boldsymbol{A^0}$: inputs; $\boldsymbol{T}$: label; $lr$: learning rate
Output: $\boldsymbol{W^{1:L}_{t+1}}$: updated float weights for step $t+1$
/* forward propagation */
1  for $l$ in $1:L$ do
2      $q\boldsymbol{W^l} = DynamicQuantization(\boldsymbol{W^l})$
3      $q\boldsymbol{A^{l-1}} = DynamicQuantization(\boldsymbol{A^{l-1}})$
4      $\boldsymbol{Z^l} = LowbitConv(q\boldsymbol{W^l}, q\boldsymbol{A^{l-1}})$
5      $\boldsymbol{Y^l} = BatchNorm(\boldsymbol{Z^l})$
6      $\boldsymbol{A^l} = Activation(\boldsymbol{Y^l})$
7  end
8  $\frac{\partial loss}{\partial \boldsymbol{A^l}} = Criterion(\boldsymbol{A^L}, \boldsymbol{T})$
/* backward propagation */
   for $l$ in $L:1$ do
9      $\frac{\partial loss}{\partial \boldsymbol{Y^l}} = \frac{\partial loss}{\partial \boldsymbol{A^l}} \times Activation'(\boldsymbol{Y^l})$
10     $\frac{\partial loss}{\partial \boldsymbol{Z^l}} = \frac{\partial loss}{\partial \boldsymbol{Y^l}} \times \frac{\partial \boldsymbol{Y^l}}{\partial \boldsymbol{Z^l}}$
11     $q\boldsymbol{E^l} = DynamicQuantization(\frac{\partial loss}{\partial \boldsymbol{Z^l}})$
12     $\boldsymbol{G^l} = LowbitConv(q\boldsymbol{E^l}, q\boldsymbol{A^{l-1}})$
13     $\boldsymbol{W^l_{t+1}} = \boldsymbol{W^l} - lr \times \boldsymbol{G^l}$
14     if $l$ is not $1$ then
15         $\frac{\partial loss}{\partial q\boldsymbol{A^{l-1}}} = LowbitConv(q\boldsymbol{E^l}, q\boldsymbol{W^l})$
16         $\frac{\partial loss}{\partial \boldsymbol{A^{l-1}}} = STE(\frac{\partial loss}{\partial q\boldsymbol{A^{l-1}}})$
17     end
18 end
   Return $\boldsymbol{W^{1:L}_{t+1}}$
```

V. Low-bit Training Framework

This section describes the low-bit training framework to leverage the MLS tensor format. Alg. 1 summarizes a training iteration in our low-bit training framework, and Fig. 4 shows the computation flow of one layer. Our framework is different from a quantization-aware training framework in that the convolution operands are actually quantized to the low-bit MLS format in our computation flow. In the algorithmic description (Alg. 1, Line 13), we use the formula of the vanilla stochastic gradient descent (SGD) for clarity, whereas in practice, one can use other optimizers. $STE(\cdot)$ in Line 16 stands for the Straight Through Estimator widely used in QAT methods [21], [26]. Finally, the t subscripts denoting the time step t are omitted for simplicity. We describe two core parts of the framework

Algorithm 2: Dynamic Quantization

```
Input: $\boldsymbol{X}$: float 4-d tensor; $\boldsymbol{R}$: $U[-\frac{1}{2},\frac{1}{2}]$ distributed
       random tensor; $\langle E_g, M_g\rangle$: bit-width of
       group-wise scaling factors; $\langle E_x, M_x\rangle$: bit-width
       of each element
Output: $\boldsymbol{S_s}$: sign tensor; $S_t$: tensor-wise scaling
        factor; $\boldsymbol{S_g}$: group-wise scaling factors; $\bar{\boldsymbol{X}}$:
        quantized elements
/* calculating scaling factors */
1  $\boldsymbol{S_s} = Sign(\boldsymbol{X})$, $\boldsymbol{S_r} = GroupMax(Abs(\boldsymbol{X}))$
2  $S_t = Max(\boldsymbol{S_r})$, $\boldsymbol{S_{gf}} = \boldsymbol{S}_r \div S_t$
3  $\boldsymbol{Exp_g}, \boldsymbol{Frac_g} = Exponent(\boldsymbol{S_{gf}}), Fraction(\boldsymbol{S_{gf}})$
4  $\boldsymbol{Exp_g} = Clip(\boldsymbol{Exp_g}, 1-2^{E_g}, 0)$
5  $\boldsymbol{Frac_g} = Ceil(\boldsymbol{Frac_g} \times 2^{M_g}) \div 2^{M_g}$
6  $\boldsymbol{S_g} = \boldsymbol{Frac_g} \times 2^{\boldsymbol{Exp_g}}$
/* calculating elements */
7  $\boldsymbol{X_f} = Abs(\boldsymbol{X}) \div \boldsymbol{S_g} \div S_t$
8  $\boldsymbol{Exp_x}, \boldsymbol{Frac_x} = Exponent(\boldsymbol{X_f}), Fraction(\boldsymbol{X_f})$
   // quantize $\boldsymbol{Frac_x}$ to $M_x$ bits with underflow
      handling
9  $E_{xmin} = 1 - 2^{E_x}$
10 $\boldsymbol{Frac_{xs}} = \boldsymbol{Frac_x} \times 2^{M_x}$ if not underflow, else
      $\boldsymbol{Frac_x} \times 2^{M_x - E_{xmin} + E_x}$
11 $\boldsymbol{Frac_{xint}} = Clip(SRound(\boldsymbol{Frac_{xs}}, \boldsymbol{R})), 0, 2^{M_x} - 1)$
12 $\boldsymbol{Frac_x} = \boldsymbol{Frac_{xint}} \times 2^{-M_x}$ if not underflow, else
      $\boldsymbol{Frac_{xint}} \times 2^{-M_x + E_{xmin} - E_x}$
13 $\boldsymbol{Exp_x} = Clip(\boldsymbol{Exp_x}, E_{xmin}, -1)$
14 $\bar{\boldsymbol{X}} = \boldsymbol{Frac_x} \times 2^{\boldsymbol{Exp}_x}$
Return $\boldsymbol{S_s}$, $S_t$, $\boldsymbol{S_g}$, $\bar{\boldsymbol{X}}$
```

to demonstrate why the conversion and computation of our proposed format are energy efficient: Sec. V-A describes the dynamic quantization $DynamicQuantization$ (FP-to-MLS conversion), and Sec. V-B describes the convolution arithmetic $LowbitConv$ (MLS-to-FP conversion).

A. Dynamic Quantization to MLS Tensor

The dynamic quantization (DQ) converts a floating-point tensor to an MLS tensor. The two main steps are calculating the scaling factors $\boldsymbol{S_s}, S_t, \boldsymbol{S_g}$ and getting the quantized elements $\bar{\boldsymbol{X}}$, as shown in Alg. 2. In Alg. 2, the sign tensor, overall maximum, and group-wise maximums are got in line 1∼2. And group-wise scaling factors are quantized by group-wise maximums in line 3∼6. $Exponent(\cdot)$ and $Fraction(\cdot)$ are to obtain the exponent (an integer) and fraction (an integer representing numbers $\in [1,2)$) of a floating-point number, which is used in the quantization of group-wise scalings and element-wise numbers in line 3 and line 8. The underflow handling follows the IEEE 754 standard [35], as shown in line 9∼13. And stochastic rounding $SRound(x,r)$ is used when calculating the quantized elements $\bar{\boldsymbol{X}}$ in line 11. [36] proposed this technique because stochastic rounding is an unbiased rounding scheme, thus the overall direction of gradient descent in training can be more accurate. It is implemented with a uniformly distributed random tensor $r \sim U[-0.5, 0.5]$ which can be generated offline as how it is done on GPU. Note that Alg. 2 describes how we simulate the dynamic quantization process on a floating-point platform. While on the hardware, the exponent and mantissa could be obtained directly, and $Clip$ is conducted by simply truncating some bits of a number.

