# High-Performance FPGA-Based CNN Accelerator With Block-Floating-Point Arithmetic

Xiaocong Lian, *Member, IEEE*, Zhenyu Liu, *Member, IEEE*, Zhourui Song, Jiwu Dai, Wei Zhou, *Member, IEEE*, and Xiangyang Ji, *Member, IEEE*

***Abstract*— Convolutional neural networks (CNNs) are widely used and have achieved great success in computer vision and speech processing applications. However, deploying the large-scale CNN model in the embedded system is subject to the constraints of computation and memory. An optimized block-floating-point (BFP) arithmetic is adopted in our accelerator for efficient inference of deep neural networks in this paper. The feature maps and model parameters are represented in 16-bit and 8-bit formats, respectively, in the off-chip memory, which can reduce memory and off-chip bandwidth requirements by 50% and 75% compared to the 32-bit FP counterpart. The proposed 8-bit BFP arithmetic with optimized rounding and shifting-operation-based quantization schemes improves the energy and hardware efficiency by three times. One CNN model can be deployed in our accelerator without retraining at the cost of an accuracy loss of not more than 0.12%. The proposed reconfigurable accelerator with three parallelism dimensions, ping-pong off-chip DDR3 memory access, and an optimized on-chip buffer group is implemented on the Xilinx VC709 evaluation board. Our accelerator achieves a performance of 760.83 GOP/s and 82.88 GOP/s/W under a 200-MHz working frequency, significantly outperforming previous accelerators.**

***Index Terms*— Block floating point (BFP), convolutional neural network (CNN) accelerator, field-programmable gate array (FPGA), three-level parallel.**

## I. Introduction

CONVOLUTIONAL neural network (CNN) is the most widely used machine learning method and has emerged as the best choice in fields such as image classification [1], object detection [2], semantic segmentation [3], and speech processing tasks [4]. The outstanding performance of CNN-based algorithms is achieved with the overhead of enormous computation and memory resources, especially with the increase in network depth. Over the past few years, the computational power of training large CNN models under acceptable timing constraints has become feasible by means of general-purpose graphics processing units (GPUs).

However, the forward inference process of CNNs also involves high computational complexity and requires good real-time performance. For instance, the VGG-16 model, which involves 138 million parameters, consumes over 30 billion multiplication and addition operations for the feed-forward process of a 224 × 224 RGB image [5]. When confronting applications with larger images, the number of arithmetic operations increases exponentially. Thus, traditional general-purpose processors cannot meet the requirements of real-time applications, and numerous CNN accelerators based on GPUs and field-programmable gate array (FPGA)/application-specified integrated circuit (ASIC) have been proposed [6]–[16].

Compared to the GPU-based CNN accelerator, FPGA has the following advantages.

1) The programmable attribute of FPGA makes it feasible for connecting FPGA directly to various image sampling devices. In contrast, the GPU accelerator requires PCIe to communicate with the sampling devices. Consequently, the data transfer latency is greatly optimized by FPGA-based design.
2) The FPGA accelerator is more efficient in terms of communication when integrating the CNN accelerator with other original FPGA intelligent properties (IPs). The obstacles of transplanting a CNN on FPGA are derived from two factors, floating-point (FP) arithmetic overhead and data traffic requirements, which severely degrade the throughput and the energy efficiency of the accelerators.

Data reuse, compression and pruning are common methods for overcoming the above-mentioned obstacles. For example, a delicate data reuse scheme that reconfigures FPGA for computations of different layers was proposed to reduce external data access and arithmetic operations [12]. With the help of the roofline model, a method that dynamically chooses optimization techniques, including loop tiling and transformation, was proposed by Zhang *et al.* [7], which can balance the computational throughput and memory bandwidth. A deep compression method was proposed in [17] and consists of three stages: pruning, trained quantization, and Huffman coding; the model

Manuscript received November 26, 2018; revised March 3, 2019; accepted March 22, 2019. Date of publication May 16, 2019; date of current version July 24, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61827804, Grant 61836012, Grant 61620106005, and Grant 61325003. *(Corresponding author: Xiangyang Ji.)*

X. Lian and X. Ji are with the Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: lian900625@tsinghua.edu.cn; xyji@tsinghua.edu.cn).

Z. Liu is with the Tsinghua National Laboratory for Information Science and Technology, Research Institute of Information Technology, Tsinghua University, Beijing 100084, China (e-mail:liuzhenyu73@mail.tsinghua.edu.cn).

