A Cost-Efficient Approximate Dynamic Ranged Multiplication and Approximation-Aware Training on Convolutional Neural Networks

HYUNJIN KIM1, (Member, IEEE), and ALBERTO A. DEL BARRIO2, (Senior Member, IEEE)

1School of Electronics and Electrical Engineering, Dankook University, Youngin, Gyeonggi, 16890 KR (e-mail: hyunjin2.kim@gmail.com)
2Department of Computer Architecture and System Engineering, Complutense University of Madrid, Madrid, ES (e-mail: abarriog@ucm.es)

Corresponding author: HyunJin Kim (e-mail: hyunjin2.kim@gmail.com).

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2021R1F1A1048054). The EDA tool was supported by the IC Design Education Center (IDEC), Korea. This research was results of a study on the “HPC Support” Project, supported by the ‘Ministry of Science and ICT’ and NIPA. Furthermore, the paper has been supported by the CM under grant S2018/TCS-4423, the EU (FEDER) and the Spanish MINECO under grant RTI2018-093684-B-I00 and by Fundacion BBVA under grant PR2003_2001.

ABSTRACT This paper proposes a low-cost approximate dynamic ranged multiplier and describes its use during the training process on convolutional neural networks (CNNs). It has been noted that the approximate multiplier can be used in the convolution of CNN’s forward path. However, in CNN inference on a post-training quantization with a pre-trained model, erroneous convolution output from highly approximate multipliers significantly degrades performance. On the other hand, with the CNN model based on an approximate multiplier, the approximation-aware training process can optimize its learnable parameters, producing better classification results considering the approximate hardware. We analyze the error distribution of the approximate dynamic ranged multiplication and characterize it in order to find the most suitable approximate multiplier design. Considering the effects of normalizing the biased convolution outputs, a low standard deviation of relative errors with respect to the multiplication outputs leads to negligible accuracy drop. Based on these facts, the hardware costs of the proposed multiplier can be further reduced by adopting the partial products’ inaccurate compression, truncated input fraction, and reduced-width multiplication output. When the proposed approximate multiplier is applied to the residual convolutional neural networks for the CIFAR-100 and Tiny-ImageNet datasets, the accuracy drops of the approximation-aware training results are negligible compared to those using 32-bit floating-point CNNs.

INDEX TERMS Approximate computing, approximate multiplier, approximation-aware training, convolutional neural network, probabilistic multiplier.

I. INTRODUCTION

In the implementation of machine learning, deep neural networks are used in various fields. Notably, CNNs have attracted the attention of many researchers because they outperform the human classification ability in image recognition. Even though there have been significant advances in machine-based image classification, tremendous power consumption is necessary to perform the convolutions of CNNs. The conventional CNN training is based on the floating-point data format and operations, producing real-valued trained models. In post-training quantizations, after obtaining the real-valued trained models, low-cost CNNs are implemented based on the quantized data format and approximate hardware without fine-tuning or retraining steps [1]. Thanks to the error-resilience of CNNs, schemes with small error can provide the appropriate inference output. However, when the approximation increases, the performance is degraded because the approximate output can be significantly different from the pre-trained models’ optimized value.

A CNN contains a bundle of layers with learnable parameters. The training process automatically finds the optimal parameters for a model and its hardware implementation with
iterative steps. The quantization-aware training adopts the quantized data format and its operations in the forward path during training [1]. Similarly, when using approximate hardware units in the simulation of the forward path, the learnable model is optimized for achieving better classification results, which is denoted as the approximation-aware training. The approximate hardware in the forward pass produces errors as part of the training loss during training, which can simulate the erroneous behavior of the approximation. After training the approximation-aware model, a CNN inference can utilize the trained model that considers the quantized data format and approximate hardware.

CNNs using high precision quantized data formats such as the 32-bit fixed-point [2], 16-bit fixed-point [3], bfloat16 floating-point [4], and 8-bit floating-point [5] formats achieve equivalent classification performance compared with the model using 32-bit floating-point format. Even though the implementation of these CNNs can reduce hardware costs, these are not affordable yet for the low-power systems. CNNs using the 8-bit fixed-point format [6] binarized CNN [7], [8], and log-based CNN [9], [10] provide highly quantized models for low-cost CNN implementation. However, these models show significant accuracy drops in CNN inference.

Notably, for the fixed-point format, the dynamic ranged multiplier obtains its exponent and uses the significant bits starting at the position of the leading one [11]. Compared to the exact fixed-point multiplier, the dynamic ranged multiplier can significantly reduce hardware costs, maintaining the original dynamic range in the fixed-point format. Depending on the method of how to approximate the significand multiplication, the error distribution of the approximate dynamic ranged multiplier is determined. The approximate logarithmic multiplier replaced the significand multiplication by adding input fractions [12]–[17]. It was proved that the approximate logarithmic multiplier could achieve reasonable performances in CNN inference [14]–[17] with pre-trained models. Several existing works focused on the unbiased design of the approximate dynamic ranged multiplier and its digital signal processing (DSP) applications [11], [18]. However, there were no in-depth studies to adopt the approximate dynamic ranged multiplier when performing approximation-aware training on CNNs.

In this paper, we propose an approximate dynamic ranged multiplier for performing convolutions on CNNs and introduce an approximation-aware training method by simulating the approximation in the training stage. The proposed design has the following characteristics:

1) This design performs 16-bit signed multiplication with a fixed scaling factor, and 32-bit multiplication outputs are accumulated in the convolution.
2) In the significand multiplication, our round-off scheme truncates the input fractions, so that only two \( k \)-bit input significands are multiplied. Then, the output width of the \( k \)-bit significand multiplication is reduced to an \( l \)-bit output, being \( l < 2k \). To further reduce hardware costs, inaccurate compressors are used in the partial product reduction of the significand multiplication.
3) The convolution output using the truncation, reduced-width output, and inaccurate partial product reduction schemes can be biased. Because the multiplier’s average relative error is far from zero, the convolution output could also have a biased error.

When training a CNN with our approximate dynamic ranged multiplier, the biased average relative error of the approximate multiplier is not critical in the performance of trained models. Even though the proposed multiplication output is biased, the normalization can offset the biased convolution output, and the following learnable scaling layer adjusts the normalized output to minimize the training loss. Instead of the biased average relative error, the dispersion of the relative error distribution seems to be important in CNN training. This paper describes the CNN basic block structure to adopt the proposed multiplier. Compared to the regular 16-bit signed fixed-width multiplier, power consumption decreases by 81.9%. For the residual neural networks [19] on both the CIFAR-100 [20] and Tiny-ImageNet [21] datasets, our training results did not show accuracy drop with respect to the 32-bit floating-point based CNN. This paper is structured as follows: firstly, the preliminaries including the dynamic ranged multiplier and its approximation schemes will be reviewed. Then, the error-resilience of the CNN training with the approximate multiplier and its error analysis will be explained. The benefits of the proposed approximate multiplier will be analyzed in terms of hardware costs and power consumption. Finally, the experimental results in the CNN training will be shown.