B. Low-bit Tensor Convolution Arithmetic

In this section, we describe how to do convolution with two low-bit MLS tensors. Using the MLS tensor format and denoting the corresponding values (scaling factors $\boldsymbol{S}$, exponents $\boldsymbol{Exp}$ and fractions $\boldsymbol{Frac}$ in the following equations) of $\boldsymbol{W}$ and $\boldsymbol{A}$ by the superscript $^{(w)}$ and $^{(a)}$, one output element $\boldsymbol{Z}[n,co,x,y]$ of $Conv(\boldsymbol{W},\boldsymbol{A})$ is calculated as

$$\begin{aligned}\boldsymbol{Z}[n,co,x,y] &= \sum_{ci=0}^{C-1}\sum_{i=0}^{K-1}\sum_{j=0}^{K-1}\left(S_t^{(w)}\boldsymbol{S_g^{(w)}}[co,ci]\bar{\boldsymbol{W}}[co,ci,i,j]\right)\\ &\qquad\left(S_t^{(a)}\boldsymbol{S_g^{(a)}}[n,ci]\bar{\boldsymbol{A}}[n,ci,x+i,y+j]\right)\\ &= \left(S_t^{(w)}S_t^{(a)}\right)\sum_{ci=0}^{C-1}[\left(\boldsymbol{S_g^{(w)}}[co,ci]\boldsymbol{S_g^{(a)}}[n,ci]\right)\\ &\qquad\sum_{i=0}^{K-1}\sum_{j=0}^{K-1}\bar{\boldsymbol{W}}[co,ci,i,j]\bar{\boldsymbol{A}}[n,ci,x+i,y+j]]\\ &= S_t^{(z)}\sum_{ci=0}^{C-1}\boldsymbol{S^{(p)}}[n,co,ci]\boldsymbol{P}[n,co,ci]\end{aligned} \tag{5}$$

Eq. 5 shows that there are **intra-group** MACs that calculate $\boldsymbol{P}[n,co,ci]$ and **inter-group** MACs that calculate $\boldsymbol{Z}$.

Intra-group MACs The intra-group calculation of $\boldsymbol{P}$ is

$$\boldsymbol{P}[n,co,ci] = \sum_{i,j=0}^{K-1}\left(\boldsymbol{Frac^{(w)}}[co,ci,i,j]\boldsymbol{Frac^{(a)}}[n,ci,i,j]\right)\times 2^{\left(\boldsymbol{Exp^{(w)}}[co,ci,i,\right.}$$

(6)

where $\boldsymbol{Frac}$, $\boldsymbol{Exp}$ are (M_x+1)-bit and E_x-bit. The intra-group calculation contains the multiplication of two (M_x+1)-bit values and $2\times(2^{E_x}-2)$-bit shifting. The resulting $(2M_x + 2^{E_x+1} - 2)$-bit integer values need to be accumulated with enough bit-width to get the partial sum $\boldsymbol{P}$. In previous 8-bit floating-point frameworks [13], a floating-point accumulator is needed since they use $E_x = 5$, and $(2M_x + 2^{E_x+1} - 2)$ is larger than 64. In contrast, we can use an integer accumulator since we adopt $E_x = 2, M_x = 4$ in the MLS tensor format on ImageNet so that the resulting integer to be accumulated is 14-bit. See Sec. V-C for detailed analysis.

Inter-group MACs As for the inter-group calculation, each element in $\boldsymbol{S^{(p)}}$ is a $\langle E, 2\rangle$ number obtained by multiplying two $\langle E, 1\rangle$ numbers.

$$\begin{aligned}\boldsymbol{Z}[co,x] &= S_t^{(z)}\sum_{ci=0}^{C-1}\boldsymbol{S^{(p)}}[co,ci]\boldsymbol{P}[x,ci]\\ &= S_t^{(z)}\sum_{ci=0}^{C-1}\left(1+\frac{Man^{(p)}[co,ci]}{4}\right)\times 2^{-Exp^{(p)}[co,ci]}\boldsymbol{P}[x,ci]\end{aligned} \tag{7}$$

Note that this can be calculated by shift (multiplying the power of two) and addition as

$$= S_t^{(z)} \sum_{ci=0}^{C-1} \begin{cases} \boldsymbol{P}[x,ci]2^{-\boldsymbol{Exp^{(p)}}[co,ci]} \\ \boldsymbol{P}[x,ci]2^{-\boldsymbol{Exp^{(p)}}[co,ci]} + \boldsymbol{P}[x,ci]2^{-\boldsymbol{Exp^{(p)}}[co,ci]-1} \\ \boldsymbol{P}[x,ci]2^{1-\boldsymbol{Exp^{(p)}}[co,ci]} + \boldsymbol{P}[x,ci]2^{-\boldsymbol{Exp^{(p)}}[co,ci]-2} \end{cases} \tag{8}$$

where the three cases correspond to $\boldsymbol{Man^{(p)}}[co,ci]$=00, $\boldsymbol{Man^{(p)}}[co,ci]$=10, and $\boldsymbol{Man^{(p)}}[co,ci]$=01, respectively. The index n is omitted for simplicity, and x denotes the original 2-dimension spatial indexes x, y.

Summary In the MLS format, *the element-wise exponent* is 2-bit instead of 5-bit. Thus the **intra-group accumulation** is simplified to use 32-bit integers. On the other hand, due to the special format of the *group-wise scaling factor*, $\boldsymbol{S^{(p)}}$ has a simple format, and the **inter-group accumulation** to calculate $\boldsymbol{Z}$ can be implemented efficiently on hardware without floating-point multiplication. Finally, the multiplication with the *tensor-wise floating-point scaling factor* $S_t^{(z)}$ in Eq. 8 can be omitted as long as there is no following element-wise addition on Z with another tensor. $S_t^{(z)}$ only needs to be multiplied with the tensor-wise floating-point scale in the following layer instead of the tensor Z, which can be expressed as $TensorScale(S_t^{(z)} \times Z) = S_t^{(z)} \times TensorScale(Z)$. Only when an element-wise addition follows Conv (e.g., in ResNets) does the multiplication with $S_t^{(z)}$ need to be executed.

C. Analysis of Accumulation Bit-Width

When different data formats are used, the results of multiplication in Conv have different dynamic ranges. As specified by the IEEE 754 standard, the gradual underflow behavior of a floating-point number with M-bit mantissa (Man) and E-bit exponent (Exp) is as follows. If Exp is not the minimum value, the float value is not underflowed and is calculated as

$$\begin{aligned} float &= Frac \times 2^{-Exp} \\ &= \left(1 + \frac{Man}{2^M}\right) \times 2^{-Exp}. \end{aligned} \tag{9}$$

If Exp is the minimum value, the float value is a gradual-underflowed value and is calculated as

$$\begin{aligned} float &= Frac \times 2^{-Exp} \\ &= \left(0 + \frac{Man}{2^M}\right) \times 2^{-Exp}, \end{aligned} \tag{10}$$

where $Frac$ is a $(M+1)$-bit fraction, calculated by adding a 0 or 1 at the highest bit of mantissa.

