Received: 25 September 2020 | Revised: 3 January 2021 | Accepted: 25 February 2021

DOI: 10.4218/etrij.2020-0370

ORIGINAL ARTICLE

ETRI Journal WILEY

A low-cost compensated approximate multiplier for Bfloat16 data processing on convolutional neural network inference

HyunJin Kim

School of EEE, Dankook University, Yongin, Rep. of Korea

Correspondence
HyunJin Kim, School of EEE, Dankook University, Youngin, Rep. of Korea.
Email: hyunjin2.kim@gmail.com

Funding information
This research was supported by the research fund of Dankook University in 2018.

This paper presents a low-cost two-stage approximate multiplier for bfloat16 (brain floating-point) data processing. For cost-efficient approximate multiplication, the first stage implements Mitchell's algorithm that performs the approximate multiplication using only two adders. The second stage adopts the exact multiplication to compensate for the error from the first stage by multiplying error terms and adding its truncated result to the final output. In our design, the low-cost multiplications in both stages can reduce hardware costs significantly and provide low relative errors by compensating for the error from the first stage. We apply our approximate multiplier to the convolutional neural network (CNN) inferences, which shows small accuracy drops with well-known pre-trained models for the ImageNet database. Therefore, our design allows low-cost CNN inference systems with high test accuracy.

KEYWORDS

Approximate computing, bfloat16 format, convolutional neural network, logarithmic multiplier, Mitchell's algorithm

1 | INTRODUCTION

The floating-point format is used to represent wide-ranged fractional numbers. Current floating-point standards occupy 32 bits (single precision) or 64 bits (double precision). Although these standards can represent any fractional number precisely, 32 bit or 64 bit data requirements are a great burden in several error-resilient applications. Low-precision applications prefer 16 bit floating-point formats to overcome the storage overhead and tremendous computation resource of standard floating-point data processing. Notably, the 16 bit bfloat16 format was proposed to maintain the dynamic range representation with an 8 bit exponent. Compared with the 32 bit floating-point standard (FP32), the fraction is represented only using 7 bits.

In digital signal processing, multiplication is one of the basic operations. The trade-off between cost and accuracy in multiplication reduces hardware costs by sacrificing multiplication accuracy. Hardware-based approximate multipliers convert their multiplication into other data processing operations. The lookup table-based approximate multiplier produces approximate output based on the values in the memory block, [1,2] so that there are large memory requirements. In n-bit integer multiplication, the approximate multiplier with n-bit fixed-width output discards n low-order output bits [3–7]. The arithmetic-based multiplier adopts low-cost arithmetic operations to approximate exact multiplication with affordable accuracy degradation [1,2,8–22].

The logarithmic multiplier can suppress the worst-case relative error ($rerr_{worst}$) under its designed limitation. By using the log-linear representation, it is suitable for multiplication with wide-ranging values. Notably, Mitchell's algorithm [1] converts two fixed-point numbers into floating-point or log-linear

representation and then approximates its multiplication by shifting the added fractions. Compared with the exact fixed-point multiplier, Mitchell's algorithm shows 3.8% average relative error ($rerr_{avg}$) and 11.11% $rerr_{worst}$ [1]. The approximate multiplications in [11,21,22] truncate fractions to decrease the hardware costs of multiplying fractions, and $rerr_{avg}$ is close to zero due to the unbiased fractions. However, $rerr_{worst}$ increases depending on the amount of truncated information.

Iterative approximate multipliers have been shown to achieve low relative error [1,8,22,23]. A stage transfers error terms into the next stage that compensates for the error from the first stage. For example, when Mitchell's algorithm is applied to n stages, $(11.11\%)^n$ $rerr_{worst}$ can be achieved. For the fixed-point multiplication, leading-one detectors (LODs) and their encoders are needed to obtain log-linear representation in each stage, which increases hardware costs significantly.

Moreover, the floating-point format represents a number using the normalized significand $(1 + x_A,\ 0 \le x_A < 1)$. When multiplying two floating-point numbers, exponents and significands of two numbers are added and multiplied, respectively. Therefore, Mitchell's algorithm does not require LODs to obtain log-linear representation. However, it is implemented at low costs, and $rerr_{worst}$ is still unchanged as 11.11%. If Mitchell's algorithm is applied iteratively, all stages except for the first stage require LODs and encoders, so that the iterative multiplication cannot make any benefits in terms of hardware costs. We have the motivation that different types of approximate multipliers are chained in the second stage to compensate for the error from the first stage. Because $rerr_{worst}$ from the first stage is 11.11%, we assume that the truncated fractions in the second stage do not degrade the error compensation significantly.

This paper presents a cost-efficient two-stage approximate multiplication for processing bfloat16 data. Mitchell's algorithm in the first stage is implemented using only two adders. The fraction parts are transferred into the second stage as error terms. The second stage adopts low-cost exact multiplication to avoid the use of LODs and their encoders. In the second stage, several low-order bits of error terms are discarded to reduce hardware overhead. Because the number of bits in the fraction part is fixed, discarded low-order bits make a low impact on the relative error. Then, our design multiplies the error terms from the first stage and adds the multiplication result to the final output. Compared with existing approximate multipliers, low-cost multiplications without LODs in both stages can reduce hardware overhead significantly and achieve low relative errors.

2 | PRELIMINARIES

2.1 | Bfloat 16 format

The bfloat16 format is a floating-point number representation that consists of 16 bits. Compared with FP32 format, the low-order bits of the fraction part are discarded while keeping the 8 bit exponent. A bfloat16 data operation is simplified by flushing the subnormal inputs and outputs and limiting rounding modes [24–26]. This format is not yet standardized, but it is preferable in neural network acceleration because of its half of the memory requirements and the advantage of adoptability in both training and inferencing [26].

In the floating-point multiplication, exponents of two numbers are added, and then bias 127_{10} ($7F_h$) is subtracted to retrieve the output exponent. Since the number of bit in the fractional part reduces to 7 bits, 8 bit significands are multiplied and normalized to get the normalized output significand. When normalizing the output significand, a shifter aligns the fraction, and then the output exponent increases by one. For example, when significands $1.100\ 000\ 0_2$ and $1.000\ 000_2$ are multiplied, the unnormalized output significand is $10.0100, \ldots, 0_2$, which is normalized into $1.001\ 00, \ldots, 0_2$. When any exponent of two bfloat16 formatted numbers are 00_h, both the output exponent and fraction are flushed to 00, …, 0, indicating that the multiplication output is 0. If any exponent of two bfloat16 numbers is FF_h for representing the infinite number, the output exponent is FF_h. When calculating the output exponent, if its result is out of the range of the normal bfloat16 number, the output exponent becomes 00_h or FF_h. A two-input exclusive-OR gate can be used to calculate the output sign.