II. PRELIMINARIES

The fixed-point format can be preferred for its hardware simplicity in CNN implementations [2], [3]. The fixed-point multiplication consists of the integer multiplication output and its scaling. When not mentioning other formats, integer multiplications with the fixed scaling factor are assumed in this paper. However, this does not mean that the proposed design is not limited to the operation with fixed-point data. The difference with the standardized or custom floating-point format is that the log-linear representation is provided in the floating-point format. The proposed design can approximate the significand multiplication output in the floating-point multiplication.

A. DYNAMIC RANGED MULTIPLIER

Whereas the exact multiplier provides accurate multiplication outputs on a given format, the approximate multiplier sacrifices its precision to reduce hardware costs and increase operating speed. The arithmetic-based approaches adopt logic blocks, making inaccurate output based on their arithmetic formulas. A signed integer \( A_{\text{sign}} \) can be represented into log-


Algorithm 1 Unsigned Dynamic Ranged Multiplier

1: procedure UDRM($A, B, n, k$)
2: \[ \exp_A \leftarrow LOD(A, n) \quad \triangleright \text{Leading one detection of } A \]
3: \[ \text{if } \exp_A > k - 1 \text{ then} \]
4: \[ A_k \leftarrow ROUND(A, \exp_A, k) \]
5: \[ \text{shift}_A \leftarrow \exp_A - (k - 1) \]
6: \[ \text{else} \]
7: \[ A_k \leftarrow A[k - 1 : 0] \]
8: \[ \text{shift}_A \leftarrow 0 \]
9: \[ \text{end if} \]
10: \[ \exp_B \leftarrow LOD(B, n) \quad \triangleright \text{Leading one detection of } B \]
11: \[ \text{if } \exp_B > k - 1 \text{ then} \]
12: \[ B_k \leftarrow ROUND(B, \exp_B, k) \]
13: \[ \text{shift}_B \leftarrow \exp_B - (k - 1) \]
14: \[ \text{else} \]
15: \[ B_k \leftarrow B[k - 1 : 0] \]
16: \[ \text{shift}_B \leftarrow 0 \]
17: \[ \text{end if} \]
18: \[ \text{sig}_{AB} \leftarrow A_k \times B_k \]
19: \[ \text{mul}_{AB} \leftarrow \text{sig}_{AB} \ll (\text{shift}_A + \text{shift}_B) \]
20: return \text{mul}_{AB}
21: end procedure

linear form as:

\[ \text{sig}_A = (-1)^{s_j}(1 + x_A)^2^{\exp_A}, \exp_A \in \mathbb{N}, s \in \{0, 1\}, 0 \leq x_A < k \]

(1)

where \( s \) is the sign bit. The term \( \exp_A \) is the exponent calculated with the location of the leading one, and \( x_A \) is the fraction part of \( A \). In (1), \((1 + x_A)\) denotes the significand.

The dynamic ranged multiplier converts two operands into the log-linear representation. Then, the significand multiplication output is shifted by the output of the exponent addition. Algorithm 1 explains the detailed process of \((n, k)\)-bit unsigned dynamic ranged multiplier, where \( n \) is the bit-width of the inputs and \( k \) is the bit-width for representing the rounded significands. The leading one detector denoted as \( LOD(A, n) \) produces \( \exp_A \) (line 2). When \( \exp_A > k - 1 \), \( k \) bits of the rounded significand, \( k_A \), are obtained using \( ROUND \) function (line 4). In Fig. 1, it is assumed that an unsigned integer \( A \) is multiplied in an \((n, k)\)-bit dynamic ranged multiplier. In Fig. 1 (a), when \( \exp_A > k - 1 \), \( k \)-bit rounded significand, \( k_A \), is adopted in the significand multiplication, and the \((\exp_A - k + 1)\) least significant bits (LSBs) of \( A \) are discarded. The most significant \( k \) bits, starting from the leading one, are adopted for the rounded significand. In addition to this, the round-off scheme can determine the LSB of the rounded significand in a solid box. On the other hand, when \( \exp_A \leq k - 1 \), \( k_A \) contains the \( k \) LSBs of \( A \) (line 7), which is illustrated in Fig. 1 (b).

The significand multiplication output \( \text{sig}_{AB} \) is produced by multiplying \( k_A \) with \( k_B \). The terms \( \text{shift}_A \) and \( \text{shift}_B \) determine how much \( \text{sig}_{AB} \) has been shifted. When \( \exp_A > k - 1 \), \( \text{shift}_A = \exp_A - (k - 1) \) (line 5); otherwise, \( \text{shift}_A = 0 \) (line 8). The same process is performed to produce \( \exp_B \), \( k_B \), and \( \text{shift}_B \) (lines 10–17). Finally, this multiplier shifts \( \text{sig}_{AB} \) to the left by \( \text{shift}_A + \text{shift}_B \), resulting in \( \text{mul}_{AB} \) (line 19). The multiplier described in Algorithm 1 can be extended into the signed multiplier using input and output sign conversions.

B. APPROXIMATION IN DYNAMIC RANGED MULTIPLIER

When the addition with exponents is not approximated, different approximate multipliers can be developed depending on the method of how to approximate the significand multiplication. Several major approximation methods are listed as: (1) partial product ignorance [13, 18]; (2) input significand round-off [11, 15, 16, 18]; (3) reduced-width output [22, 23]. When using the exact significand multiplication, all partial products are added to make the multiplication output. In the approximated multiplication, according to its underlying technique, several partial products are ignored. The round-off scheme for input significands reduces the number of bits multiplied. It determines the approximate value of the discarded low-order bits from each input significand. The reduced-width output scheme only utilizes several high-order bits from the significand multiplication output.

In the following, \( MUL_{\text{exact}} \) and \( MUL_{\text{appr}} \) shall be the exact and approximate multiplication outputs for the same input, respectively. In order to present the error characteristic of the signed approximate multiplier, the relative error (called RERR later) is formulated as:

\[ RERR = \frac{MUL_{\text{appr}} - MUL_{\text{exact}}}{\text{abs}(MUL_{\text{exact}})}. \]  

(2)

The biased approximate multiplication is defined as:

Definition 1 (biased): When the approximate multiplication is biased, (1) the average relative error (called \( RERR_{\text{avg}} \) later) of multiplication outputs for uniformly distributed inputs cannot be zero, or (2) \( RERRs \) are not normally distributed.

To make \( RERR_{\text{avg}} \) close to zero and obtain the normal distribution of \( RERRs \), the unbiased rounding is imple-
A \times (1 + x_A)) into the linear term truncation of extra bits [24]. As presentation, additional costs are required compared with the unbiased rounding consumes one bit in the significand replication, which requires an accurate log. \[ \log_2(1 + x_A) \] for \( A \) into the linear term. Because the significand multiplication is replaced by the addition of input fractions, these multipliers ignore several partial products and reduce their output width. The truncated Mitchell multiplier [15] denoted as Mitchell-\( w \) adds \( (w - 1) \)-bit truncated fractions for the approximate significand multiplication. The main drawback of these logarithmic approaches is the antilogarithm block, which requires an \( (n + w) \)-bit shifter as well as a \( w \)-bit right shifter. Furthermore, a zero-unit detector is required in order to achieve high-accuracy values. On the other hand, unlike the Mitchell-\( w \) multipliers, the dynamic ranged multiplier with \( k \) sums all partial products together using a \( k \)-bit significand multiplication and requires a \( 2k \)-bit left shifter in the final stage.