The production of two numbers is calculated as

$$\begin{aligned} float_1 \times float_2 &= Frac_1 \times 2^{-Exp_1} \times Frac_2 \times 2^{-Exp_1} \\ &= (Frac_1 \times Frac_2) \times 2^{-Exp_1 - Exp_2}, \end{aligned} \tag{11}$$

where $Frac_1 \times Frac_2$ is a $(M+1)$-bit multiplication, and the result is $(2M+2)$-bit. Since the minimum value of the exponent is used to represent underflow, following the IEEE 754 standard [35], an E-bit Exp number has $2^E - 1$ levels. That means "$Frac \times 2^{-Exp}$" enables $Frac$ to be shifted by up to $(2^E - 2)$ bits. Therefore, "$Frac \times 2^{-Exp_1 - Exp_2}$" enables $Frac$ to be shifted by up to $(2^{E+1} - 4)$ bits, and the final result of floating-point multiplication has a dynamic range of $2M+2+2^{E+1}-4 = (2M+2^E-2)$-bit. These resulting $(2M+2^{E+1}-2)$-bit integers need to be accumulated with enough bit-width to get the partial sum. In previous 8-bit floating-point frameworks, a floating-point accumulator is needed since they use $E = 5$. In contrast, we can use a 32-bit integer accumulator since we adopt $E = 2, M = 4, (2M + 2^{E+1} - 2) = 14$ in the MLS tensor format on ImageNet.

VI. Experiments

A. Experimental Setup

1) Training settings: We train ResNet [37], VGG [4], and GoogleNet [38] on CIFAR-10 [39] and ImageNet [5]. In all the experiments, the first and the last layer are unquantized following previous studies [14], [26], [40]. On both CIFAR-10 and ImageNet, SGD with momentum 0.9 and weight decay 5e-4 is used, and the initial learning rate is set to 0.1. We train the models for 90 epochs on ImageNet and decay the learning rate by 10 every 30 epochs. These training hyper-parameters are proposed in the ResNet paper [37]. On CIFAR-10, we train the models for 160 epochs and decay the learning rate by 10 at epoch 80 and epoch 120.

2) Data format settings: The group-wise scaling is in $\langle 8, 1\rangle$ format, and stochastic rounding is only used in error quantization. We experiment using different $\langle E_x, M_x\rangle$ configurations on CIFAR-10. We adopt $\langle 2, 4\rangle$ format on ImageNet since we found that $\langle 2, 4\rangle$ is sufficient for training commonly used models, and our hardware evaluation is based on this setting. For a new task, the average relative quantization error (ARE) in the ablation study (Sec. VI-C) can be used to guide the selection of bit configurations. We adopt the same quantization bit-width for weight, activation, and error for a simple hardware design. The computation flow of the low-bit training experiment is shown in Fig. 4. All weight, activation, and error tensors are quantized (DQ in Sec. V-A) before Conv, and the result tensors of low-bit convolution arithmetic are in full precision (Sec. V-B). Other operations, such as BN, are calculated with full precision.

B. Results on CIFAR-10 and ImageNet

1) Comparison with previous studies: The training results on CIFAR-10 and ImageNet are shown in Table II. The floating-point accuracy of baselines are directly taken from their papers. As there are slight differences in the floating-point accuracy, we compare the accuracy drop of different methods. We can see that our method can achieve a better balance between higher accuracy and lower bit-width. A previous study [26] finds that quantizing error to a low bit-width hurt the accuracy, but our method can quantize error to $M_x = 1$ on CIFAR-10, with a small accuracy drop of 0.48%, 0.55%, and 0.42% for ResNet-20, GoogleNet, and VGG-16, respectively.

TABLE II
COMPARISON OF LOW-BIT TRAINING METHODS ON CIFAR-10 AND IMAGENET. SINGLE NUMBER IN THE BIT-WIDTH STANDS FOR FIXED-POINT FORMAT BIT-WIDTH, WHICH IS EQUIVALENT TO M_x AND THE CORRESPONDING E_x IS 0. "F X" INDICATES THAT X-BIT FLOATING-POINT NUMBERS ARE USED. "ACCUM" IN THE "BIT-WIDTH" COLUMN STANDS FOR "ACCUMULATION", WHILE "ACC." STANDS FOR "ACCURACY".

Dataset	Method	Model	Bit-Width (W/A/E/ACCUM)	Acc.	FP baseline	Acc. Drop
CIFAR-10	S2fFP8 [15]	ResNet-20	$\langle 5,2\rangle$ $\langle 5,2\rangle$ $\langle 5,2\rangle$ f32	91.1%	91.5%	0.4%
	WAGE [11]	VGG-like	2 8 8 32	93.2%	94.1%	0.9%
	RangeBN [27]	ResNet-20	1 1 2 -	81.5%	90.36%	8.86%
	Ours	ResNet-20	4 4 4 16	92.32%	92.45%	0.13%
		ResNet-20	2 2 2 16	90.39%	92.45%	2.06%
		ResNet-20	$\langle 2,1\rangle$ $\langle 2,1\rangle$ $\langle 2,1\rangle$ 16	91.97%	92.45%	0.48%
		GoogleNet	$\langle 2,1\rangle$ $\langle 2,1\rangle$ $\langle 2,1\rangle$ 16	93.95%	94.50%	0.55%
		VGG-16	$\langle 2,1\rangle$ $\langle 2,1\rangle$ $\langle 2,1\rangle$ 16	93.34%	93.76%	0.42%
		VGG-16	$\langle 1,1\rangle$ $\langle 1,1\rangle$ $\langle 1,1\rangle$ 8	92.77%	93.76%	0.99%
ImageNet	FlexPoint [29]	AlexNet	16 16 16 32	80.1% (Top5)	79.9% (Top5)	-0.2%
	DFP16 [30]	VGG-16	16 16 16 32	68.2%	68.1%	-0.1%
	DFP16 [30]	GoogleNet	16 16 16 32	69.3%	69.3%	0
	RangeBN [27]	ResNet-18	8 8 16 f32	66.4%	67.0%	0.6%
	DoReFa [26]	AlexNet	8 8 8 32	53.0%	55.9%	2.9%
	FullINT [12]	ResNet-18	8 8 8 32	64.8%	68.7%	3.9%
	FullINT [12]	ResNet-34	8 8 8 32	67.6%	72.0%	4.4%
	WAGE [11]	AlexNet	2 8 8 32	48.4%	56.0%	7.6%
	HFP8 [14]	ResNet-18	$\langle 5,3\rangle$ $\langle 5,3\rangle$ $\langle 5,3\rangle$ f32	69.0%	69.3%	0.3%
	S2FP8 [15]	ResNet-18	$\langle 5,2\rangle$ $\langle 5,2\rangle$ $\langle 5,2\rangle$ f32	69.6%	70.3%	0.7%
	Ultra-Low [31]	ResNet-18	4 4 $\langle 3,1\rangle$ f16	68.3%	69.4%	1.1%
	Ours	ResNet-18	8 8 8 32	68.5%	69.1%	0.6%
		ResNet-18	4 4 4 16	66.5%	69.1%	2.6%
		ResNet-18	$\langle 2,4\rangle$ $\langle 2,4\rangle$ $\langle 2,4\rangle$ 32	68.2%	69.1%	0.9%
		ResNet-34	$\langle 2,4\rangle$ $\langle 2,4\rangle$ $\langle 2,4\rangle$ 32	75.3%	76.1%	0.8%
		VGG-16	$\langle 2,4\rangle$ $\langle 2,4\rangle$ $\langle 2,4\rangle$ 32	70.8%	70.9%	0.1%
		GoogleNet	$\langle 2,4\rangle$ $\langle 2,4\rangle$ $\langle 2,4\rangle$ 32	69.6%	69.5%	-0.1%

TABLE III
NUMBER OF OPERATIONS AND SENSITIVITY OF RESNETS, VGG-16, AND GOOGLENET.