2.2 | Mitchell's algorithm

In the log-linear representation, Mitchell's algorithm approximates multiplication using adders and shifters without any multiplier [1]. A positive fractional number A is expressed as

$$A = 2^{k_A} \cdot \left(1 + x_A\right),\ k_A \in \mathbb{Z}, 0 \le x_A < 1. \tag{1}$$

In (1), where k_A and x_A denote the exponent and fraction of A, respectively. The multiplication of two positive numbers A and B, $A{\cdot}B$, is given by

$$A \cdot B = 2^{k_A} \cdot \left(1 + x_A\right) \cdot 2^{k_B} \cdot \left(1 + x_B\right) = 2^{k_A + k_B} \cdot \left(1 + x_A\right) \cdot \left(1 + x_B\right). \tag{2}$$

When applying a logarithmic conversion on both sizes of (2),

$$\log_2 (A \cdot B) = k_A + k_B + \log_2 \left(\left(1 + x_A\right) \cdot \left(1 + x_B\right)\right). \tag{3}$$

When $C = A{\cdot}B$ and $C = 2^{k_C}\left(1 + x_C\right), k_C \in \mathbb{Z}, 0 \le x_C < 1$, k_C and x_C are obtained as follows:

$$\begin{cases} k_C = k_A + k_B + 1, x_C = x_A + x_B - 1, x_A + x_B \ge 1 \\ k_C = k_A + k_B, x_C = x_A + x_B, x_A + x_B < 1. \end{cases} \tag{4}$$

Considering (4), the carry-out of $x_A + x_B$ is added to $k_A + k_B$. Given any binary fractional numbers, two adders are needed to calculate the exponent and fraction of C. Unlike general multipliers, when any input value is zero, the output cannot be zero in Mitchell's algorithm. Therefore, zero detectors [27] should be implemented additionally. When handling bfloat16 numbers, the implementation of Mitchell's algorithm should check whether k_C is out of range and outputs can be zero, infinite floating numbers, or neither.

2.3 | Error calculation and unbiasing

When the error of an approximate multiplication $\mathrm{MUL}_{\mathrm{appr}}$ from the exact multiplication $\mathrm{MUL}_{\mathrm{exact}}$ is defined as $\mathrm{err} = \left|\mathrm{MUL}_{\mathrm{exact}}\right| - \left|\mathrm{MUL}_{\mathrm{appr}}\right|$, the relative error (denoted as rerr) is expressed as follows:

$$\mathrm{rerr} = \frac{\left|\mathrm{MUL}_{\mathrm{exact}}\right| - \left|\mathrm{MUL}_{\mathrm{appr}}\right|}{|\mathrm{MUL}|_{\mathrm{exact}}}. \tag{5}$$

In the log-linear representation, a fraction resolution depends on the assigned bits. Particularly, the unbiasing technique for the truncated fractions in Mitchell's algorithm is shown in [21,22] where the adder of n-bit fractions x_A and x_B receives a carry-in value of $2^{-\mathrm{n}}$. In the unbiased n-bit Mitchell's algorithm, rerr is expressed as follows:

$$\begin{cases} \dfrac{(1-x_A)\cdot(1-x_B) - 2^{-n+1}}{(1+x_A)\cdot(1+x_B)}, x_A + x_B + 2^{-n} \geq 1 \\ \dfrac{x_A \cdot x_B - 2^{-n}}{(1+x_A)\cdot(1+x_B)}, x_A + x_B + 2^{-n} < 1 \end{cases}. \tag{6}$$

In (6), if 2^{-n} and 2^{-n+1} are elmininated, (6) represents rerr of the original Mitchell's algorithm [1]. Therefore, $\mathrm{rerr}_{\mathrm{worst}} = \left(\left|\mathrm{rerr}_{\mathrm{max}}\right| > \left|\mathrm{rerr}_{\mathrm{min}}\right| \,?\, \mathrm{rerr}_{\mathrm{max}} : \mathrm{rerr}_{\mathrm{min}}\right)$. For bfloat16 data processing with $n = 7$, the original Mitchell's algorithm has $\mathrm{rerr}_{\mathrm{worst}} = 11.11\%$ when $x_A = x_B = 0.5$ and $\mathrm{rerr}_{\mathrm{min}} = 0.0\%$. The unbiased Mitchell's algorithm has $\mathrm{rerr}_{\mathrm{worst}} = -0.781\%$ for $x_A = 0.0_2$, $x_B = 0.0_2$, while $\mathrm{rerr}_{\mathrm{worst}}$ reduces to 10.65% for $x_A = 0.011\ 111\ 1_2$, $x_B = 0.100\ 000\ 0_2$.

3 | PROPOSED APPROXIMATE MULTIPLICATION

In the bfloat16 data processing, multiply-accumulate operations can accumulate multiplication results using different formats, such as FP32 or scaled fixed-point data. Referencing, [24–26] this paper assumes that the multiplication result is represented as an FP32 value.

3.1 | Motivations of proposed design

In Mitchell's algorithm, high $\mathrm{rerr}_{\mathrm{worst}}$ (11.11%) is the most serious problem degrading target applications. In, [1] iterative multiplication was briefly discussed; the error terms from a stage are multiplied in the next stage and added to compensate for the error from the previous stage. In the bfloat16 data processing, the iterative multiplication has several drawbacks. Based on (2) and (4), the error terms of the first stage, A_{ef} and B_{ef}, are as follows:

$$\begin{cases} A_{\mathrm{ef}} = 2^{k_A}\cdot\left(1-x_A\right), B_{\mathrm{ef}} = 2^{k_B}\cdot\left(1-x_B\right), x_A + x_B \geq 1 \\ A_{\mathrm{ef}} = 2^{k_A}\cdot x_A, B_{\mathrm{ef}} = 2^{k_B}\cdot x_B, x_A + x_B < 1. \end{cases} \tag{7}$$

The error terms are calculated depending on $x_A + x_B$. When $x_A + x_B \geq 1$, 2's complements of x_A and x_B represent $1-x_A$ and $1-x_B$, respectively, so that the error term calculation inverts all bits of x_A and x_B and adds value of "1." Two adders are needed to calculate error terms toward the next stage. When the next stage adopts Mitchell's algorithm, error terms A_{ef} and B_{ef} should be transformed into log-linear representations, which requires LODs and encoders [22]. Besides, a $2n$-bit adder is required to sum n-bit multiplication results from all stages.