### TABLE 1: Truth Table of 4:2 Compressor in [25].

| \( i_1 \) | \( i_2 \) | \( i_3 \) | \( i_4 \) | \( carry \) | \( sum \) | \( err \) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | -1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | -1 |
| 0 | 1 | 1 | 0 | 1 | 1 | +1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | +1 |
| 1 | 0 | 1 | 0 | 1 | 1 | +1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | -1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | -1 |

The probabilistic multiplier adopts inaccurate compressors when reducing the partial products. With the inaccurate compressors, the carry propagation with \( cout \) is not needed, which achieves the speedup and reduces hardware costs. Fig. 3 illustrates an example of the partial product reduction in a 4-bit probabilistic multiplier. In step 2, two inaccurate 4:2 compressors are adopted. In step 3, the 8-bit final output is obtained after adding the reduced products. This mixed version uses accurate compressors for the high-order output bits to keep the maximum error small, as shown in [26]. On the other hand, inaccurate compressors are adopted for the low-order output bits to produce pseudo-random outputs.

FIGURE 2: Schematics of 4:2 compressors: (a) exact compressor with 2 full adders (FA); (b) inaccurate compressor in [25].

FIGURE 3: Partial product reduction using both accurate and inaccurate compressors.
In general, a kernel consists of the \( w \) weights multiplied with the input features, where \( K_{i,j} \) denotes the kernel for the \( i \)-th input and \( j \)-th output channels. After accumulating the multiplication outputs with \( m \) input features and kernels \( K_{0,j}, K_{m-1,j} \), the output feature for the \( j \)-th output channel is obtained. Therefore, when using the \( w_{K} \times h_{K} \) kernel, the number of accumulated multiplication outputs can reach up to \( w_{K} \times h_{K} \times m \) for achieving an output feature. For example, the 9-th convolution layer in ResNet-34 [19] has \( 3 \times 3 \) kernel with 128 input channels, accumulating 1,152 multiplication outputs for an output feature in the convolution. It must be noted that the number of accumulated outputs for each output feature can be significant in the convolution.

In post-training quantization, after producing a pre-trained model with real-valued hardware, the trained model parameters are quantized [1]. CNNs perform their forward pass with the quantized values during inference. In a similar way, a low-cost CNN can adopt imprecise hardware units to approximate model parameters of the pre-trained model in the inference step. This post-training approximation does not consider the approximate hardware in the training stage.

Unlike the CNN inference using the approximate pre-trained models, the approximation-aware training process finds the optimized model under its given resource limitation such as data quantization, hardware approximation, etc. During training, the forward pass of CNNs adopts approximate hardware, producing additional training loss from erroneous operation results. This approximation-aware training simulates the erroneous behavior of approximate hardware implementations in the training stage. Therefore, the trained model obtained from approximation-aware training can consider the quantized data format and approximate hardware. Depending on model parameter initialization methods, a real-valued pre-trained model can be adopted. When fine-tuning a model, its real-valued pre-trained model is used to initialize its model parameters. Then, the model is retrained by adopting the approximate hardware in the forwarding path. On the other hand, a model can be trained from scratch without using the real-valued pre-trained model. If the model can be trained from scratch, the training process can be simplified without any need of pre-trained model parameters.

In most CNN structures, the batch normalization layer normalizes convolution outputs in a mini-batch [27]. The equation of the batch normalization is given as:

\[
\hat{x}_i = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}},
\]

where \( \mu_\beta \) and \( \sigma \) are the mini-batch mean and variance for a channel. The terms \( \mu_\beta \) and \( \sigma \) are learned with the iterative mini-batch process. A convolution output denoted as \( x_i \) is one element in the channel. The term \( \epsilon \) is used to prevent the division by zero. The batch normalization layer reduces the change of the convolution output distribution [27]. After normalizing convolution outputs, learnable affine parameters scale the normalized convolution outputs.

In several CNN inferences using pre-trained models on realistic datasets, logarithmic multiplications can achieve affordable performance [15]–[17]. When the average relative error is adjusted in the convolution output by rounding errors [17] or implementing the unbiased rounding [15], [16], it is empirically proved that the approximate multiplications can be valid in the forward path. However, as the approximation increases up to a certain level, it shows significant performance drops.

After performing convolutions, the non-linearity in the activation layer can filter unnecessary information of convolution outputs, providing the error resilience. Various error sources help to enhance the performance of optimized models in CNN training. When features in a channel are normalized, the normalization adds correlated noise to the features. Data augmentation such as random erasing [28] and dropout technique [29] can insert intentional noise for achieving better trained models. It is known that this approximate noise provides the regularization for avoiding so-called overfitting. We expect that an acceptable error from the approximate multiplication will not be critical in obtaining good training results. Additionally, CNN’s error resilience and noise-friendly CNN training lead us believe that a sophisticated approximate multiplier with an acceptable error will not degrade the performance of the trained model.

III. PROPOSED MULTIPLIER DESIGN AND ITS USAGE IN CONVOLUTION LAYER

A. DESIGN MOTIVATIONS

Multiplication occupies the most considerable portion of the computations of the convolution layer. The 32-bit floating-point real-valued model has wide ranged input operands and multiplication outputs. However, their expensive computations could be redundant considering the error-resilience and noise-awareness of CNN training. On the other hand, several previous works indicate that the narrow range of input operands effectively reduces hardware costs implementing CNN models. In [5], [9], 5-bit exponent is used to represent the input operands. In [3], 16-bit fixed-point multiplications are adopted with the stochastic rounding of each operand.
In [6], 8-bit integer multiplications are used in the convolution layer, and each convolution layer needs its optimized scaling factor to access most of the input distribution.

The unbiased rounding in [11], [15], [16] and error compensation in [17] can make $RERR_{avg}$ close to zero and produce the normal distribution of $RERR$s in the multiplication outputs, assuming that input operands are randomly distributed. However, the unbiased rounding and error compensation require additional resources to implement the rounding scheme and scale convolution outputs. Because the zero $RERR_{avg}$ could be achieved based on the assumption that input values are evenly distributed, these methods do not consider the training process.

We have a question about whether this unbiased design and error compensation for the multiplication are necessary in CNN training. Notably, when the batch normalization is used in each channel [27], the convolution outputs are normalized and learnable affine parameters are applied in the following scaling layer in the channel. Unlike the inference on the pre-trained model, the batch normalization finds the mini-batch mean and variance in CNN training. This normalizing process motivates us to put away the existing unbiassing design and error compensation in the CNN training.

Spread errors in the normalized convolution outputs also depend on the errors from approximate multiplications. Empirically, we say that a multiplication design is suitable after evaluating the design in CNN training. However, since it depends on both its adopted CNN structure and dataset, it is not possible to predict which the multiplication design will be suitable in other models or datasets. In the error analysis of the next section, we explain which approximation methods could be more suitable in CNN training. In this section, it will be shown that the proposed approximate dynamic ranged multiplier can afford approximation methods such as the partial product ignorance, truncation, and reduced-width output scheme.