Z. Song is with the School of Cyberspace Security, Beijing University of Posts and Telecommunications (BUPT), Beijing 100876, China (e-mail: zrsong99@163.com).

J. Dai and W. Zhou are with the School of Electronics and Information, Northwestern Polytechnical University, Xi'an 710129, China (e-mail: mr_dai333@mail.nwpu.edu.cn; zhouwei@nwpu.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2019.2913958

can significantly reduce the scale of CNN models without obvious accuracy loss. However, the main drawback of the above-mentioned methods lies in the time-consuming model retraining.

By adopting a lower bit length data format, the performance of the CNN accelerator in terms of chip area, power efficiency, and memory requirement can be improved greatly. Furthermore, as FPGA lacks FP arithmetic units, researchers have used a low-bit integer format instead of the traditional single precision FP format to reduce memory bandwidth and computational resource requirements. A 16-bit fixed-point representation with stochastic rounding was proposed in [18], without an obvious degradation in accuracy. A 8-/16-bit fixed-point quantization method was proposed in [9], which can find the best exponent bit for each layer. However, a shared weakness of the methods mentioned above is also that time-consuming retraining required to amend the weights. Nvidia proposed an 8-bit fixed-point inference architecture that does not require additional fine-tuning or retraining [19]. However, the architecture requires FP division in the quantization phase and FP multiplication in the dequantization phase.

Recently, a flexible low-bit length numeric format—block FP (BFP)—has begun to draw attention among academics. In [20], an adaptive numeric format for efficient training of deep neural networks (DNNs) was proposed, which consists of a 16-bit mantissa and a 5-bit shared exponent. The limitation of this paper stems from the complex intelligent exponent management algorithm. A BFP arithmetic was proposed by our group for neural network inference [21]. The 8-bit mantissa and 5-bit shared exponent can satisfy the applications of typical neural networks. However, the rough processing of the rounding and zero setting model hinders further reduction of the mantissa length. In addition, this method has not been deployed on the hardware platform.

To overcome the above-mentioned obstacles, we first develop an optimized BFP arithmetic to reduce the off-chip traffic and hardware cost with negligible classification accuracy loss. Next, we adopt three-level parallel convolution PUs, a ping-pong memory access scheme, and an optimized on-chip buffer group to improve the throughput of our accelerator. Finally, the reconfigurable accelerator is deployed on a Xilinx VC709 evaluation board, yielding high performance that significantly outperforms previous accelerators. The contributions of this paper are summarized as follows.

1) A BFP-based arithmetic is employed in our FPGA-based CNN accelerator, in which merely shifting operations are required in the quantization and dequantization phases. The design can reduce memory and off-chip bandwidth requirements by 50% and 75% compared to the 32-bit FP counterpart, with an accuracy loss of not more than 0.12%. An optimized rounding and zero setting model is developed to further improve the classification accuracy with a low-bit mantissa.
2) A mathematical model for analyzing the error propagation in BFP formatted neural network is proposed in this paper. The average deviation is 4.64 dB, and thus, the model can efficiently support the verification of CNN accelerator design. Our BFP arithmetic can be directly applied to different CNN models without retraining and fine-tuning.
3) Our accelerator consists of 1024 PUs, and it can achieve three-level parallelism (input channel parallelism, output channel parallelism, and pixel parallelism). The ping-pong memory access scheme can eliminate the DRAM transition latency between the write and read modes. An optimized on-chip buffer group is adopted to reduce the data traffic with off-chip memory.

The remainder of this paper is organized as follows. The BFP arithmetic and corresponding error analysis is explained in Section II. The related hardware implementation of our accelerator is presented in Section III. Section IV presents the experimental results. Finally, conclusions are drawn in Section V.

## II. Block-Floating-Point Arithmetic-Oriented CNN

### A. Preliminary of Block-Floating-Point Arithmetic

An $N$-data block represented with the BFP format consists of two parts: $N$ mantissas and one exponent shared by the $N$ numbers in a block. The process of the BFP conversion is defined as follows. Assuming that $\mathbf{X}$ is a data set containing $N$ FP numbers, we can express the set as

$$\begin{aligned}\mathbf{X} &= (x_1, \ldots, x_i, \ldots, x_N) \\ &= (m_1 \times 2^{e_1}, \ldots, m_i \times 2^{e_i}, \ldots, m_N \times 2^{e_N}).\end{aligned} \tag{1}$$