Model	Inference GOPs	Acc. Drop of 6-bit Training
ResNet-18	1.88	0.9%
ResNet-34	3.59	0.8%
VGG-16	15.25	0.1%
GoogleNet	1.58	-0.1%

TABLE IV
ACCURACY OF TRAINING 200 EPOCHS ON IMAGENET WITH COSINE ANNEALING SCHEDULER.

Model	Full Precision Training	Our Low-Bit Training (Bit-Width is $\langle 2,4\rangle$)	Acc. Drop
ResNet-34	75.8%	75.6%	0.2%
VGG-16	72.1%	72.1%	0%
GoogleNet	73.6%	73.4%	0.2%

On ImageNet, the accuracy degradation of our method is minor under 8-bit quantization (0.6% accuracy drop from 69.1% to 68.5%), which is comparable with other state-of-the-art studies. In the cases with lower bit-width, our method achieves a higher accuracy (66.5%) with only 4-bit than [27] who uses 8-bit (66.4%). With $\langle 2,4\rangle$ data format, for all the models, including ResNet-18, ResNet-34, VGG-16, and GoogleNet, our method can achieve an accuracy loss less than 1%. In this case, the bit-width of the intermediate results is $2M_x + 2^{E_x+1} - 2 = 14$, which means that the accumulation can be conducted using integers, instead of floating-point numbers [14], [15], as we discussed in Sec. V-B. Although a previous work [31] quantizes W/A/E to 4-bit, the three types of Convs between them are in different data formats, requireing three different Conv unit implementations. In contrast, our work unifies the W/A/E format and the Conv calculation, thus requiring only one type of Conv unit.

2) Analysis of different CNN models: We note that the performance of VGG and GoogleNet CNN models in low-bit training is better than ResNets. We think this is because there are fewer channels in ResNet than VGG and GoogleNet when the network depth configuration is similar. And the little redundancy of ResNets makes them more sensitive to quantization errors. In fact, VGG-16 is 7 times as much as ResNet-18 in terms of computation, as shown in Table III. That means even if ResNet adopts a higher bit-width and higher accuracy scheme for training, it still has higher energy efficiency. In contrast, the model structure of GoogleNet shows higher adaptability in low-bit training scenarios, which brings inspiration to future neural network architecture design.

3) Results using longer training: We also experiment with another training setting with 200 epochs on ImageNet. The initial learning rate is set as 0.1 and decayed following a cosine annealing schedule [41]. The training results in this setting are shown in Table IV. We can see that these results are better or comparable to those of the default experimental setting in Table II. The low-bit accuracy is very close to the full precision accuracy, which indicates that our method's effectiveness does not rely on using a specific training setting.

TABLE V
ACCURACY (%) OF TRAINING RESNET-20 ON CIFAR-10. "DIV." MEANS THAT THE TRAINING FAILED TO CONVERGE. "NONE" MEANS THAT GROUP-WISE SCALING IS NOT USED (#GROUP=1).

#group	M_g	E_x	M_x=4	M_x=3	M_x=2	M_x=1
1	None	0	90.02	85.68	Div.	Div.
c	0	0	91.54	88.35	82.29	Div.
n	0	0	91.78	89.62	80.71	Div.
nc	0	0	92.14	91.64	88.97	76.98
nc	1	0	92.37	91.73	90.39	82.61
1	None	0	90.02	85.68	Div.	Div.
1	None	1	91.67	90.11	84.72	70.4
1	None	2	92.32	92.34	91.58	90.32
nc	1	0	92.37	91.73	90.39	82.61
nc	1	1	92.52	92.16	91.48	89.97
nc	1	2	92.37	92.65	92.05	91.97

TABLE VI
ACCURACY (%) OF TRAINING RESNET-20 ON CIFAR-10 WHEN S_t IN DIFFERENT FORMAT.

M_g	E_x	M_x	S_t in $\langle 8, 23\rangle$	S_t in $\langle 8, 8\rangle$	S_t in $\langle 8, 4\rangle$
1	1	2	91.48	91.16	86.36
1	2	2	92.05	92.15	87.22

C. Ablation Studies

1) Tensor-wise Scaling: The default tensor-wise scaling is in $\langle 8, 23\rangle$ format. In fact, we find that S_t can be further quantized. Table VI shows the results of training ResNet-20 on CIFAR-10 using different S_t formats. It can be seen that S_t in $\langle 8, 8\rangle$ format can still maintain a good training accuracy. However, it is worth noting that quantizing S_t to $\langle 8, 8\rangle$ format has little influence on the overall energy efficiency. We will give a discussion in Sec. VII.

2) Group-wise Scaling: Group-wise scaling is beneficial as the data ranges vary across different groups. We compare the average relative quantization errors (AREs) of using the three grouping dimensions (Sec. IV-A) with $\langle 8, 1\rangle$ group-wise scaling format and $\langle 0, 3\rangle$ element format. The first row of Fig. 7 shows that the AREs are smaller when each tensor is split into $N \times C$ groups. Furthermore, we compare these grouping dimensions in the training process. The first section of Table V shows that the training accuracy is higher when tensors are split into $N \times C$ groups. This indicates that the reduction of AREs is essential for the accuracy of low-bit training. Moreover, we can see that using $M_g = 1$ is important for retaining accuracy, especially with low M_x (e.g., when M_x=1, 76.98% V.S. 82.61%). We also add an ablation study on the "Error" format. Weight and Activation are quantized with group-wise scaling factors solely consisting of exponent. When Error is also quantized as this, the accuracy drops to 90.7%. After introducing our $\langle 8, 1\rangle$ group-wise scaling for Error, the accuracy is 91.9%. This indicates that our MLS format is better than the shared-exponent format [20] for accurate low-bit training.

3) Element-wise Exponent: To study the influence of the element-wise exponent, we compare the AREs of quantization with different E_x without group-wise scaling, and the results are shown in the second row of Fig. 7. Intuitively, using more exponent bits results in larger dynamic ranges and smaller AREs. Furthermore, with larger E_x, the AREs of different layers are closer. Besides the ARE evaluation, Table V shows that a larger E_x achieves a better accuracy, especially when M_x is extremely small. As shown in Fig. 7 Row 3 and Table V, the ARE and accuracy are further improved when jointly using the group-wise scaling and the element-wise exponent. We can see that the group-wise scaling is essential for simplifying the floating-point accumulator to a integer one. One can use a smaller E_x with group-wise scaling (#group=nc, M_g=1, E_x=0, Acc.=92.37%) to get a comparable accuracy to a configuration with larger E_x=2 without group-wise scaling (Acc.=92.32%).

TABLE VII
ACCURACY (%) OF TRAINING RESNET-20 ON CIFAR-10, USING STOCHASTIC ROUNDING WHEN QUANTIZING DIFFERENT TENSORS.