### B. PROPOSED APPROXIMATE DYNAMIC RANGED MULTIPLIER

Fig. 5 describes the hardware structure of the proposed approximate multiplier. For the convolution layer using 16-bit signed multipliers, the proposed design adopts $n = 16$.

The two Sign Converters produce unsigned integer operands $A$ and $B$ from $A_{sign}$ and $B_{sign}$, where 1's complement conversion is adopted like the idea in [15]. The process for producing several terms is also explained in Algorithm 1. The location of the leading one in $n$-bit $A$ is encoded into the $[\log_2 n]$-bit $exp_A$. In the Encoder, $shift_A$ is produced, which is used in the Mux&Decoder. It is noted that $shift_A = exp_A - (k - 1)$ for $exp_A > k - 1$ (line 5 in Algorithm 1). Otherwise, $shift_A = 0$ (line 8 of Algorithm 1). The Mux&Decoder generates the $k$-bit $k_A$ signal depending on $shift_A$.

The $ROUND$ function in Algorithm 1 is implemented in the Mux&Decoder to adopt the truncation scheme only by discarding $shift_A$ LSBs from $A$ when $shift_A > 0$ (line 4 of Algorithm 1). When $shift_A = 0$, $k_A$ is produced with the $k$-bit LSBs of $A$ (line 7 of Algorithm 1). The above process is also employed in order to obtain $k_B$ and $shift_B$.

In Fig. 5, $k_A$ and $k_B$ are multiplied by a $k$-bit probabilistic multiplier. In our design, partial product reductions for the high-order bits adopt the accurate compression for limiting errors. Fig. 3 describes the probabilistic multiplier for $k = 4$. On the other hand, for the low-order bits, the inaccurate compressor can produce the probabilistic error. Additionally, the significand multiplication has the reduced-width output. In step 3 of Fig. 3, the LSBs $O_L \sim O_{2k-1}$ are discarded, so that only the most significant $l$ bits of the output are transferred to $l$-bit Left Shifter.

There are several existing inaccurate compressors to implement the probabilistic multiplier. From the survey [26], several compressors have non-zero output even though input operands are zero [30], [31]. When applying the compressors with this non-zero output effect, the multiplication with zero input operands can produce non-zero output. Even though a zero detector can be used to overcome this problem, additional hardware costs and speed degradation are not avoidable. The proposed design adopts the compressor in [25] considering its balanced error distribution and hardware cost reduction. In Table 1, because the numbers of $+1$ and $-1$ error entries are 4 and 3, if it is assumed that inputs for a compressor are evenly distributed, the probability of having a positive error is slightly higher than that of having a negative error. On the other hand, the truncated input fraction and reduced-width output always cause a negative error. We expect these two different error distributions to be somewhat neutralized.
With \( shift_A + shift_B \) from the adder, the \( l \)-bit output from the probabilistic multiplier is internally shifted to the left by \( shift_A + shift_B + (2k - l) \), producing the \( 2n \)-bit output. The final signed output is obtained by performing the output sign conversion. The exclusive-OR gate uses the MSBs of signed input operands \( A_{sign} \) and \( B_{sign} \). When the signs of input operands are different from each other, the exclusive-OR gate outputs ‘1’. It is known that the performance drop of the CNN inference from 1’s complement conversion is negligible [15]. Whereas 2’s complement conversion requires a \( 2 \)-bit adder, only \( n \)-bit inverter is needed for the 1’s complement sign conversion. Therefore, our design adopts 1’s complement sign conversion for the signed multiplication output.

Fig. 6 describes an example of 16-bit multiplication when \( k = 4 \). When \( A = 174D_{16} \) and \( B = 17A1_{16} \), \( shift_A = 9 \) and \( shift_B = 5 \) with \( exp_A = 12 \) and \( exp_B = 8 \). Because \( exp_A > k - 1 \) and \( exp_B > k - 1 \), 4-bit \( k_A = A[12 : 9] = 1011_2 \) and \( k_B = B[8 : 5] = 1011_2 \) are 4 MSBs from the leading ones of \( A \) and \( B \), respectively. The significand multiplication on the left in Fig. 6 adopts \( k_A = 1011_2 \) and \( k_B = 1011_2 \) as input operands. With \( l = 4 \), the probabilistic multiplier outputs 1000_2, which means that the significand multiplication output is approximated into 10.02. In this case, a negative error is produced compared with the exact significand multiplication. Next, this output is shifted by \( shift_A + shift_B + (2k - l) = 18 \) with \( k = l = 4 \), so that this approximate dynamic ranged multiplier outputs 32-bit 200000_16. On the other hand, the exact multiplication output for \( A \times B \) is 2267B2_{16}, so that error and \( RERR \) is \(-267B2_{16} \) and \(-7.52\% \), respectively. Even though Fig. 6 describes the multiplication with two integers, this method can be applied to the fixed-point multiplication with a fixed-point scaling factor.

**FIGURE 6:** Example of the proposed multiplication.

**C. CONVOLUTION LAYER WITH PROPOSED MULTIPLIER**

Whereas a large number of accumulations exist in the convolution, the scaling and non-linear activation layers only perform the multiplication for each element without any accumulation. The multiplication error in these layers could be critical in the classification result. Because accumulation is not required, the number of multiplications is small compared to the convolution layer. Therefore, the proposed approximate multiplier is limited to the convolution in CNN models.

The customized convolution layer adopts the proposed multiplier in the convolution. Fig. 7 describes the details of the convolution and its following layers. We target 16-bit fixed-point activations and weights. In the terms \( FL, IL > \), \( IL \) is the number of bits for representing an exponent, while \( FL \in \mathbb{N} \) denotes the scaling factor \( 2^{-FL} \) for the fixed-point operation. The same fixed-point scaling factor is applied to all customized convolution layers.

In the batch normalization, each convolution output from the accumulated multiplication outputs is normalized from (3). The mini-batch mean \( (\mu) \) and variance \( (\sigma) \) for a channel are learned based on the convolution outputs using the approximate multiplier. After the normalization, the normalized value \( \hat{x}_i \) is scaled and shifted into \( y_i \), which can be equated as:

\[
y_i = \lambda \hat{x}_i + \beta, \tag{4}
\]

where parameters \( \lambda \) and \( \beta \) are learnable in CNN training. Next, an activation layer such as ReLU [32] is located. The clip layer is mandatory to limit the maximum output value from the activation layer. When \( FL = 10 \), the clip layer’s output is smaller than \( 2^5 \) for 16-bit fixed-point operation. The clip layer’s output is used as the input of the next convolution layer.