The largest exponent in $\mathbf{X}$ is defined as the block exponent $\epsilon_X$

$$\epsilon_X = \max_i e_i \quad i \in \{1, 2, \ldots, N\}. \tag{2}$$

After deriving the common block exponent $\epsilon_X$, the mantissa number $m_i$ is right shifted by $d_i$ bits, where $d_i = \epsilon_X - e_i$. Thus, the BFP format of $\mathbf{X}$, i.e., $\mathbf{X}_b$ is expressed as

$$\begin{aligned}\mathbf{X}_b &= (x_{b1}, \ldots, x_{bi}, \ldots, x_{bN}) \\ &= M_{bX} \times 2^{\epsilon_X} \\ &= (m_{b1}, \ldots, m_{bi}, \ldots, m_{bN}) \times 2^{\epsilon_X}\end{aligned} \tag{3}$$

where $m_{bi} = m_i >> d_i$ is the BFP formatted mantissa.

### B. Convolution Operations in CNN

The convolutional layer is the most computationally intensive component in CNN and mainly consists of two key operations: local correlation and receptive field sliding. Each kernel is treated as a filter, calculating the local feature. The forward propagation process of the filter calculates the node in the output feature maps through the convolution of the input feature maps and kernel matrix. Suppose that the current layer has $C_i$ input feature maps and $C_o$ output feature maps. There are $C_i \times K_W \times K_H \times C_o$ corresponding weights in the 4-D kernel matrix. Let $w^n_{i,x,y}$ denote the weight of filter node $(i, x, y)$ corresponding to the $n$th node of the output unit matrix, and $b^n$ represents the bias corresponding to the $n$th node of the output unit matrix. The $n$th node of the output unit matrix $of\, m^n$ can be derived by

TABLE I
HARDWARE COST OF DIFFERENT DATA PRECISION OPERATIONS

| Operation | Energy(pJ) | Area($\mu m^2$) |
|---|---|---|
| 8b fixed Add | 0.03 | 36 |
| 32b fixed Add | 0.1 | 137 |
| 16b floating Add | 0.4 | 1360 |
| 32b floating Add | 0.9 | 4184 |
| 8b fixed Mult | 0.2 | 282 |
| 32b fixed Mult | 3.1 | 3495 |
| 16b floating Mult | 1.1 | 1640 |
| 32b floating Mult | 3.7 | 7700 |

$$ofm^n = ReLU\left(\sum_{i=1}^{C_i}\sum_{x=1}^{K_W}\sum_{y=1}^{K_H} ifm_{i,x,y} \times w^n_{i,x,y} + b^n\right)$$

where $ifm_{i,x,y}$ is the node $(i, x, y)$ of the input matrix. A rectified linear unit (ReLU) is used as a nonlinear function to improve the expressive power of the neural network by adding nonlinear factors.

### C. Data Flow of the BFP Arithmetic in Our Accelerator

The hardware costs of different data precision operations [27] are shown in Table I. The results show that the energy and area costs of fixed-point operations are smaller than those of FP operations, especially for addition operations. For the FPGA-based CNN accelerator design, the fixed-point arithmetic is also much more efficient compared with FP arithmetic. A 16-bit four-stage FP multiplier requires 2 DSP, 51 LUT, and 95 flip-flops (FFs) for a maximum working frequency of 219 MHz, while a 16-bit fixed-point multiplier achieves a working frequency of 300 MHz with only one DSP. Two common quantization methods used in the fixed-point arithmetic accelerators are shown in Fig. 1. The scheme in [9] does not increase the on-chip resource consumption; however, it introduces additional test and fine-tuning overheads. With an 8-bit data length, the largest accuracy loss in [9] accounted for 6%. The method in [19] can maintain the classification accuracy without retraining and fine-tuning. However, FP divisions and multiplications are required in the quantization and dequantization phases.