M_g	E_x	M_x	Stochastic Rouding	Accuracy
1	1	2	None	88.48
1	1	2	Weight	88.24
1	1	2	Activation	87.91
1	1	2	Error	91.48

4) Stochastic Rounding: We also conduct experiments to study the influence of stochastic rounding. We employ stochastic rounding in the quantization of different tensors when training ResNet-20 on CIFAR-10. The results in Table VII show that it is necessary to use stochastic rounding to quantize the Error tensors. Otherwise, the training accuracy would drop sharply. In contrast, it is not necessary to use stochastic rounding to quantize Weight and Activation tensors.

D. Comparison of Energy Consumption of Conv Units

Fig. 1 shows a typical convolution hardware architecture, which consists of three main components: local multiplication (MUL), local accumulation (LocalACC), and addition tree (TreeAdd). Our framework mainly improves the local multiplications and accumulations. Compared with the full precision design, our design simplifies the floating-point multiplication (FP MUL) to use a bit-width less than 8 and the local floating-point (FP ACC) to use 16-bit or 32-bit integer. To evaluate the energy consumption, we implement the RTL design of four types of Conv units: Full Precision, FP8 [14], Int8 [12], and our MLS format. For a fair comparison, these hardware implementations have the same overall architecture and parallelism and only differ in the arithmetic and data path bit-widths. Each Conv unit consists of 16 Conv PEs with the structure shown in Fig. 1. The parallelism of the input channel and the output channel of each PE are both set to 8. These 16 Conv PEs are equivalent, and during the execution of three different types of convolutions, they can be allocated flexibly either as sample parallelism or pixel parallelism. The total MAC parallelism of a Conv unit is 1024, the amount of input data is $16 \times 8 + 8 \times 8 = 128 + 64 = 192$, and the amount of results is $16 \times 8 = 128$.

Table VIII shows the hardware power results given by Design Compiler synthesis with TSMC 65nm process and 1GHz

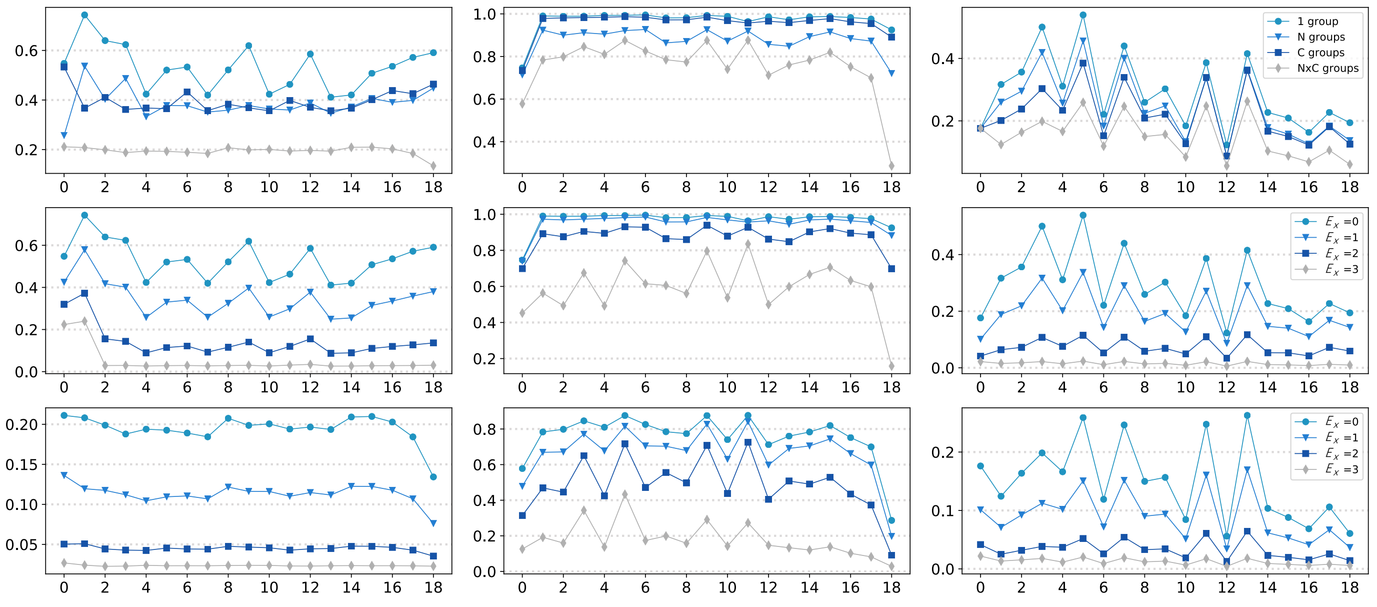

Fig. 7. Average relative quantization errors (AREs) of weight, error, activation (left, middle, right) in each layer when training a ResNet-20 on CIFAR-10. X axis: Layer index. Row 1: Different grouping dimensions ($\langle 0,3\rangle$ formatted $\bar{\boldsymbol{X}}$, $\langle 8,1\rangle$ formatted $\boldsymbol{S_g}$); Row 2: Different E_x ($\langle E_x,3\rangle$ formatted $\bar{\boldsymbol{X}}$, no group-wise scaling); Row 3: Different E_x ($\langle E_x,3\rangle$ formatted $\bar{\boldsymbol{X}}$, $\langle 8,1\rangle$ formatted $\boldsymbol{S_g}$, $N\times C$ groups).

clock frequency. We can see that the Conv unit supporting our MLS low-bit arithmetic has an area similar to those of other low-bit arithmetic and has lower multiplication and accumulation power. Because multiplication and accumulation are executed many times in a convolution, our method has higher overall energy efficiency. Given the numbers of multiplication and addition in convolution, we can use the following formula to calculate our energy efficiency improvement ratio r over full precision training in a 3×3 Conv.

$$\begin{aligned} r &= (\#MUL\times 8.067 + \#LocalACC\times 4.284 + \#TreeAdd\times 2.494)\\ &\div(\#MUL\times 0.800 + \#LocalACC\times 0.765 + \#TreeAdd\times 3.845)\\ &\approx 6.34 \end{aligned} \tag{12}$$

where the energy consumption of the scale unit is merged into TreeAdd, and the number of executions of each module ($\#MUL, \#LocalACC, \#TreeAdd$) is calculated by the size of convolution. The energy comparison of a Conv unit is also shown in Fig. 2. Next, in Sec. VI-E, we further take the energy consumption of different types of operations into consideration and present the energy analysis details of the whole network.

E. Energy Consumption of the Whole Network

This section elaborates on the method to simulate the energy consumption of the whole network training. The training of ResNet-34 on ImageNet is taken as an example. The energy of different operations is given in Table IX.

1) Simulation Methodology: The energy consumption is given by a power simulator similar to a widely used DNN power simulator [42]. This simulator has been used for studies of network pruning, which has demonstrated its credibility. The methodology is illustrated in Fig. 8. The whole network is divided into layers, and the energy consumption of each layer is simulated as the sum of computing energy and data energy. Firstly, we synthesis RTL designs of the Conv unit and the element-wise unit to obtain the power of each computing module. The parallelism of the Conv unit is introduced in Sec. VI-D, and the parallelism of the element-wise unit is 256. The element-wise unit is in full precision and can perform element-wise addition, max and sum reduction, and other

TABLE VIII
THE POWER (W) AND AREA (mm^2) RESULTS OF DIFFERENT MODULES IN CONV UNIT WITH DIFFERENT ARITHMETIC. DESIGN COMPILER IS USED FOR RTL SYNTHESIS WITH TSMC $65nm$ PROCESS AND $1GHz$ CLOCK.