Whereas the forward path of a CNN model adopts this customized convolution layer in both CNN training and inference, the backward path does not adopt the proposed approximate multiplier. It is known that 16-bit fixed-point operations cannot provide convergence of CNN training in [2]. In [3], the stochastic rounding is required to apply the 16-bit fixed-width multiplier to the backward path. Although
approximate operations in the backward path can be used in the implementation of a low-cost training engine, this causes a noticeable accuracy drop in inference, and also long training time is needed. Accuracy drop in inference and long training time are needed. Instead, we aim for the trained model to have high inference accuracy when using an approximate hardware. Our approximation-aware training adopts 32-bit floating-point operations in the backward path. Therefore, real-valued weights are updated in the backward path and then approximated into 16-bit fixed-point values in the forward path during training. This paper focuses on the multiplication of the customized convolution layer in Fig. 7, so that the approximation of other hardware elements is not considered in the rest of this paper.

IV. ERROR AND HARDWARE COST ANALYSIS

A. ERROR ANALYSIS

This section will introduce how to configure suitable multipliers, following the structure shown in Fig. 5 in terms of error distribution. We cannot determine which approximate multiplier can be the best only with the final classification result on a specific dataset and CNN structure. Firstly, many different factors affect the final classification result. In addition to the multiplication method, the final classification result depends on other hyperparameters and CNN structure. Secondly, the software-based emulation of the multiplier hardware in CNN training requires very long evaluation time and tremendous computation resources. Therefore, several design points of the approximate multiplier should be considered from our error analysis.

The corner case analysis of RERR is as follows: The proposed design adopts 1’s complement and conversion for input operands and multiplication output. The minimum relative error, RERR_{min}, can be -100% for a specific case. When A_{sign} = -1 and B_{sign} = -1, A and B are all zeros after 1’s complement sign conversion, so that approximate A_{sign} × B_{sign} is also zero. Therefore, RERR can reach up to -100%. However, the number of these specific cases with large RERR is negligible. Additionally, its scaled error is only $-2^{-2F_L}$ in the fixed-point multiplication.

The maximum relative error RERR_{max} happens when one of the input operands is zero and the other is negative. For example, when A_{sign} = 0 and B_{sign} = -1, the proposed design approximates A × B into 0. Afterwards, the approximate multiplication output becomes -1 after 1’s complement sign conversion. Because exact A_{sign} × B_{sign} is zero, RERR can be infinite according to (2). However, its scaled error is $-2^{-2F_L}$, which is negligible.

We explain the effect of multiplier’s RERR distribution in the back normalization, so that (3) is rewritten as:

$$
\hat{x}_i \rightarrow \frac{(1 + e)x_{i\text{exact}} - (1 + \bar{c}_m)\mu_{\beta, \text{exact}}}{\sqrt{\frac{1}{m} \sum_{m} ((1 + e)x_{i\text{exact}} - (1 + \bar{c}_m)\mu_{\beta, \text{exact}})^2 + \epsilon}}.
$$

(5)

The terms $x_{i\text{exact}}$ and $x_i$ are the exact convolution output and erroneous convolution output using the proposed design in (3). The term $e$ denotes the relative error of the convolution output using an approximate multiplier, so $x_i = (1 + e)x_{i\text{exact}}$. The terms $\bar{c}_m$ and $\mu_{\beta, \text{exact}}$ are the mini-batch means of the relative errors and the exact convolution outputs, respectively. The mini-batch variance using these terms is shown in the denominator of (5).

If $e$ and $\bar{c}_m$ are close to zero, $\hat{x}_i \simeq x_{i\text{exact}}$. This condition can be met when the approximate multiplier has small RERRs and RERR_{avg} $\simeq$ 0 with well balanced RERR distribution. For example, DRUM-k can provide this case with the unbiased rounding. On the other hand, when $e$ and $\bar{c}_m$ are far from zero, if $e$ has a low standard deviation, $e$ is clustered around $\bar{c}_m$. Based on the approximation of $(1 + e) \approx (1 + \bar{c}_m)$, the terms can be cancelled in the numerator and denominator in (5), so $\hat{x}_i \simeq x_{i\text{exact}}$. When multiplications outputs have non-zero RERR_{avg}, the convolution output could produce non-zero $e$ and $\bar{c}_m$. Even though multiplication outputs have non-zero RERR_{avg}, if the outputs can produce a low standard deviation of RERRs, the normalized multiplication output can keep track of the normalized value using the exact multiplier.

In the dynamic ranged multiplier, whereas the approximate significand multiplication always multiplies positive significands, the sign of the final multiplication output depends on its sign conversion. Except for the specific corner cases, output signs of the exact and our approximate multipliers are always the same. Thus, it has a RERR distribution mirror-symmetric with respect to zero, producing a bimodal distribution.

Fig. 8 illustrates RERR distributions of the proposed design with $k = l = 4$ for every 5% interval. One million pairs of two 16-bit signed inputs were randomly selected in simulations. In [12], [13], [17], the error analysis is performed based on the absolute RERRs because it is guaranteed that RERR $\leq 0$ for MUL_{exact} $\geq 0$ and it has mirrored distribution with respect to RERR = 0. Because of the inaccurate compressor, the proposed multiplier can have the cases with RERR $> 0$ for MUL_{exact} $\geq 0$. We adopt a transformed RERR distribution by shifting the RERR distribution to make its unimodal distribution with respect to zero. The average absolute relative error, abs(RERR)_{avg}, is used to obtain the shifting value denoted as shift_{val}, which

![FIGURE 8: RERR distribution of proposed multiplication of k = l = 4: (a) RERR distribution; (b) adjusted RERR distribution with shift_{val}.](image-url)
is calculated as:

\[ shift_{val} = \frac{1}{1 - \text{abs}(\text{RERR})_{avg}} \]  

(6)

The multiplication outputs in Fig. 8 (a) are multiplied by the \( shift_{val} \) from (6). This design has 13.79% \( \text{abs}(\text{RERR})_{avg} \), so that the adjusted \( \text{RERR} \) distribution is obtained by multiplying \( \text{RERRs} \) with \( \frac{1}{1 - \text{abs}(\text{RERR})_{avg}} \) \( \approx 1.16 \), which produces an adjusted \( \text{RERR} \) distribution in Fig. 8 (b).

In our experiments, there was no significant accuracy drop when \( k = 4 \), so that Table 2 lists the standard deviation and the percentage of \( \text{RERR} \) on the specific conditions. The \textit{normalized mean error distance} denoted as \textit{NMED} in Table 2 is defined as:

\textbf{Definition 2 (NMED):} The NMED is the average of \( \frac{|M_{\text{upper}} - M_{\text{exact}}|}{\text{MaxOut}} \), where \( \text{MaxOut} = 2^{2n-2} \) for the \( n \)-bit signed multiplication.