To overcome these drawbacks, we propose a new two-stage compensated approximate multiplier. The first stage implements Mitchell's algorithm, and the second stage adopts the fixed-width multiplier with 7 bit output. After discarding several least significant bits (LSBs) of error terms A_{ef} and B_{ef} into A'_f and B'_f, two n'-bit values are exactly multiplied and $(2n'-n)$-bit LSBs of its multiplication output are discarded in the fixed-width multiplication, producing $(n = 7)$-bit output. Because the second stage adopts the exact multiplier with truncated values, LODs and encoders are not required. The fixed-width multiplication reduces hardware costs of the adder to sum the outputs from both stages. The proposed design also approximates the error terms from the first stage, where 2's complement conversion requires two adders to transfer the exact error terms. The proposed design adopts 1's complement conversion to eliminate the need for the two adders. Because the bfloat16 format has the fixed resolution of (2^{-7}) in fractions, the effect of 1's complement conversion on the error term is insignificant.

The multiplication in bfloat16 data processing can adopt the technique of, [11] which discards LSBs of inputs and multiplies them. The unbiasing technique [11] does not need high hardware costs, but it has small $\mathrm{rerr}_{\mathrm{avg}}$ values. However, as the number of discarded bits increases, $\mathrm{rerr}_{\mathrm{worst}}$ increases

rapidly. In our design, although the second stage consists of a multiplier with truncated values, rerr degraded smoothly because the second stage compensates for the error from the first stage. The fixed-width multipliers using approximate least significant adders [3,6] have been researched for digital signal processing. In the proposed design, the fixed-width multiplication in the second stage discards the least significant values. This study analyzes the performance of the proposed approximate multiplication using different types of multiplications in the two stages.

3.2 | Proposed multiplication and hardware structure

Figure 1 shows the structure of the proposed compensated logarithmic multiplier. Algorithm 1 describes its multiplication process. Given two bfloat16 data, A and B, with signs (A_S, B_S), exponents (A_e, B_e), and fraction parts (A_f, B_f). First, 7-bit A_f, B_f, and the carry-in value of "1" are added in the f-Adder to produce unbiased 8-bit f_{ma} (line 2). If the carry-out is "1," $C(1)_f = 2f_{\text{ma}}$ (line 4); otherwise, $C(1)_f = 2^n + f_{\text{ma}}$ is produced in the 1-bit Level Shifter block (line 8). In the Error Term Calcuation block, 1's complement conversions using bitwise NOT operations are applied to obtain error terms A_{ef} and B_{ef} when the carry-out is "1" (lines 5 and 6); otherwise, $A_{\text{ef}} = A_f$, $B_{\text{ef}} = B_f$ (lines 9 and 10). In the Truncator block, (n−n') LSBs of the given error terms are discarded (lines 12 and 13) and multiplied to produce C_{mul} in the Fixed-width Multiplier block (line 14), so that the Fixed-width Multiplier block generates n-bit multiplication output, denoted as $C(2)_f$ (line 15). In the m-Adder block, 9 bit $C(1)_f$ and aligned $C(2)_f$ are added to produce 9 bit $C(1,2)_f$ (line 16). Although two 9 bit values are added, 9 bit m-Adder is sufficient to produce $C(1,2)_f$ because 1's complement is applied in the error term calculation. Then, the Normalizer block generates range and the normalized C_f (lines 17 to 23). If the output significand can be equal or greater than 10.0_2, $C(1,2)_f \geq 2^{n+1}$. The fraction part of FP32 output C_f can be $\{C(1,2)_b\ [n:0],\ 15'b0\}$, which means that 15 "0" bits are concatenated in a low-order (line 18). At this time, range = 1, which means that the output exponent should increase by "1" (line 19). When the output significand is smaller than 10.0_2, $C_f = \{C(1,2)_f\ [n-1:0], 16'b0\}$ (line 21) and range = 0 (line 22). Zero detection is done (lines 24–26), which is not shown in Figure 1 for simplicity. If any exponent of A and B indicates the infinite value, the output exponent also indicates the infinite value by setting as FF_h (lines 27 and 28). When $A_e + B_e + \text{range} \geq 7F_h$, $C_e \Leftarrow A_e + B_e + \text{range} - 7F_h$ (lines 29 and 30). If $C_e > \text{FF}_h$, C_e indicates the infinite value by setting as FF_h (lines 32 and 33). An XOR gate can produce the output sign value C_s (line 35). Finally, the proposed design returns FP32 data (C_s, C_e, C_f) (line 36).

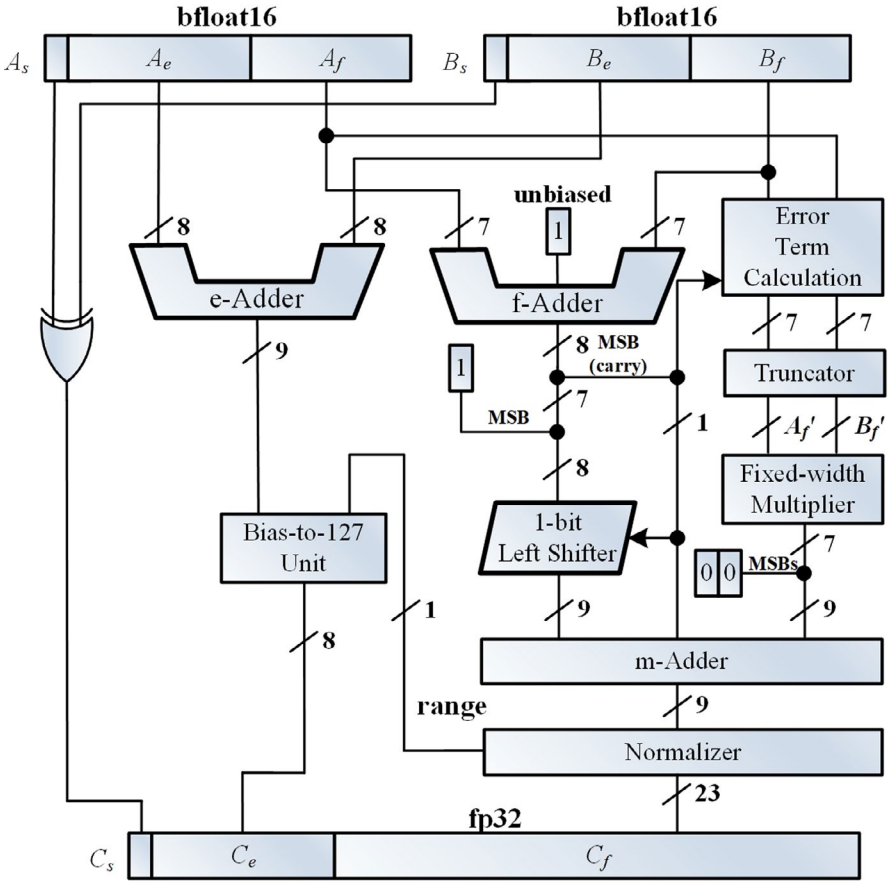


FIGURE 1 Structure of proposed design

4 | EXPERIMENTAL RESULTS

4.1 | Error and cost analysis

We evaluated the proposed design and other designs in terms of error and hardware costs. We coded functions in C to emulate the hardware blocks in the multipliers. In addition, designs were coded as combinational multipliers using dataflow descriptions in Verilog hardware description language (HDL), so that the adder, shifter, and exact multiplier used inside the multiplication are described using the HDL operators. The simulation data from the C function and HDL module of each design were compared for the equivalence check. The C functions were employed in the error analysis and convolutional neural network (CNN) emulation. The HDL modules were synthesized for the hardware cost analysis.