	Training Method	MUL	LocalAcc	AddTree	Scale
power	Full Precision	8.067	4.284	2.494	-
	8-bit FP [14]	0.877	3.253	2.646	-
	8-bit INT [12]	1.000	1.371	1.286	-
	Ours	0.800	0.765	2.689	1.156
area	Full Precision	15.974	9.026	8.816	-
	8-bit FP [14]	0.896	9.070	8.807	-
	8-bit INT [12]	1.307	0.867	0.900	-
	Ours	1.006	0.542	8.820	0.938

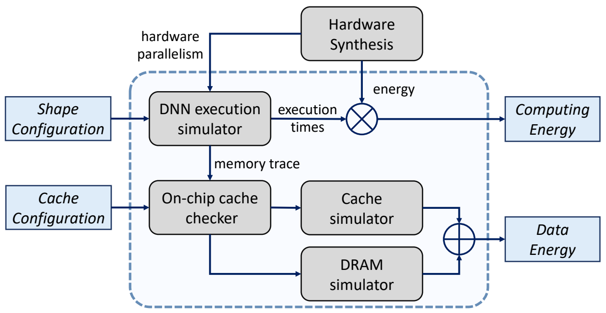

Fig. 8. The energy simulation methodology.

TABLE IX
THE COMPARISON OF THE DETAILED ENERGY COMSUMPTION OF TRAINING RESNET-34 ON IMAGENET WITH USING FULL PRECISION TRAINING AND OUR LOW-BIT TRAINING FRAMEWORK. (BATCHSIZE=16) "DQ" MEANS DYNAMIC QUANTIZATION, WHICH IS AN ADDITIONAL OPERATION IN OUR FRAMEWORK. "C ENERGY" AND "D ENERGY" ARE SHORT FOR COMPUTING ENERGY AND DATA ENERGY.

Op Name	Full Precision Training				Our Low-Bit Training			
	Type	Amount	C Energy/mJ	D Energy/mJ	Type	Amount	C Energy/mJ	D Energy/mJ
Conv	MUL	1.91E+11	1506.4	13534.8	MUL	1.91E+11	149.4	2042.8
	ADD	1.91E+11	850.5		ADD	1.91E+11	220.7	
BN	MUL	5.25E+08	3.3	303.6	MUL	5.25E+08	3.3	303.6
	ADD	5.84E+08	1.9		ADD	5.84E+08	1.9	
SGD Update	MUL	6.7E+07	0.4	39.7	MUL	6.7E+07	0.4	39.7
	ADD	4.4E+07	0.1		ADD	4.4E+07	0.1	
DQ	-	0	0	0	MUL	4.31E+8 + 8.9E+7	3.3	129.3
					ADD	2.16E+8 + 4.4E+7	0.8	
EW-Add	MUL	0	0	21.0	MUL	4.7E+07	0.3	21.1
	ADD	4.7E+07	0.2		ADD	4.7E+07	0.2	
Sum			2362.7	13899.1			380.3	2536.5

operations to support non-convolution parts in CNN training. Then, the shape configuration of a layer is processed by a **DNN execution simulator**. It conducts loop unrolling and gives the number of executions of each module and memory access trace according to the parallelism and the operation amount. Thus, with the module power and number of executions, computing energy can be obtained. As for data energy, the original memory access trace is processed by an **on-chip cache checker**. If the access interval of the same data is less than the on-chip memory size, it will be marked as cache access, and its latency and energy will be measured by SRAM simulator [43]. The on-chip memory is set to 512KB for all accelerators, and it will be used equally by the feature map and the filter during convolution. Finally, the processed trace is passed to DRAM simulator [44] to get the DRAM energy consumption.

2) Analysis of Different Operations: Considering a **convolution** with C_i input channels, C_o output channels, $K \times K$ kernel size, and $W \times H$ feature map size, the operation amounts of floating-point multiplications and additions are $C_i \times C_o \times K \times K \times W \times H$, and the operation amount in the whole network is calculated by accumulating the operation amounts of each layer in both the forward and backward processes. In our low-bit training framework, floating-point additions are only reserved in the adder tree, and the amount is $C_i \times C_o \times W \times H$. The other additions are integer accumulation. The **group-wise scaling factors** introduce additional scaling operations. Fortunately, when using the $\langle E_g, 0\rangle$ or $\langle E_g, 1\rangle$ format, we can efficiently implement group-wise scaling with shifting (Eq. 4). In this way, the energy consumption of a scaling operation is comparable to an integer addition operation.

For **batch normalization**, fully connected layer, and SGD update, the operation amount and energy consumption are **the same for both the full precision and our low-bit training framework.** Specifically, nine multiplications and ten additions are performed on each element of a $C \times W \times H$ feature map in the forward and backward processes for batch normalization. The forward process of batch normalization is

$$\begin{aligned} \mu &= \frac{1}{M}\sum_{i=1}^{M} x_i; \sigma^2 = \frac{1}{M}\sum_{i=1}^{M} x_i^2 - \mu^2 \\ y_i &= \frac{x_i - \mu}{\sqrt{\sigma^2 + 0.00005}}; z_i = \gamma y_i + \beta. \end{aligned} \tag{13}$$

We can see that in the forward process of BN, for each input element, one addition is required to calculate the batch mean, and one multiplication and one addition are used to calculate the batch variance, and two multiplications and two additions are used for normalization and affine transformation. The backward process of batch normalization is

$$\begin{aligned} \frac{\partial L}{\partial \gamma} &= \sum_{i=1}^{M} \frac{\partial L}{\partial z_i} \cdot y_i; \frac{\partial L}{\partial \beta} = \sum_{i=1}^{M} \frac{\partial L}{\partial z_i}; \frac{\partial L}{\partial y_i} = \frac{\partial L}{\partial z_i} \cdot \gamma; \\ t_1 &= \sum_{j=1}^{M} \frac{\partial L}{\partial y_j}; t_2 = \sum_{j=1}^{M} (\frac{\partial L}{\partial y_j} \cdot y_j) \\ \frac{\partial L}{\partial x_i} &= \frac{M\frac{\partial L}{\partial y_i} - t_1 - y_i \cdot t_2}{M \cdot \sqrt{\sigma^2 + 0.00005}}. \end{aligned} \tag{14}$$

There are six multiplications and six additions performed on one element in the backward of BN ("1M1A, 1A, 1M, 1A, 1M1A, 3M2A" for each formula in Eq. 14). As shown in Table I, the number of MACs in BN is orders of magnitude smaller than that in Conv. Hence, the energy consumption of BN is relatively smaller compared with convolution.

Our method introduces the overhead of **dynamic quantization**. We consider that four multiplications and two additions are needed for one element in dynamic quantization. One addition is to calculate the max (Alg. 2 Line 2), and the other is to calculate the sum of $\boldsymbol{Frac_{xs}}$ and $\boldsymbol{R}$ (Alg. 2 Line 11), and four multiplications are used in Alg. 2 Line 2 and Line 7. Note that other multiplications and divisions in Alg. 2 describe the simulation of the dynamic quantization process, and do not correspond to actual computations on hardware. The number of elements is $C \times W \times H$ for Activation and Error and $C_i \times C_o \times K \times K$ for Weight, and their energy consumptions

TABLE X
THE LATENCY (ms) OF TRAINING WITH DIFFERENT DATA FORMATS.