Depending on the approximation methods mentioned in the previous section, DesignI, DesignII, DesignIII, and DesignIV have been analyzed. Notably, DesignIV adopted all methods of the proposed approximate dynamic ranged multiplier. We define that \( P(\text{cond}) \) is the percentage of \( \text{RERRs} \) on a specific condition. Except for DRUM-I and DesignII, the standard deviation and \( \text{NMED} \) were calculated from the transformed \( \text{RERR} \) distributions after applying the \( shift_{val} \). Considering \( P(\text{cond}) \) in Table 2, all \( \text{RERR} \) distributions were well balanced. Because DesignI adopted exact 4-bit significant multiplication, its standard deviation was the smallest. Compared to DesignIII, DesignIV slightly degraded \( \text{RERR} \) distribution due to the inaccurate compressor. Interestingly, designs using the unbiased rounding, DRUM-I and DesignII, had large standard deviations and \( \text{NMEDs} \). Compared to other designs, the unbiased rounding was not helpful for reducing the standard deviations and \( \text{NMEDs} \). When decreasing \( k \) or \( l \), the standard deviation and \( \text{NMED} \) increased, which spread \( \text{RERRs} \) widely. For example, when \( k = 4 \) and \( l = 3 \) in DesignIV, the standard deviation and \( \text{NMED} \) increased up to 10.570 and 2.64E-2 with \( P(\text{RERR} > 5\%) = 32.95\% \) and \( P(\text{RERR} < -5\%) = 33.16\% \). In Mitchell-3, the

### TABLE 2: Comparison with Error Distributions.

| design | NMED | stddev | \( \text{RERR} > 5\% \) | \( \text{RERR} < -5\% \) | \( \text{RERR} > 10\% \) | \( \text{RERR} < -10\% \) | \( \text{RERR} > 20\% \) | \( \text{RERR} < -20\% \) |
|--------|------|--------|----------------|----------------|----------------|----------------|----------------|----------------|
| DRUM-I | 1.41E-2 | 7.308 | 25.45\% | 25.49\% | 8.93\% | 8.94\% | 0.16\% | 0.16\% |
| Mitchell-I | 1.57E-2 | 7.848 | 27.33\% | 27.39\% | 10.81\% | 10.85\% | 0.25\% | 0.25\% |
| Mitchell-II | 9.72E-3 | 4.753 | 15.69\% | 15.73\% | 1.36\% | 1.36\% | 0.01\% | 0.00\% |
| DesignII | 7.47E-3 | 3.867 | 10.12\% | 10.13\% | 0.20\% | 0.19\% | 0.01\% | 0.01\% |
| DesignIII | 1.66E-2 | 8.782 | 28.59\% | 28.62\% | 12.16\% | 12.21\% | 1.27\% | 1.27\% |
| DesignIV | 1.16E-2 | 5.852 | 20.57\% | 20.67\% | 4.01\% | 4.03\% | 0.01\% | 0.01\% |
| DesignV | 1.28E-2 | 6.473 | 22.65\% | 22.79\% | 5.98\% | 6.00\% | 0.04\% | 0.04\% |

1 All designs multiply 16-bit signed input operands.
2 With exact \( k \)-bit significant multiplication when \( k = 4 \) and \( l = 2k = 8 \).
3 With \( k \)-bit significands multiplication adopting the inaccurate compressors and unbiased rounding when \( k = 4 \) and \( l = 5 \).
4 With exact \( k \)-bit significant multiplication when \( k = l = 4 \).
5 With proposed \( k \)-bit significant multiplication adopting the inaccurate compressors when \( k = l = 4 \).

![FIGURE 9: Standard deviations of RERR distributions.](image-url)

B. HARDWARE COST ANALYSIS

Several multipliers were implemented to evaluate costs in terms of circuit area and power consumption. We coded our designs as combinational multipliers using Verilog hardware description language (HDL), where internal hardware blocks were described using the dataflow modeling. A 16-bit exact fixed-point multiplier was described using the Verilog multiply operator to produce the intrinsic multiplier in the logic synthesis based on a target library. The descriptions of
TABLE 3: Comparison in terms of hardware costs.

| design | stddev | delay (ns) | circuit area (um²) | power (uW) |
|--------|--------|------------|--------------------|------------|
| Fixed  | -      | 4.0        | 2708.1             | 198.971    |
| DRUM-4 | 7.308  | 4.57       | 1269.7             | 53.444     |
| Mitchell-4 | 4.753  | 4.4        | 1315.3             | 53.996     |
| DesignII | 5.852  | 4.58       | 1213.1             | 38.308     |
| DesignIV | 6.473  | 4.4        | 1188.2             | 35.912     |

1 Exact k-bit significand multiplication when $k = l = 4$.
2 k-bit significand multiplication adopting the inaccurate compressors when $k = l = 4$.

DRUM- $k$ and Mitchell- $w$ were obtained from [33] and [34]. The error analysis indicates that the reduced-width significand multiplication output can be useful, and the unbiased rounding in the dynamic ranged multiplier was redundant for CNN training. Therefore, DesignI and DesignII were not considered in the hardware cost analysis.

These codes were synthesized using Synopsys Design Compiler on 32nm generic standard cell library from Synopsys. The adopted cells had the characteristic on the slow process (SS), 0.75 V power source, and 125°C temperature. The cells were used to find the critical path delay and circuit area reports. The target critical path delay was varied at 0.2 ns steps between 4 and 5 ns. Table 3 summarizes the comparison in terms of hardware costs and power consumptions. The intrinsic 16-bit exact fixed-point multiplier can be produced in 4.0 ns using the target library cells. It required more than twice the circuit area compared to other approximate multipliers in Table 3. Mitchell-4 and DesignIV can be synthesized at 4.4 ns with the zero slack. However, DRUM- $k$ and DesignIII showed the negative slacks, which indicate that the long critical paths were required in the significand multiplication. The inaccurate compressor and reduced-width significand multiplication output reduced the circuit area for implementing DesignIV compared to other implementation results. Mitchell- $w$ requires the antilogarithm module, which consists of a left and a right shift registers as well as several multiplexers. Furthermore, a zero detector unit is necessary in Mitchell- $w$ [17]. Therefore, even though the significand multiplication was approximated by adding the fractions, Mitchell-4 required additional circuit area.

Based on the possible 4.4 ns target delay, power reports were obtained from the library cells for estimating power consumptions. The adopted cells can produce power reports that characterized the typical process (TT), 1.05 V power source, and -40°C temperature. As shown in Table 3, the low power consumptions in DesignIII and DesignIV indicate that the reduced-width significand multiplication output was useful in reducing power consumption. Additionally, the inaccurate compressor was helpful for achieving lower power consumptions than other designs.

V. EXPERIMENTS WITH CNN TRAINING

This section explains the residual neural network’s structure and experimental environments. In many existing works, CNN models with highly stacked convolution layers and large-scaled datasets such as ImageNet [35] have been evaluated in CNN training. However, the emulation of an approximate multiplier consists of a bundle of operations, which requires tremendous computation resources. Instead, we adopted an affordable model (ResNet-18 [19]) and datasets (CIFAR-100 [20] and Tiny-ImageNet [21]) in CNN training.

A. TRAINING FRAMEWORKS

The Caffe deep learning framework [36] was adopted in our experiments. We coded functions in CUDA C++ [37] to emulate the internal hardware blocks used in the approximate dynamic ranged multiplier. The multiply operator (‘*’) was replaced by a CUDA C++ function. In the customized convolution layer, the general matrix multiplication (GEMM) called the function to emulate the proposed approximate multiplier in the forward path. For the backward path and weight updating, 32-bit floating-point GEMMs were used for the approximation-aware training. We can parameterize each convolution layer along with reusing the existing Caffe setup and prototxt files. The scaling for 16-bit fixed-point data was fixed as $10^{-10}$ in all customized convolution layers. The customized function can accept 16-bit signed input operands and generate 32-bit multiplication outputs.