Algorithm 1 Multiplication in proposed design

procedure $MUL_{\text{appr}}((A_s, A_e, A_f), (B_s, B_e, B_f), n', n = 7)$
  $f_{\text{ma}} \Leftarrow A_f + B_f + 1$ ▷ adding unbiased value of '1'
  if $f_{\text{ma}} \geq 2^7$ **then**
    $C(1)_f \Leftarrow 2f_{\text{ma}}$
    $A_{ef} \Leftarrow \sim A_f$ ▷ applying 1's complement
    $B_{ef} \Leftarrow \sim B_f$ ▷ appyling 1's complement
  else
    $C(1)_f \Leftarrow 2^n + f_{\text{ma}}$
    $A_{\text{ef}} \Leftarrow A_f$
    $B_{\text{ef}} \Leftarrow B_f$
  end if
  $A'_f \Leftarrow A_e f[n-1 : n-n']$ ▷ discarding $n-n'$ LSBs
  $B'_f \Leftarrow B_e f[n-1 : n-n']$ ▷ discarding $n-n'$ LSBs
  $C_{\text{mul}} \Leftarrow A'_f \cdot B'_f$
  $C(2)_f \Leftarrow C_{\text{mul}}[2n'-1 : 2n'-n]$ ▷ discarding $2n'-n$ LSBs
  $C(1,2)_f \Leftarrow C(1)_f + C(2)_f$
  if $C(1,2)_f \geq 2^{n+1}$ **then**
    $C(1,2)_f \Leftarrow \{C(1,2)_f[n : 0], 15'b0\}$
    range $\Leftarrow 1$
  else
    $C(1,2)_f \Leftarrow \{C(1,2)_f[n-1 : 0], 16'b0\}$
    range $\Leftarrow 0$
  end if
  if $(A_e == 0)||(B_e == 0)$ **then**
    $C_e \Leftarrow 0$
    $C_f \Leftarrow 0$
  else if $(A_e == FF_h)||(B_e == FF_h)$ **then**
    $C_e \Leftarrow FF_h$
  else if $A_e + B_e + \text{range} \geq \text{bias}(7F_h)$ **then**
    $C_e \Leftarrow A_e + B_e + \text{range} - \text{bias}(7F_h)$
  end if
  if $C_e > FF_h$ **then**
    $C_e \Leftarrow FF_h$
  end if
  $C_s \Leftarrow A_s \oplus B_s$
  return (C_s, C_e, C_f) ▷ return sign, exponent, and fraction parts
end procedure

For error analysis, we simulated all significand combinations of $1 + x_A$ and $1 + x_B$ and extracted $\text{rerr}_{\text{worst}}$ and rerr_{avg}. Figure 2 summarizes $\text{rerr}_{\text{worst}}$ and rerr_{avg} according to n'-bit-truncated multiplication in the second stage. In the legends of Figure 2, prefixes $1c$ and $2c$ mean 1's and 2's complement conversions of error terms, respectively. Suffixes *biased* and *unbiased* represent with and without unbiasing technique application, respectively. In Figure 2(A), when $n' = 6$ and $n' = 7$, $|\text{rerr}_{\text{min}}| > |\text{rerr}_{\text{max}}|$, so $\text{rerr}_{\text{worst}}$ was fixed as -0.781%. As shown in Figure 2(A), the difference between $1c_{\text{unbiased}}$ and $2c_{\text{unbiased}}$ was negligible. When $n' \leq 5$, any unbiased multiplications had smaller $|\text{rerr}_{\text{worst}}|$s than those of the biased multiplications. Because of the additional overhead of 2's complement conversion, we concluded that the unbiasing technique was sufficient to reduce $\text{rerr}_{\text{worst}}$ in the proposed design. When $n' = 4$ and $n' = 5$, $\text{rerr}_{\text{worst}} = 2.25\%$ and 0.980%, respectively. Figure 2(A) shows that as n' decreased, $\text{rerr}_{\text{worst}}$ increased linearly to 11.11%. Figure 2(B) shows rerr_{avg} by averaging rerrs of all signifcand combinations. The unbiasing multiplications had small rerr_{avg}s compared with the biased cases for each n'. Notably, although rerr_{avg} increased by reducing n', for $n' = 4$ and $n' = 5$, $rerr_{\text{avg}} = 0.408\%$ and 0.048%, respectively. When $n' = 1$, because either A'_f or B'_f became $00, ..., 0_2$, there was no compensation from the second stage. Therefore, $\text{rerr}_{\text{worst}}$s and rerr_{avg}s for $n' = 0$ and $n' = 1$ were equal, as shown in Figure 2.

The HDL modules were synthesized, targeting 333 MHz target-operating clock frequency in ultramode, using 32 nm standard generic library and Design Compiler from Synopsys. The adopted cells were characterized by a typical process (TT), 1.05 V power source, and −40°C temperature. The cells were used to generate circuit area and power reports with the general switching activity of 0.1. For apple-to-apple comparisons, all described designs implemented zero and infinite number detectors.

Figure 3 illustrates hardware costs in terms of circuit area according to n'. This summary shows that the circuit area difference between unbiased and biased cases was negligible. However, the differences between 1's and 2's complement conversions of error terms were significant. For example, for $n' = 5$, the circuit areas of $1c_{\text{unbiased}}$ and $2c_{\text{unbiased}}$ were 500 μm^2 and 545.5 μm^2, respectively, so that 1's complement conversion can reduce circuit area by 8.3%.