To overcome the above-mentioned obstacles, a BFP arithmetic-based accelerator without retraining and fine-tuning is proposed in this paper. The precision of BFP quantization is determined by the bit length of the mantissa and the block partition mode. In this paper, the mantissa bit length is defined as 8 for typical CNN models, which will be discussed in detail in Section IV, and thus determining how to partition the block is the key issue. The process of the convolution operation is described as

$$\mathbf{O}_{C_o \times M} = \mathbf{W}_{C_o \times K}\mathbf{I}_{K \times M} \tag{4}$$

where $\mathbf{O}$, $\mathbf{W}$, and $\mathbf{I}$ represent matrices transformed from output feature maps, kernels, and input feature maps, respectively. $C_o$ is the number of output channels, while

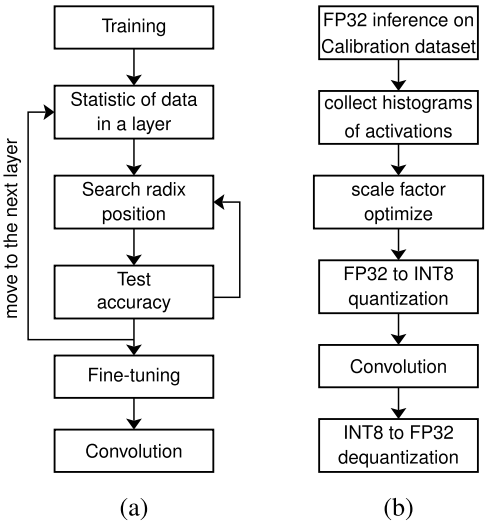



Fig. 1. Data quantization flow of previous CNN accelerators. (a) Angel-Eye [9]. (b) TensorRT [19].

TABLE II
COMPLEXITY OF FOUR BFP TRANSFORM METHODS ($L_W$ IS THE MANTISSA BIT LENGTH OF WEIGHT, $L_I$ IS THE MANTISSA BIT LENGTH OF INPUT FEATURE MAP, THE SIGN BIT IS INCLUDED IN $L_W$ AND $L_I$, $L_e$ IS THE EXPONENT BIT LENGTH, "TIME" IS THE NUMBER OF TIMES NEEDED TO RUN BLOCK FORMATTING)

| Method | storage of $\mathbf{W}$ | storage of $\mathbf{I}$ | Time |
|---|---|---|---|
| Equation(4) | $L_W + L_e/(C_o \times K)$ | $L_I + L_e/(K \times M)$ | 2 |
| Equation(5) | $L_W + L_e/K$ | $L_I + L_e/K$ | $C_o + M$ |
| Equation(6) | $L_W + L_e/K$ | $L_I + L_e/(K \times M)$ | 1+$C_o$ |
| Equation(7) | $L_W + L_e/(C_o \times K)$ | $L_I + L_e/K$ | 1+M |

$K = C_i \times K_W \times K_H$ and $M = O_W \times O_H$ are the sizes of the filters and receptive fields.

There are three other ways to perform matrix multiplication: entrywise, rowwise, and columnwise, which are described by the following equations:

$$o_{cm} = \vec{w}_c^T \cdot \vec{i}_m \tag{5}$$
$$\vec{o}_c^T = \vec{w}_c^T \cdot \mathbf{I} \tag{6}$$
$$\vec{o}_m = \mathbf{W} \cdot \vec{i}_m. \tag{7}$$

The above-mentioned equations represent four different ways to realize block partition. Equation (4) shows that $\mathbf{W}$ and $\mathbf{I}$ constitute two blocks individually, which represents the most commonly used fixed-point method. This method has minimal storage requirements but will seriously affect data accuracy. In (5), $\mathbf{W}$ and $\mathbf{I}$ are divided into $C_o$ and $M$ blocks, respectively. This method has maximal storage requirements with negligible accuracy loss. Equations (6) and (7) are balanced approaches with respect to the storage requirement and accuracy loss, and divide $\mathbf{W}$ or $\mathbf{I}$ into multiple blocks, respectively. Table II shows the complexity of the above four BFP transform methods.

For typical CNN models, $M$ is much greater than $C_o$. An analysis given in Table II reveals that the methods in (5) and (7) involve thousands of block formatting

TABLE III
ACCURACY ANALYSIS OF TWO BFP TRANSFORM METHODS FOR VGG-16

| Method | Top-1 | Top-5 |
|---|---|---|
| Equation(4) | 0.6672 | 0.8776 |
| Equation(6) | 0.6831 | 0.8844 |
| Floating point | 0.6834 | 0.8846 |