Training Method	ResNet-34	VGG-16	GoogleNet
Full Precision	1185.4	5051.2	619.6
FP8 [14]	988.0	3628.8	481.4
Ours	1034.0	3770.0	529.7

are shown in Table IX. Stochastic rounding is used for the quantization of Error, and we consider the overhead of reading the random tensor $\boldsymbol{R}$ in generating memory access trace.

For **element-wise addition** of two MLS tensors z_1, z_2, we need to multiply the ratio of their tensor-wise scales $S_t^{z_1}/S_t^{z_2}$ to Z_2, and then the element-wise addition can be conducted. Therefore, extra multiplications of the same amount are needed in our low-bit training framework.

3) Summary of the energy simulation experiment: Since convolution accounts for the main computational cost of CNN training (Table I), the overhead introduced by our framework is low compared with the reduced cost. The last row of Table IX shows the sum of the energy consumption of previous operations. The results show that our low-bit training framework achieves $(2362.7 + 13899.1) \div (280.3 + 2536.5) = 5.6\times$ higher energy efficiency than full precision training. The energy consumption of other networks and 8-bit floating-point training can be conducted similarly to the above analysis and is not described here. Considering all the overheads, our low-bit training framework could achieve $4.9 \sim 10.2\times$ higher energy efficiency than the full precision framework, when training different models on ImageNet. And compared with previous low-bit floating-point training frameworks (FP8) [14], our computing efficiency is $2.2\times$ higher due to the simplified integer accumulator, and the overall energy efficiency could be $1.2\times$ higher considering data energy.

4) Latency Evaluation: The number of executions given by the DNN simulator is the number of cycles. We uniformly set the clock frequency to 1GHz, and obtain the computing latency according to the number of computing cycles and clock frequency. And the latency of data access is given by the DRAM simulator. Finally, we obtain the overall latency of the forward process as follows:

$$Latency = Latency_{computing} + Latency_{data} \tag{15}$$

The latency evaluation results are shown in Table X, we can see that our training framework achieves at least $1.15\times$ speedup compared with full precision training.

VII. Discussion

A. Potential of Inference with MLS Data Format

We disable the training overhead in the simulator (described in Sec. VI-E) to only consider the forward process. The BNs are calculated in the evaluation mode, and the weights are quantized into the MLS format and fixed. The batch size is set to 16, and the results are shown in Table XI. It is worth noting that although these results demonstrate the potential of the MLS format to be used by PTQ for low-bit inference, the low-bit training framework proposed in this paper is decoupled from the PTQ methods introduced in Sec. II. This is because although our method conducts computations in the MLS low-bit format during the training process, the obtained weight is still in full precision (as shown in Fig. 4). Thus, any PTQ algorithms with other low-bit data format can be used.

TABLE XI
THE ENERGY CONSUMPTION (mJ) AND LATENCY (ms) OF FORWARD PROPAGATION WITH DIFFERENT DATA FORMATS.

Training Method	Model	Energy	Latency
Full Precision	ResNet-34	3508.6	326.6
	VGG-16	26373.7	1446.9
	GoogleNet	2007.9	150.6
FP8 [14]	ResNet-34	945.9	290.8
	VGG-16	3292.4	1078.6
	GoogleNet	646.6	130.3
Ours	ResNet-34	745.1	295.3
	VGG-16	2539.6	1094.0
	GoogleNet	573.8	136.4

B. Quantization of S_t

Our method keeps S_t in the floating-point format. Retaining floating-point in the operations involving S_t does not have a significant impact on the overall energy efficiency, since MACs in convolutions account for the majority of the operations in a convolution layer, as shown in Table I. Actually, quantizing S_t could introduce undesirable overheads. Firstly, quantizing S_t will introduce more operations into the DQ procedure. Furthermore, using a quantized S_t might require a specially designed element-wise computing unit to achieve energy-efficient multiplication, since we need to multiply some full-precision feature maps with S_t (after Conv layers followed by element-wise addition). And this unit cannot be reused by other operations with different data formats, increasing the area overhead. Therefore, it is better to keep S_t in the floating-point format, so that BN, DQ, and other operations can use the same floating-point element-wise unit.

VIII. Conclusion

This paper proposes a CNN low-bit training framework for higher energy efficiency while retaining accuracy. We design a multi-level scaling tensor format containing tensor-wise scaling, group-wise scaling, and element-wise scaling. Furthermore we describe the dynamic quantization procedure and low-bit convolution arithmetic and analyze why our data format and hardware design improve energy efficiency. Instead of using traditional systolic array hardware architecture, we adopt an adder tree architecture hardware to support our MLS data format. Experimental results and the energy consumption simulation demonstrate the effectiveness of our framework. Compared with previous low-bit integer training frameworks, our framework can retain a higher accuracy for various models, including ResNet, VGG, and GoogleNet. Our framework can achieve much higher energy efficiency than previous low-bit floating-point training frameworks.

Acknowledgment

This work was supported by National Natural Science Foundation of China (No. U19B2019, 61832007, 61621091); Beijing National Research Center for Information Science and Technology (BNRist); Beijing Innovation Center for Future Chips; Beijing Academy of Artificial Intelligence. And we thank Huawei Technologies for the support and discussion.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

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

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[6] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, 2014.

[7] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

[8] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "Pact: Parameterized clipping activation for quantized neural networks," *ArXiv*, vol. abs/1805.06085, 2018.

[9] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," in *NeurIPS*, 2018.

[10] Z. Dong, Z. Yao, A. Gholami, M. Mahoney, and K. Keutzer, "Hawq: Hessian aware quantization of neural networks with mixed-precision," *ArXiv*, vol. abs/1905.03696, 2019.

[11] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," *ArXiv*, vol. abs/1802.04680, 2018.

[12] Y. Yang, S. Wu, L. Deng, T. Yan, Y. Xie, and G. Li, "Training high-performance and large-scale deep neural networks with full 8-bit integers," *Neural networks : the official journal of the International Neural Network Society*, vol. 125, pp. 70–82, 2020.

[13] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *NeurIPS*, 2018.

[14] X. Sun, J. Choi, C.-Y. Chen, N. Wang, S. Venkataramani, V. Srinivasan, X. Cui, W. Zhang, and K. Gopalakrishnan, "Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks," in *NeurIPS*, 2019.

[15] L. Cambier, A. Bhiwandiwalla, T. Gong, M. Nekuii, O. H. Elibol, and H. Tang, "Shifted and squeezed 8-bit floating point format for low-precision training of deep neural networks," *ArXiv*, vol. abs/2001.05674, 2020.

[16] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015.

[17] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," *ArXiv*, vol. abs/1511.06393, 2015.

[18] E. Park, J. Ahn, and S. Yoo, "Weighted-entropy-based quantization for deep neural networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7197–7205, 2017.

[19] P. W. Jacob *et al.*, "gemmlowp: a small self-contained low-precision gemm library.(2017)," [OL], 2017, https://github.com/google/gemmlowp Accessed February 1, 2021.

[20] B. Darvish Rouhani, D. Lo, R. Zhao, M. Liu, J. Fowers, K. Ovtcharov, A. Vinogradsky, S. Massengill, L. Yang, R. Bittner *et al.*, "Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[21] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*, 2016.