Several other existing logarithmic multiplications were evaluated. These original works were developed for the fixed-point multiplication based on the log-linear

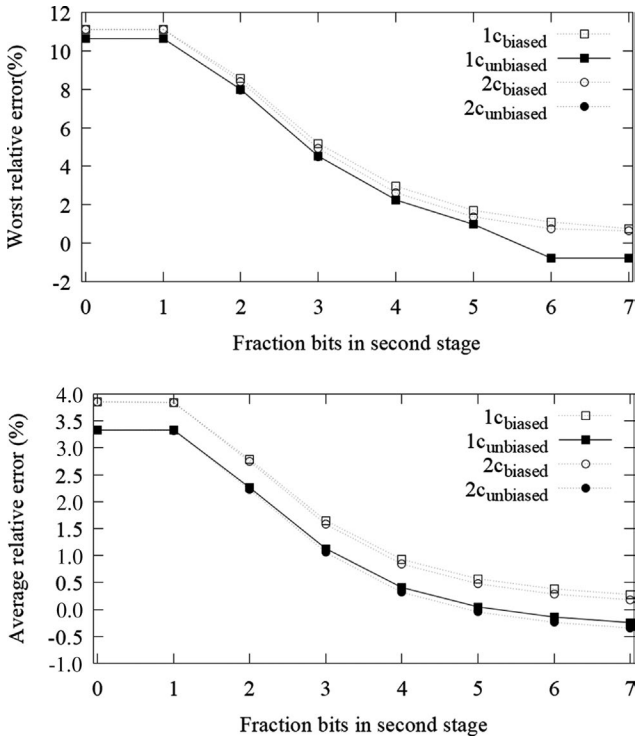


FIGURE 2 Relative errors according to fraction bits in the second stage n'. (A) $\text{rerr}_{\text{worst}}$ (B) rerr_{avg}

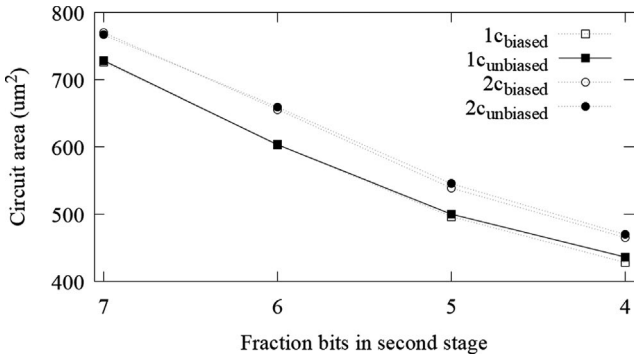


FIGURE 3 Circuit areas according to fraction bits in the second stage n'

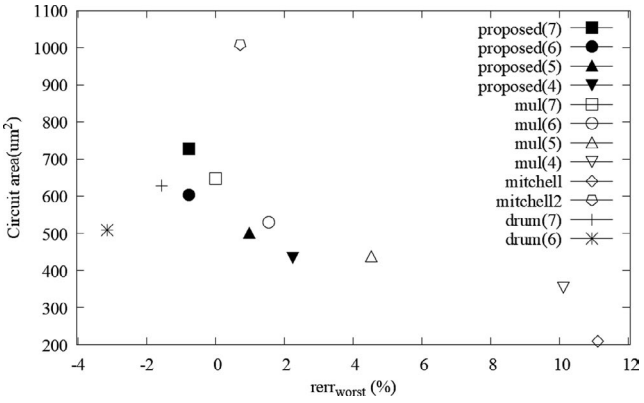


FIGURE 4 Comparison in terms of $\text{rerr}_{\text{worst}}$ and hardware costs

conversion, whereas our comparisons adopted the methods for multiplying input significands approximately. Assuming that a design discards $n-n'$ LSBs of the fraction part in bfloat16 data A and B to get truncated fractions x'_f and x'_B. Then, an $(n' + 1)$-bit multiplier denoted as mul(n') multiplies two significands $\left(1 + x'_A\right)$ and $\left(1 + x'_B\right)$. For example, for $1+x_A = 1.110\ 010\ 1_2$ and $1+x_B = 1.111\ 001\ 1_2$, when $n' = 6$, $x'_A = 0.110\ 010_2$, and $x'_B = 0.111\ 001_2$ with 6 bit fractions. In, [11] after discarding $n - n' + 1$ LSBs of the fraction part, $2^{-n'}$ is added to both x'_A and x'_B, which is denoted as drum(n'). In the unbiased truncation [11] with $n' = 6$, $1 + x'_A = 1.110\ 011_2$ and $1 + x'_B = 1.111\ 001_2$ are multiplied in drum(6), after discarding two LSBs from x_A and x_B and adding 2^{-6} to x'_A and x'_B. Therefore, we note that drum(n') required $(n' + 1)$-bit exact multiplier to multiply the truncated significands. Besides, mitchell1 and mitchell2 denoted the implementations of Mitchell's algorithm and its two-stage iterative design, where the 7 bit fractions of bfloat16 inputs were adopted.

Figure 4 shows the comparison in terms of circuit area and $\text{rerr}_{\text{worst}}$ of the proposed design and other existing designs, where proposed(n') for the proposed design are colored in black. The two-stage iterative Mitchell's algorithm provided 0.71% $\text{rerr}_{\text{worst}}$. However, the circuit area was 1008.8 μm^2. Overall, when $4 \le n' \le 6$, it was concluded that the proposed design provided better choices compared with mul(n') and drum(n'). For example, proposed(5) had 0.98% $\text{rerr}_{\text{worst}}$ and circuit area of 500.0 μm^2. Meanwhile, mul(6) had 1.54% $\text{rerr}_{\text{worst}}$ and circuit area of 530.8 μm^2.

Table 1 summarizes the total power consumption and $\text{rerr}_{\text{worst}}$ of notable approximate designs in Figure 4, where bfloat16 represents exact bfloat16 multiplication. In the proposed design, the critical path delay increased due to the two-stage error compensation. Compared with exact bfloat16 multiplication, the total power consumption of the multiplier significantly reduced in the proposed design for $n' = 5$ and $n' = 4$. Similar to the comparison of the circuit area and $\text{rerr}_{\text{worst}}$, when total power consumptions were in a similar range of values, our proposed design can outperform others in terms of $\text{rerr}_{\text{worst}}$ and circuit area.

4.2 | Evaluation on convolutional neural network

Our design was evaluated in CNNs using a modified Caffe deep learning framework [28]. In the general matrix multiplication of the modified Caffe framework, the proposed design was emulated after converting FP32 numbers into bfloat16 data. The multiplication outputs with FP32 format were accumulated in the matrix multiplication. For sufficient experiments, we evaluated our design and the counterparts

TABLE 1 Comparison in terms of the total power consumption

Multiplier	$rerr_{worst}$ (%)	Delay (ns)	Circuit Area (μm^2)	Power (μW)
bfloat16[a]	0	2.14	647.9	67.7
Proposed(5)	0.980	2.50	500.0	47.6
Proposed(4)	2.25	2.35	436.2	40.5
Mul(6)[b]	1.54	1.97	530.8	51.7
Mul(5)[b]	4.53	1.83	437.4	39.3
Drum(7)[c]	−1.57	2.08	627.8	64.7
Drum(6)[c]	−3.15	1.92	508.9	49.2
Mitchell1d	11.11	1.48	209.5	19.4

The total power consumption was obtained on 333 MHz target-operating clock frequency.