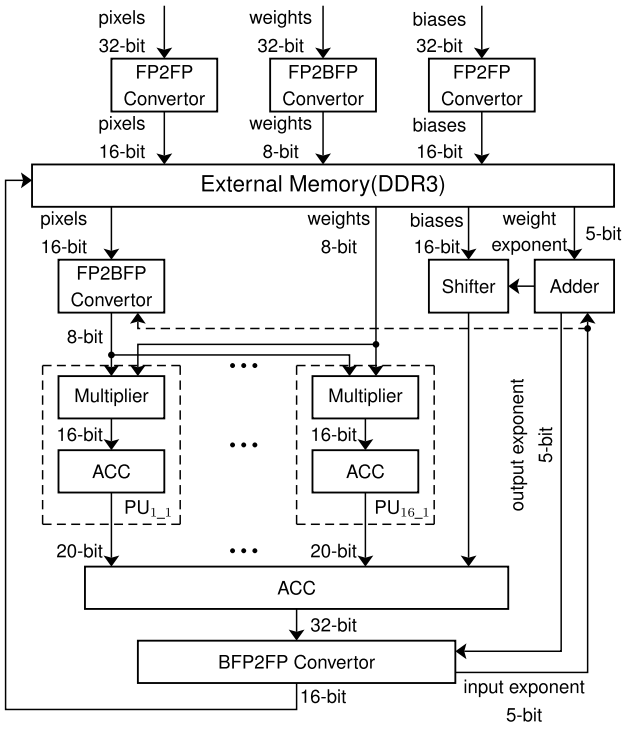



Fig. 2. Data flow of our CNN accelerator in one output channel (FP2BFP: FP to BFP, BFP2FP: BFP to FP, FP2FP: 32-bit FP to 16-bit FP, and ACC: accumulator).

operations, which slows the entire calculation process. We test the other two methods in the VGG-16 model on the ILSVRC12 data set [22], and the results are shown in Table III. The experiment reveals that the top-1 and top-5 accuracies of (6) are 1.59% and 0.68% higher than those of (4). Thus, the method in (6) is adopted in this paper to partition the block.

The data flow of our CNN accelerator in one output channel is shown in Fig. 2. The input pixels, weights, and biases are first converted to the corresponding format and stored in DDR3 memory. The input pixels are represented in a 16-bit FP format in the DDR3 memory and are converted to an 8-bit BFP format in the FP2BFP module. The biases are also stored with a 16-bit FP format in the DDR3 memory, and the mantissa bits are shifted by the difference between the bias exponent and output exponent in the shifter module. The weights are computed with an 8-bit integer in the multiplier. The products of the multiplier, adder, and accumulator are collected without bit truncation. The final output pixels adopt a 32-bit BFP format, which will increase the memory and bandwidth requirements. As the largest exponent of all the output feature maps is unavailable during the convolution procedure, the output pixels are converted to a 16-bit FP format in the BFP2FP module and transferred to the DDR3 memory. The block exponent of all the output feature maps is registered as the block exponent of next layer input feature maps after completing the convolution of the current layer. Only shifting operations are required in the FP2BFP and BFP2FP modules.

TABLE IV
ROUNDING MODE IN IEEE 754 STANDARD

| mode | interpretation |
|---|---|
| RN | round to nearest, ties to even |
| RZ | round towards zero |
| RU | round towards + $\infty$ |
| RD | round towards - $\infty$ |

TABLE V
PERFORMANCE OF THREE ROUNDING ALGORITHMS (TOP-1 ACCURACY, 32-BIT FLOATING-POINT RESULTS AS A REFERENCE)

| network | RN mode | RZ mode | RI mode | Reference |
|---|---|---|---|---|
| VGG-16 | 68.31% | 68.29% | 68.26% | 68.34% |
| GoogleNet | 68.86% | 68.83% | 68.85% | 68.93% |
| ResNet-50 | 72.76% | 72.69% | 72.70% | 72.87% |

### *D. Round-to-Nearest and Zero-Setting Algorithm*

The accuracy loss in our accelerator mainly derives from the BFP2FP and FP2BFP conversions. Truncation and rounding are two common methods to handle shifted bits. Because the truncation mode introduces a biased error, the error will accumulate between layers and eventually produce an obvious deviation. In contrast, the rounding mode only introduces unbiased Gaussian white noise, and no accumulated deviations exist.

Because the bit length is limited, the value that FP number can accurately represent is finite. Between the two adjacent FP numbers, there must be an infinite number of real numbers that cannot be accurately represented by FP numbers. In the IEEE 754 standard, these numbers are approximated by the nearest FP numbers, namely, the rounding. There are four common rounding modes [31], as shown in Table IV.