[22] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[23] Z. Liu, Z. Shen, M. Savvides, and K. Cheng, "Reactnet: Towards precise binary neural network with generalized activation functions," *ArXiv*, vol. abs/2003.03488, 2020.

[24] H. Qin, R. Gong, X. Liu, Z. Wei, F. Yu, and J. Song, "Ir-net: Forward and backward information retention for highly accurate binary neural networks," *ArXiv*, vol. abs/1909.10788, 2019.

[25] P. Gysel, J. J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, pp. 5784–5789, 2018.

[26] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *ArXiv*, vol. abs/1606.06160, 2016.

[27] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," in *NeurIPS*, 2018.

[28] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *ArXiv*, vol. abs/1603.04467, 2016.

[29] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Hall, L. Hornof, A. Khosrowshahi, C. Kloss, R. J. Pai, and N. Rao, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *NIPS*, 2017.

[30] D. Das, N. Mellempudi, D. Mudigere, D. Kalamkar, S. Avancha, K. Banerjee, S. Sridharan, K. Vaidyanathan, B. Kaul, E. Georganas, A. Heinecke, P. Dubey, J. Corbal, N. Shustrov, R. Dubtsov, E. Fomenko, and V. O. Pirogov, "Mixed precision training of convolutional neural networks using integer operations," *ArXiv*, vol. abs/1802.00930, 2018.

[31] X. Sun, N. Wang, C.-Y. Chen, J. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. El Maghraoui, V. V. Srinivasan, and K. Gopalakrishnan, "Ultra-low precision 4-bit training of deep neural networks," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[32] N. Jouppi, C. Young, N. Patil, D. A. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. luc Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. A. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. Yoon, "In-datacenter performance analysis of a tensor processing unit," *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12, 2017.

[33] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded fpga platform for convolutional neural network," *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016.

[34] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, pp. 35–47, 2018.

[35] D. G. Hough, "The ieee standard 754: One for the history books," *Computer*, vol. 52, no. 12, pp. 109–112, 2019.

[36] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *ICML*, 2015.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions,"

2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9, 2015.

[39] A. Krizhevsky, "Convolutional deep belief networks on cifar-10," 2010.

[40] N. Mellempudi, S. Srinivasan, D. Das, and B. Kaul, "Mixed precision training with 8-bit floating point," *ArXiv*, vol. abs/1905.12334, 2019.

[41] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv: Learning*, 2017.

[42] T.-J. Yang, Y. hsin Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6071–6079, 2017.

[43] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," 2009.

[44] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer Architecture Letters*, vol. 15, pp. 45–49, 2016.

Kai Zhong received his B.S. degree in electronic engineering from Tsinghua University, Beijing, in 2019. He is currently pursuing his Ph.D. degree at the Department of Electronic Engineering, Tsinghua University, Beijing. His research mainly focuses on deep learning acceleration and hardware-friendly algorithm optimization.

Xuefei Ning received the B.S. and Ph.D. degrees in electronic engineering from Tsinghua University, in 2016 and 2021. Xuefei's research mainly focuses on efficient deep learning algorithm design and neural architecture search.

Guohao Dai (S'18) is currently a postdoctoral researcher in the Department of Electronic Engineering, Tsinghua University, Beijing, China. He has received his B.S. degree in 2014 and Ph.D. degree (with honor) in 2019 from Tsinghua University, Beijing. Guohao's research mainly focuses on large-scale sparse graph computing, heterogeneous hardware computing, emerging hardware architecture, and etc. He has received Best Paper Award in ASP-DAC 2019, and Best Paper Nomination in DATE 2018. He is the winner of the NeurIPS Billion-Scale Approximate Nearest Neighbor Search Challenge in 2021, the recipient of the Outstanding Ph.D. Dissertation Award of Tsinghua University in 2019. Currently, he serves as PI/Co-PI for several projects with a personal share of over RMB 6 million.

Zhenhua Zhu received his B.S. degree in electronic engineering department of Tsinghua University, Beijing, China, in 2018. He is currently pursing his Ph.D degree in electronic engineering department of Tsinghua University. His research mainly focuses on memristor, computer architecture, and Processing-In-Memory.

Tianchen Zhao received his B.S. degree in electronic engineering from Beihang University in 2020. He is currently pursuing his Ms. degree at EE Department, Beihang University. His research interest mainly focuses on efficient deep learning algorithm design.

Shulin Zeng received his B.S. degree in electronic engineering department of Tsinghua University, Beijing, China, in 2018. He is currently pursing his Ph.D. degree in electronic engineering department of Tsinghua University. His research mainly focuses on software-hardware co-design for deep learning and virtualization in the cloud.

Yu Wang (S'05-M'07-SM'14-F'22) received the B.S. and Ph.D. (with honor) degrees from Tsinghua University, Beijing, in 2002 and 2007. He is currently a tenured professor with the Department of Electronic Engineering, Tsinghua University. His research interests include brain inspired computing, application specific hardware computing, parallel circuit analysis, and power/reliability aware system design methodology. He has authored and coauthored more than 200 papers in refereed journals and conferences. He has received Best Paper Award in ASPDAC 2019, FPGA 2017, NVMSA 2017, ISVLSI 2012, and Best Poster Award in HEART 2012 with 9 Best Paper Nominations (DATE18, DAC17, ASPDAC16, ASPDAC14, ASPDAC12, 2 in ASPDAC10, ISLPED09, CODES09). He is a recipient of DAC under 40 innovator award (2018), IBM X10 Faculty Award (2010). He served as TPC chair for ICFPT 2019 and 2011, ISVLSI2018, finance chair of ISLPED 2012-2016, track chair for DATE 2017-2019 and GLSVLSI 2018, and served as program committee member for leading conferences in these areas, including top EDA conferences such as DAC, DATE, ICCAD, ASP-DAC, and top FPGA conferences such as FPGA and FPT. Currently, he serves as co-editor-in-chief of the ACM SIGDA E-Newsletter, associate editor of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,the IEEE Transactions on Circuits and Systems for Video Technology, the Journal of Circuits, Systems, and Computers,and Special Issue editor of the Microelectronics Journal. He is now with ACM Distinguished Speaker Program.

Huazhong Yang (M'97-SM'00-F'20) received B.S. degree in microelectronics in 1989, M.S. and Ph.D. degree in electronic engineering in 1993 and 1998, respectively, all from Tsinghua University, Beijing. In 1993, he joined the Department of Electronic Engineering, Tsinghua University, Beijing, where he has been a Professor since 1998. Prof. Yang was awarded the Distinguished Young Researcher by NSFC in 2000, Cheung Kong Scholar by the Chinese Ministry of Education (CME) in 2012, science and technology award first prize by China Highway and Transportation Society in 2016, and technological invention award first prize by CME in 2019. He has been in charge of several projects, including projects sponsored by the national science and technology major project, 863 program, NSFC, and several international research projects. Prof. Yang has authored and co-authored over 500 technical papers, 7 books, and over 180 granted Chinese patents. His current research interests include wireless sensor networks, data converters, energy-harvesting circuits, nonvolatile processors, and brain inspired computing. He has also served as the chair of Northern China ACM SIGDA Chapter science 2014, general co-chair of ASPDAC'20, navigating committee member of AsianHOST'18, and TPC member for ASP-DAC'05, APCCAS'06, ICCCAS'07, ASQED'09, and ICGCS'10.