[a]Bfloat16 multiplication.

[b]Multiplication with $(n' + 1)$-bit significands.

[c]DRUM [11] with $(n + 1)$-bit significands.

[d]Mitchell [11] multiplication with 7-bit fraction.

TABLE 2 Image transformation for convolutional neural network (CNN) models

Model	Original image	Cropped image	Scaled[a]
AlexNet	256 × 256	256 × 256	No scale
VGG16	256 min side[b]	256 × 256	No scale
GoogLeNet	256 × 256	224 × 224	No scale
ResNet50	256 × 256	224 × 224	No scale
InceptionV4	320 min side[b]	299 × 299	0.017
MobileNetV1	256 min side[b]	224 × 224	0.017
MobileNetV2	256 × 256	224 × 224	0.017
SeNet-ResNet50	256 min side[b]	224 × 224	0.017
DenseNet121	256 min side[b]	224 × 224	0.017

[a]Scaling factor of normalizing pixel values in data augmentation.

[b]Image resizing by making the length of the minimum side m.

on prominent pre-trained models such as AlexNet, [29] VGG16, [30] GoogLeNet, [31] ResNet50, [32] InceptionV4, [33] MobileNetV1, [34] MobileNetV2, [35] and SeNet-ResNet50, [36] and DenseNet121. [37] The ImageNet [38] ILSVRC2012 validation dataset was adopted in this evaluation. Table 2 summarizes the test environments with input images, where "m min side" means the image resizing by making the length of the minimum side m. When the image scaling was adopted (denoted as "scaled"), each pixel value was scaled in the data augmentation. The pre-trained models were obtained from the FP32-based training process. In this experiment, approximate multiplications were applied to the models.

Inferences using the CNN models in Table 3 were performed to show Top-1 and Top-5 accuracies, where fp32,

TABLE 3 Evaluations on inference using convolutional neural networks (CNN)

Model	Multiplier	Top−1 (%)	Top−5 (%)
AlexNet [29]	fp32	56.82	79.95
	bfloat16	56.79	79.95
	mitchell1	56.55	79.79
	drum(6)	56.81	79.95
	proposed(4)	56.81	79.92
VGG16 [30]	fp32	68.35	88.44
	bfloat16	68.35	88.45
	mitchell1	68.10	88.21
	drum(6)	68.44	88.39
	proposed(4)	68.33	88.46
GoogLeNet [31]	fp32	68.92	89.14
	bfloat16	68.89	89.14
	mitchell1	67.62	88.50
	drum(6)	68.95	89.12
	proposed(4)	68.91	89.15
ResNet50 [32]	fp32	72.92	91.18
	bfloat16	72.93	91.19
	mitchell1	70.30	89.32
	drum(6)	71.64	90.29
	proposed(4)	72.89	91.06
InceptionV4 [33]	fp32	78.13	94.10
	bfloat16	78.10	94.11
	mitchell1	72.08	90.60
	drum(6)	75.33	92.42
	proposed(4)	78.08	94.06
MobileNetV1 [34]	fp32	70.72	89.92
	bfloat16	70.65	89.90
	mitchell1	31.1	54.95
	drum(6)	44.89	70.41
	proposed(4)	68.01	88.26
MobileNetV2 [35]	fp32	71.86	90.45
	bfloat16	71.82	90.43
	mitchell1	28.95	53.31
	drum(6)	50.15	75.44
	proposed(4)	69.04	88.85
SeNet-ResNet50 [36]	fp32	78.06	94.18
	bfloat16	79.06	94.18
	mitchell1	76.24	93.06
	drum(6)	77.08	93.60
	proposed(4)	77.82	94.06
DenseNet121 [37]	fp32	74.74	92.16
	bfloat16	74.75	92.19
	mitchell1	71.25	90.26
	drum(6)	73.15	91.20
	proposed(4)	74.46	92.10

bfloat16, and proposed(4) denote the FP32 multiplication, exact bfloat16 multiplication, and proposed approximate multiplication for $n' = 4$. Besides, we compared our results with evaluations with Mitchell [1] and DRUM [11] multiplications, which are denoted as mitchell1 and drum(6) for $n' = 6$. Table 3 shows that proposed(4) had considerable accuracy drops on MobileNetV1 and MobileNetV2 models. In MobileNetV1 and MobileNetV2 models, after performing the depthwise convolution for spatial filtering, the 1×1 convolution was used in the pointwise convolution. In this convolution, the number of accumulated multiplication outputs for each output element was only that of input channels. Even though rerr_{avg} was small in the proposed design, if the number of accumulated outputs was not enough, the error distribution could deviate from the expected rerr_{avg} significantly. When $n' = 6$, our Top-5 test accuracies on MobileNetV1 and MobileNetV2 models were enhanced up to 88.26% and 88.85%, respectively. In this case, the accuracy drops were uncritical compared with mitchell1 and drum(6). Therefore, we concluded that the MobileNet models required more accurate multiplication to obtain test accuracies close to the bfloat16 cases.

Except for the MobileNet models, the performance drops were negligible in terms of Top-1 and Top-5 test accuracies. Notably, when evaluating the ResNet50 model, 0.13% Top-5 accuracy was degraded compared with that using bfloat16 multiplication. Besides, compared with the evaluation results of mitchell1 and drum(6), which were significantly