B. MODEL STRUCTURE

Among existing CNN models, we adopted a residual neural network [19]. Reasons why experiments were done using ResNet are as follows: firstly, the skip connection or shortcut in the ResNet has been used in many prominent CNN models [38]–[41], providing fast training speed and reducing the impact of vanishing gradients. Secondly, more accurate CNN models can be easily constructed using a pyramid structure.

Fig. 10 describes the pyramid structure using eight basic blocks for the CIFAR dataset [20]. The $3 \times 3$ convolution layer is denoted as $3 \times 3$ Conv, where the number in the box means the number of output channels from the layer. The dotted box indicates a basic block containing two subblocks, where the shortcut skips two subblocks. The dotted arrow means the shortcut consisting of $1 \times 1$ convolution for the downsampling, where the number of output channels is twice that of input channels. Both $3 \times 3$ and $1 \times 1$ convolution layers adopted the approximate multipliers. Fig. 11 describes the
The CIFAR dataset contains 60,000 colour images, where 50,000 and 10,000 images are used in CNN training and test, respectively [20]. The CIFAR-100 dataset has 100 classes, providing 600 images to each class. Compared to 10 classes of the CIFAR-10 dataset, more sophisticated classification is needed on the CIFAR-100 dataset.

The experimental setup was as follows: For the data augmentation in CNN training, 40 \times 40 pixel colour images were generated by padding four zero-valued pixels on the edge of each image. Each image was randomly cropped into 32 \times 32 pixel colour images for the training process. The cropped images can be horizontally mirrored at random. Then, 128 was used as the mean of 8-bit pixel values, which was subtracted from each pixel value. In the accuracy test, the random mirroring was not performed, and the original size of each test image was maintained without the random crop. Finally, the data layer scaled each pixel values by \frac{1}{255}. The training was performed during 120 epochs (200 iterations per an epoch) with batch size=250, where the learning rate started at 0.1 and was decayed by multiplying 0.1 at (40, 70, 100) epochs. In other words, the learning rate was decayed at (8,000, 14,000, 20,000) iterations. This training adopted the Nesterov optimizer [42] with momentum=0.9 and weight decay=5e^{-4}.

Fig. 12 illustrates the classification result and training loss during the training for the CIFAR-100 dataset. As explained in the legend of Table 2, DesignIII \((k = l = 4)\) and DesignIV \((k = l = 4)\) were evaluated. The terms Log, FP32, and Fixed16 indicate the training results using 16-bit log, 32-bit floating-point, and 16-bit fixed-point formats and their multiplication. We define the Log format and its multiplication in these experiments as follows: When using the Log multiplication, the input fractions of two unsigned integers \(A\) and \(B\) were set as zeros, so that \(A \times B \approx 2^{\exp(A)} \times 2^{\exp(B)}\). Fig. 12 shows that the training using the Log multiplication failed to achieve suitable convergence. In the early iterations for the FP32 and Fixed16 CNNs, the increase in the test accuracy and decrease in the training loss were slightly steeper. However, the final accuracies of the designs above converged into about 74.5% in Fig. 12, which means that the proposed multiplier can provide an acceptable error distribution with \(k = l = 4\).

Fig. 13 illustrates Top-1 accuracies of the proposed DesignIV \((k = 4)\), which were compared with those using Mitchell-w, FP32, and Fixed16 CNNs. The final Top-1 accuracies shown in Fig. 13 were obtained by averaging five runs. DesignIV \((k = l = 4)\) achieved 74.66% Top-1 accuracy, which had negligible accuracy drops compared to the Top-1 accuracies of FP32 (74.8%) and Fixed16 (74.95%) CNNs. On the other hand, when \(l < 4\) and \(w < 4\), DesignIV \((k = 4)\) and Mitchell-w degraded classification accuracies in Fig. 13. Based on Figs. 12 and 13, we concluded that the proposed approximate multiplier with \(k = l = 4\) can achieve the acceptable RERR distribution and Top-1 accuracy in the CNN training.
The validation dataset can be used in the accuracy test. All images consist of 36,253 images and a validation dataset with 10,000 images. Tiny-ImageNet dataset [21] contains a training dataset with 100,000 images and a validation dataset. During about 36,000 iterations (about 90 epochs) with batch size=256. We adopted the policy, so that the learning rate \( lr \) was decayed by \( \text{base}_{lr} \times (1 - \frac{\text{iteration}}{\text{epoch}}) \). The term \( \text{base}_{lr} \) means the starting learning rate, which was initialized as 0.05. This training adopted the default SGD optimizer [44] for the Tiny-ImageNet dataset. Like Fig. 12, Fig. 14 shows the classification result and training loss for the Tiny-ImageNet dataset. In the error analysis, Log’s standard deviation of \( RERR \) distribution was 21.644 after applying \( \text{shift}_{\text{val}} \) in (6). Therefore, we can assure that the error of the Log format and its multiplier can exceed the threshold for achieving relevant training results.

Interestingly, the approximate dynamic ranged multipliers showed no accuracy drop compared to the result of FP32 CNN. In Fig. 14, the final Top-1 test accuracies with FP32, Mitchell-4, DesignIII \( (k = l = 4) \), and DesignIV \( (k = l = 4) \) CNNs were 59.59\%, 59.76\%, 59.77\%, and 59.72\%, respectively. The increase in the test accuracy of the FP32 multiplication was slightly steeper in the early iterations. However, the final Top-1 test accuracies of CNNs using the designs above were very close in Fig. 14. This evaluation result shows that the error distributions in Fig. 14 did not significantly degrade the performance of a trained ResNet-18 model with the Tiny-ImageNet dataset.

We evaluated the Top-1 accuracy drops by varying \( l \) when \( k = 4 \). The final Top-1 test accuracies for Mitchell-2, DesignIII \( (k = 4, l = 2) \), and DesignIV \( (k = 4, l = 2) \) were 59.32\%, 60.13\%, and 59.89\%, respectively. Therefore, we concluded that these approximate multipliers could provide the acceptable error distribution in this CNN training.

VI. CONCLUSION

This paper proposes an approximate dynamic ranged multiplier based on error and hardware cost analyses and its application to CNNs. Besides, we introduce the approximation-aware training using the proposed hardware design. The proposed approximate multiplier is evaluated by being adopted in the ResNet model. When a multiplier has a low standard deviation of \( RERR \) distribution, the normalized output can keep track of the normalized value using the exact multiplier. Based on these facts, we can determine which approximate design can have a good \( RERR \) distribution and achieve low-power consumption. In this paper, the detail of the fixed-point model structure and its application is described. Hardware costs can be further reduced using the partial products’ inaccurate compression, truncated input fraction, and reduced-width output. Furthermore, the proposed design integrates the 1’s complement approach to approximate sign handling. With the ResNet model on target datasets, the training results using the proposed approximate multiplier show negligible accuracy drop. Considering both the reduced hardware costs and the acceptable classification results, it is concluded that the proposed approximate dynamic ranged multiplier could be useful for providing high-performance, low-cost systems that perform neural network training and inference.