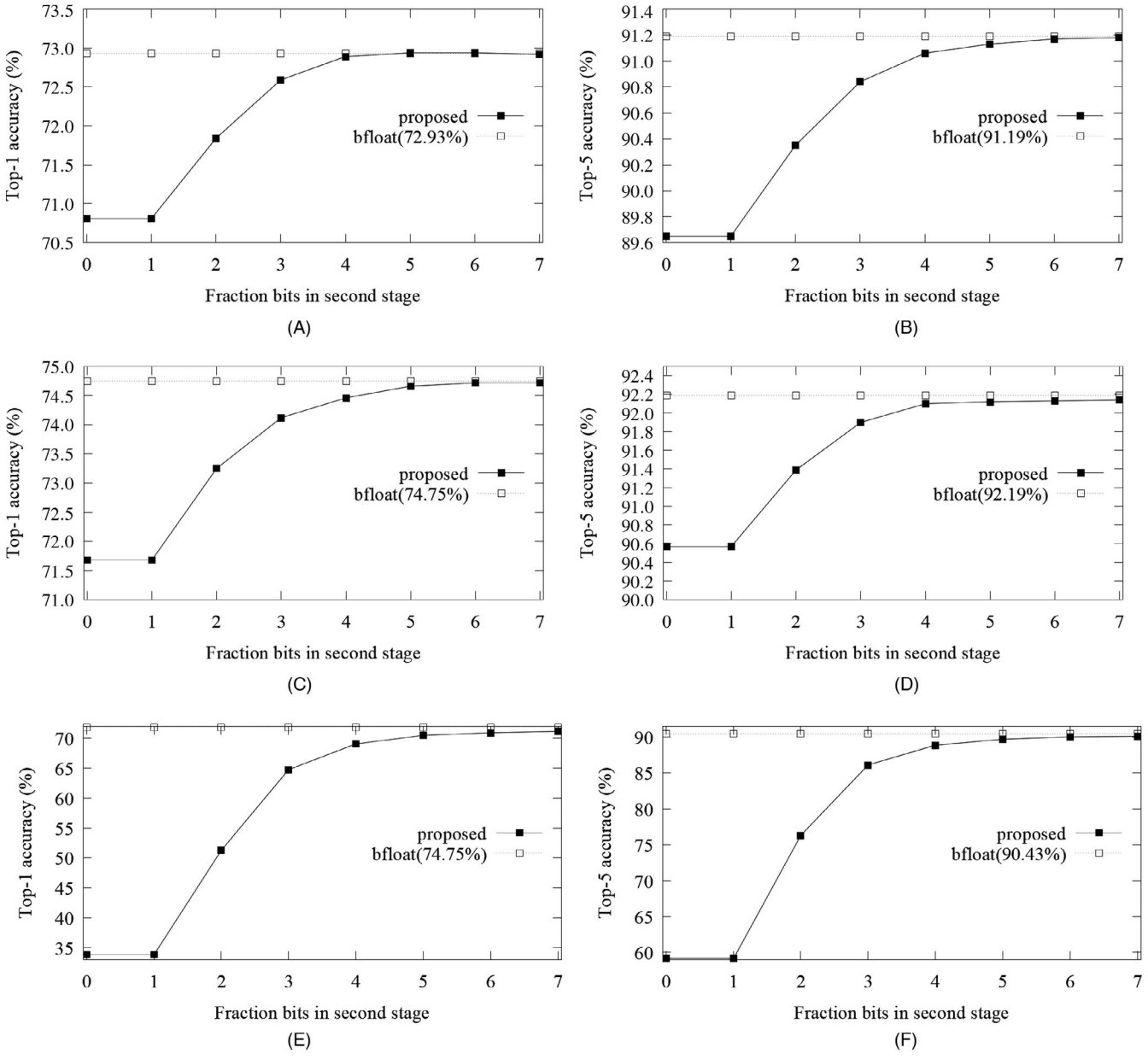


FIGURE 5 Inference accuracies according to fraction bits in the second stage n'

degraded on InceptionV4 and DenseNet121 models, the proposed design did not show considerable performance degradation. These performance comparisons revealed that the proposed design could well-compensate for errors on CNN inferences.

Figure 5 illustrates Top-1 and Top-5 accuracies with ResNet50, [32] MobileNetV2, [35] and DenseNet121 [37] models by sweeping n'. Notably, there was no difference in accuracy between the cases of $n' = 0$ and $n' = 1$. As mentioned before, because the second stage cannot compensate for the error from the first stage with $n' = 1$, the accuracy was not enhanced when $n' = 1$. On ResNet50 and DenseNet121 models, when $n' \geq 4$, there were no significant test accuracy drops (under 0.2%). However, as n' decreased, we concluded that the degraded rerr of the proposed design had adverse effects on the inference performance. Meanwhile, the test accuracy of MobileNetV2 was significantly affected depending on n'. When $n' = 1$, Top-1 and Top-5 test accuracies were only 33.90% and 59.19%, respectively. These results showed that the layer structure of MobileNetV2 could be vulnerable to the degree of multiplication approximation. For obtaining better test accuracy on the MobileNetV2 model, we concluded that the design with $n' > 4$ could be valid in terms of hardware costs and test accuracy. For example, the proposed design with $n' = 5$ can reduce the total power consumption of the multiplier by 30% and achieve 89.68% Top-5 test accuracy on the MobileNetV2 model, where the accuracy drop was under 2%, compared with that of the bfloat16 case. Overall, we concluded that low $\text{rerr}_{\text{worst}}$ and rerr_{avg} of the proposed compensated approximate multiplication could provide low-power solutions on the inference accuracies for the conventional pretrained CNN models.

5 | CONCLUSION

The proposed approximate multiplier can have good error distribution with reduced hardware costs. The proposed design adopts different types of multipliers in two stages, so that our proposed method overcomes the drawbacks of existing designs for the bfloat16 data processing in terms of hardware costs. Based on the error and cost analysis, our design can provide better choices using the approximate multiplication. Above all, based on the experiment results from various CNN inferences, we concluded that the proposed approximate multiplier can be useful in CNN inferences, allowing affordable error.

ORCID

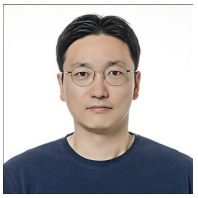
HyunJin Kim https://orcid.org/0000-0001-5017-3995

REFERENCES

1. J. N. Mitchell, *Computer multiplication and division using binary logarithms*, IRE Trans. Electr. Comput. **EC-11** (1962), no. 4, 512–517.
2. D. J. McLaren, *Improved Mitchell-based logarithmic multiplier for low-power dsp applications*, in Proc. IEEE Int. [Systems-on-Chip] SOC Conf. (Portland, OR, USA), Sept. 2003, pp. 53–56.
3. J. M. Jou, S. R. Kuang, and R. Der Chen, *Design of low-error fixed-width multipliers for DSP applications*, IEEE Trans. Circuits Syst. II Analog Digit. Signal Process. **46** (1999), no. 6, 836–842.
4. L.-D. Van, S.-S. Wang, and W.-S. Feng, *Design of the lower error fixed-width multiplier and its application*, IEEE Trans. Circuits Syst. II Analog Digit. Signal Process. **47** (2000), no. 10, 1112–1118.
5. S. J. Jon and H. H. Wang, *Fixed-width multiplier for DSP application*, in Proc. Int. Conf. Comput. Des. (Austin, TX, USA), Sept. 2000, pp. 318–322.
6. K.-J. Cho et al., *Design of low-error fixed-width modified booth multiplier*, IEEE Trans Very Large Scale Integr. VLSI Syst. **12** (2004), no. 5, 522–531.
7. S. S. Bhusare and V. S. Kanchana Bhaaskaran, *Fixed-width multiplier with simple compensation bias*, Procedia Mater. Sci. **10** (2015), 395–402.
8. Z. Babić, A. Avramović, and P. Bulić, *An itera-tive logarithmic multiplier*, Microprocess. Microsyst. **35** (2011), no. 1, 23–33.
9. P. Kulkarni, P. Gupta, and M. D. Ercegovac, *Trading accuracy for power in a multiplier architecture*, J. Low Power Electron. **7** (2011), no. 4, 490–501.
10. M. B. Sullivan and E. E. Swartzlander, *Truncated error correction for flexible approximate multiplication*, in Proc. Asilomar Conf. Signals, Syst. Comput. (ASILOMAR) (Pacific Grove, CA, USA), Nov. 2012, pp. 355–359.
11. S. Hashemi, R. Bahar, and S. Reda, *DRUM: A dynamic range unbiased multiplier for approximate applications*, in Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (Austin, TX, USA), Nov. 2015, pp. 418–425.
12. H. Jiang et al., *Approxi-mate radix-8 booth multipliers for low-power and high-performance operation*, IEEE Trans. Comput. **65** (2016), no. 8, 2638–2644.
13. S. E. Ahmed, S. Kadam, and M. B. Srinivas, *An iterative logarithmic multiplier with improved precision*, IEEE Symp. Comput. Arithmetic (ARITH), (Silicon Valley, CA, USA), July 2016, pp. 104–111.
14. R. Zendegani et al., *RoBA multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing*, IEEE Trans. Very Large Scale Integr. VLSI Syst. **25** (2017), no. 2, 393–401.
15. V. Mrazek et al., *Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods*, in Proc. Des., Autom. Test (Lausanne, Switzerland), Mar. 2017, pp. 258–261.
16. W. Liu et al., *Design of approximate logarithmic multipliers*, in Proc. Great Lakes Symp. VLSI, (Banff, Canada), May 2017, pp. 47–52.
17. M. S. Kim et al., *Low-power implementation of Mitchell's approximate logarithmic multiplication for convolutional neural networks*, in Proc. Asia S. Pac. Des. Autom. Conf. (ASP-DAC), (Jeju, Rep. of Korea), Jan. 2018, pp. 617–622.

18. I. Alouani et al., *A novel heterogeneous approximate multiplier for low power and high performance*, IEEE Embed. Syst. Lett. **10** (2018), no. 2, 45–48.
19. S. Ullah, S. S. Murthy, and A. Kumar, *Smapproxlib: Library of FPGA-based approximate multipliers*, in Proc. ACM/ESDA/IEEE Des. Autom. Conf. (DAC), (San Francisco, CA, USA), June 2018, pp. 1–6.
20. P. Yin et al., *Design of dynamic range approximate logarithmic multipliers*, in Proc. Great Lakes Symp. VLSI, (Chicago, IL, USA), May 2018, pp. 423–426.
21. M. S. Kim et al, *Efficient Mitchell's approximate log multipliers for convolutional neural networks*, IEEE Trans. Comput. **68** (2018), no. 5, 660–675.
22. H. J. Kim et al., *A cost–efficient iterative truncated logarithmic multiplication for convolutional neural networks*, in Proc. IEEE Symp. Comput. Arithmetic (ARITH), (Kyoto, Japan), June 2019, pp. 108–111.
23. Z. Babic, A. Avramovic, and P. Bulic, *An iterative Mitchell's algorithm based multiplier*, in Proc. IEEE Int. Symp. Signal Process. Inform. Technol. (Sarajevo, Bosnia and Herzegovina), Dec. 2008, pp. 303–308.
24. N. Burgess et al., *Bfloat16 processing for neural networks*, in Proc. IEEE Symp. Comput. Arithmetic (ARITH), (Kyoto, Japan), June 2019, pp. 88–91.
25. D. Lutz, *Arm floating point 2019: Latency, area, power*, in Proc. IEEE Symp. Comput. Arithmetic (ARITH), (Kyoto, Japan), June 2019, pp. 97–98.
26. D. Kalamkar et al., *A study of bfloat16 for deep learning training*, arXiv preprint, CoRR, 2019, arXiv: 1905.12322.
27. K. H. Abed and R. E. Siferd, *CMOS VLSI implementation of a low-power logarithmic converter*, IEEE Trans. Comput. **52** (2003), no. 11, 1421–1433.
28. Y. Jia et al., *Caffe: Convolutional architecture for fast feature embedding*, in Proc. ACM Int. Conf. Multimed. (Orlando, FL, USA), Nov. 2014, pp. 675–678.
29. A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Proc. Int. Conf. Neural Inform. Process. Syst. (Red Hook, NY, USA), Dec. 2012, pp. 1097–1105.
30. K. Chatfield et al., *Return of the devil in the details: Delving deep into convolutional nets*, arXiv preprint, CoRR, 2014, arXiv: 1405.3531.
31. C. Szegedy et al., *Going deeper with convolutions*, in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (Boston, MA, USA), June 2015, pp. 1–9.
32. K. He et al., *Deep residual learning for image recognition*, in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (Las Vegas, NV, USA), June 2016, pp. 770–778.
33. C. Szegedy et al., *Inception-v4, inception-resnet and the impact of residual connections on learning*, arXiv preprint, CoRR, 2016, arXiv: 1602.07261.
34. A. G. Howard et al., *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, arXiv preprint, CoRR, 2017, arXiv: 1704.04861.
35. M. Sandler et al., *Mobilenetv2: Inverted residuals and linear bottlenecks*, in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (Salt Lake City, UT, USA), June 2018, pp. 4510–4520.
36. J. Hu, L. Shen, and G. Sun, *Squeeze-and-excitation networks*, in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (Salt Lake City, UT, USA), June 2018, pp. 7132–7141.
37. G. Huang, et al., *Convolutional networks with dense connectivity*, IEEE Trans. Pattern Anal. Mach. Intell. (2019), **1**.
38. O. Russakovsky, et al., *Imagenet large scale visual recognition challenge*, Int. J. Comput. Vis. (IJCV) **115** (2015), no. 3, 211–252.

AUTHOR BIOGRAPHY



HyunJin Kim received a Ph.D in Electrical and Electronic Engineering, a master degree and a bachelor degree in Electrical Engineering, all from Yonsei University, Republic of Korea. He worked as a mixed-signal VLSI circuit designer at Samsung Electromechanics (2002–2005), and as a senior engineer in a flash memory controller project at Samsung Electronics (2010–2011). He is currently an associate professor in the School of Electronics and Electrical Engineering of Dankook University, Republic of Korea. His current research interests reside in the realm of approximate computing based on the arithmetic solution for deep neural network implementation, string matching algorithm development, embedded & parallel system, and system-on-chip (SoC) design.