REFERENCES

[1] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv preprint arXiv:1806.08342, 2018.
[2] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Lijiang He, Jia Wang, Ling Li, Tianshi Chen, Zhwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In 2014 47th Annual IEEE/ACM
International Symposium on Microarchitecture, pages 609–622. IEEE, 2014.

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

[4] Shibo Wang and Pankaj Kanwar. Fload16: the secret to high performance on cloud tpus. Google Cloud Blog, 2019.

[5] 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, 2016.

[6] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalcheniuch. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2704–2713, 2018.

[7] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagery classification using binary convolutional neural networks. In European conference on computer vision, pages 525–542, Springer, 2016.

[8] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In Proceedings of the European conference on computer vision (ECCV), pages 722–737, 2018.

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

[10] Edward H Lee, Daisuke Miyashita, Elaina Chai, Boris Murmann, and S Simon Wong. Lognet: Energy-efficient neural networks using logarithmic computation. In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 5900–5904, IEEE, 2017.

[11] Soheil Hashemi, R Bahar, and Sherief Reda. Drum: A dynamic range unbiased multiplier for approximate applications. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pages 418–425. IEEE Press, 2015.

[12] John N Mitchell. Computer multiplication and division using binary logarithms. IRE Transactions on Electronic Computers, (4):512–517, 1962.

[13] Zdenka Babic, Aleksej Avramovic, and Patricio Bulic. An iterative logarithmic multiplier. Microprocessors and Microsystems, 35(1):23–33, 2011.

[14] Min Soo Kim, Alberto A Del Barrio, Roman Hermida, and Nader Bagherzadeh. Low-power implementation of Mitchell’s approximate logarithmic multiplication for convolutional neural networks. In Design Automation Conference (ASP-DAC), 2018 23rd Asia and South Pacific, pages 617–622. IEEE, 2018.

[15] Min Soo Kim, Alberto Antonio Del Barrio Garcia, Leonardo Tavares Oliveira, Roman Hermida, and Nader Bagherzadeh. Efficient mitchell’s approximate log multipliers for convolutional neural networks. IEEE Transactions on Computers, 2018.

[16] HyunJin Kim, Min Soo Kim, Alberto A Del Barrio, and Nader Bagherzadeh. A cost-efficient iterative truncated logarithmic multiplication for convolutional neural networks. In 2019 IEEE 26th Symposium on Computer Arithmetic (ARITH), pages 108–111. IEEE, 2019.

[17] Min Soo Kim, Alberto Antonio Del Barrio Garcia, Hyunjin Kim, and Nader Bagherzadeh. The effects of approximate multiplication on convolutional neural networks. IEEE Transactions on Emerging Topics in Computer, 2021.

[18] Shaghayegh Vahdat, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. Tosam: An energy-efficient truncation-and-rounding-based scalable approximate multiplier. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 27(5):1161–1173, 2019.

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

[20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[21] Leon Yao and John Miller. Tiny imagenet classification with convolutional neural networks. CS 231N, 2(5):8, 2015.

[22] Kyoung-Ju Cho, Kwang-Chul Lee, Jin-Gyun Chung, and Keshab K Parhi. Design of low-error fixed-width modified booth multiplier. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 12(5):522–531, 2004.

[23] Saroja S Bhusare and VS Kanchana Bhaaskaran. Fixed-width multiplier with simple compensation bias. Procedia Materials Science, 10:395–402, 2015.

[24] Israel Koren. Computer arithmetic algorithms. CRC Press, 2018.

[25] Mohammad Ahmadinejad, Mohammad Hossein Moayeri, and Farnaz Sabtezadeh. Energy and area efficient imprecise compressors for approximate multiplication at nanoscale. AEU-International Journal of Electronics and Communications, 110:152859, 2019.

[26] Antonio Giuseppe Maria Strollo, Ettore Napoli, Davide De Caro, Nicola Petra, and Gennaro Di Meo. Comparison and extension of approximate 4-2 compressors for low-power approximate multipliers. IEEE Transactions on Circuits and Systems I: Regular Papers, 2020.

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

[28] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In AAAI, pages 13001–13008, 2020.

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1):1929–1958, 2014.

[30] Amir Memomi, Jie Han, Paolo Montuschi, and Fabrizio Lombardi. Design and analysis of approximate compressors for multiplication. IEEE Transactions on Computers, 64(4):984–994, 2014.

[31] Farnaz Sabtezadeh, Mohammad Hossein Moayeri, and Mohammad Ahmadinejad. A majority-based imprecise multiplier for ultra-efficient approximate image multiplication. IEEE Transactions on Circuits and Systems I: Regular Papers, 66(11):4200–4208, 2019.

[32] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In ICML, 2010.

[33] Drum. https://github.com/scale-lab/DRUM, 2020. [Online; accessed 07-Dec-2020].

[34] log-arithmetic. https://github.com/albertodbg/log-arithmetic, 2020. [Online; accessed 07-Dec-2020].

[35] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV), 115(3):211–252, 2015.

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

[37] Cuda c++ programming guide. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, 2019. [Online; accessed 11-Nov-2020].

[38] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv preprint arXiv:1602.07261, 2016.

[39] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalcheniuch, WeiJun Wang, Tobias Weyand, Marco Andreaetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

[40] Xiaoyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 6848–6856, 2018.

[41] Gao Huang, Zhuang Liu, Geoff Pleiss, Laurens Van Der Maaten, and Kilian Weinberger. Convolutional networks with dense connectivity. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2019.

[42] Yuriy Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/k^2). In Doklady an ussr, volume 269, pages 453–457, 1983.

[43] Lei Sun. Resnet on tiny imagenet. Submitted on, 14, 2016.

[44] Zhe Cui and Andrew Murphy. Speech recognition with simple compensation bias. Procedia Materials Science, 10:395–402, 2015.
HYUNJIN KIM is the Associate professor in the School of Electronics and Electrical Engineering at the Dankook University, Republic of Korea. He received the Ph.D in Electrical and Electronics Engineering (2010), Master (1999), and Bachelor (1997) degrees in Electrical Engineering from Yonsei University, Republic of Korea. He worked as the Mixed-Signal VLSI Circuit Designer at Samsung Electromechanics (2002.02~2005.01). Besides, He is an Senior Engineer in the Field of Flash Memory Controller Project at the Memory Division of Samsung Electronics (2010.04~2011.08). His current research interests reside in the realm of the Approximate & Stochastic Computing for Neural Network Implementation Methodology, String Matching Engine, and Energy-Aware Embedded System.

ALBERTO A. DEL BARRIO received the Ph.D. degree in Computer Science from the Complutense University of Madrid (UCM), Madrid, Spain, in 2011. Since 2020, he is an Associate Professor of Computer Science with the Department of Computer Architecture and System Engineering, UCM. His research interests include Design Automation, Arithmetic as well as Video Coding Optimizations. Since 2019 he is an IEEE Senior Member and since December 2020 he is an ACM Senior Member, too.