The above-mentioned modes are equivalent except when $x = (y_1 + y_2)$, where $y_1$ and $y_2$ are two successive representable numbers. If one of the latter three rounding modes is used in the calculation, the error is likely to increase layer by layer. Therefore, rounding half up and rounding half down are reasonable. From a statistical point of view, the probability that the number $x$ is even or odd is 50% for a large number of calculations. Thus, the round-to-nearest (RN) mode can minimize the expected error.

The latter two methods can be combined into round toward infinite (RI). We test the RN, round toward zero (RZ), and RI modes on the Caffe [24] scheme, and the results are shown in Table V. The results show that the classification accuracy of the RN mode is higher than that of the other two modes for the typical network models. Therefore, the RN mode is adopted in this paper.

The 32-bit FP number consists of a sign bit, exponent bits, and mantissa bits, where the length of the mantissa is 24 (including an invisible "1"). In our previous work [21], a loose zero setting method was adopted. When the right-shifted length $d_i$ is equal to or greater than 24 in the weights FP2BFP module (for the conversion of input feature map, $d_i$ is equal

TABLE VI

FREQUENCY OF ZERO SETTING ERROR IN OUR PREVIOUS WORK [21] (TAKING WEIGHTS AS EXAMPLES)

| mantissa length | 5-bit | 6-bit | 7-bit | 8-bit |
|---|---|---|---|---|
| VGG-16 | 21.03% | 12.42% | 6.56% | 3.34% |
| GoogLeNet | 17.99% | 10.01% | 5.17% | 2.61% |
| ResNet | 18.62% | 10.48% | 5.43% | 2.74% |

TABLE VII

PERFORMANCE OF OUR OPTIMIZED BFP ARITHMETIC (THE 32-BIT FP RESULT IS SET AS THE BASELINE)

| | | 6-bit mantissa | | 7-bit mantissa | | 8-bit mantissa | |
|---|---|---|---|---|---|---|---|
| | | [21] | ours | [21] | ours | [21] | ours |
| VGG-16 | top-1 | 30.96% | 2.02% | 2.68% | 0.17% | 0.02% | 0.03% |
| | top-5 | 27.78% | 1.35% | 1.08% | 0.15% | 0.01% | 0.02% |
| GoogLeNet | top-1 | 2.72% | 0.89% | 0.28% | 0.15% | 0.14% | 0.07% |
| | top-5 | 1.64% | 0.62% | 0.52% | 0.13% | 0.16% | 0.06% |
| ResNet | top-1 | 10.38% | 6.83% | 1.28% | 0.95% | 0.08% | 0.11% |
| | top-5 | 7.02% | 4.74% | 1.16% | 0.62% | 0.18% | 0.12% |

to or greater than 11), the BFP formatted number $x_{bi}$ is set to zero. $d_i$ and $x_{bi}$ are defined in Section II-A. Because the highest bit of the mantissa is 1, rounding up instead of rounding down is usually required when $d_i$ is equal to 24. The statistical results in Table VI show that the frequency of this situation increases significantly as the length of the mantissa decreases. Thus, the method results in high rounding errors for low-bit mantissa BFP arithmetic.

To overcome the above-mentioned obstacle, this paper proposes an optimized zero setting method. Only when the right-shifted length $d_i$ is greater than 24 is the BFP formatted number $x_{bi}$ set to zero. Our method can significantly reduce the rounding error in FP2BFP conversion; a detailed performance analysis is provided in Table VII.

Our optimized BFP arithmetic and previous work [21] are compared in Table VII. The results show that the rounding and zero setting algorithm employed in this paper can significantly improve the classification accuracy under the condition of a low-bit mantissa.

### *E. Error Analysis of BFP-Oriented Convolution Operations*

Based on the data flow of our accelerator, we analyze the error of our arithmetic in four stages. The first stage concerns the quantization error in the FP2BFP conversion, and the error accumulation in the matrix multiplication and addition is explained in the second stage. The third stage is the quantization error in the BFP2FP conversion, and the error propagation between the convolutional layers is described in stage four.

For block $\mathbf{X}$, the quantization error of FP2BFP conversion can be defined as $\alpha = e_{\text{bm}} \cdot 2^{\xi}$, where $e_{\text{bm}}$ is the quantization error of the block mantissa, which can be modeled as an uncorrelated random variable [30]. $\xi$ is the block exponent, which can be assumed to be uncorrelated. When the RN mode is used in the conversion, the mean of quantization error $\alpha$ is zero, and the variance is expressed as

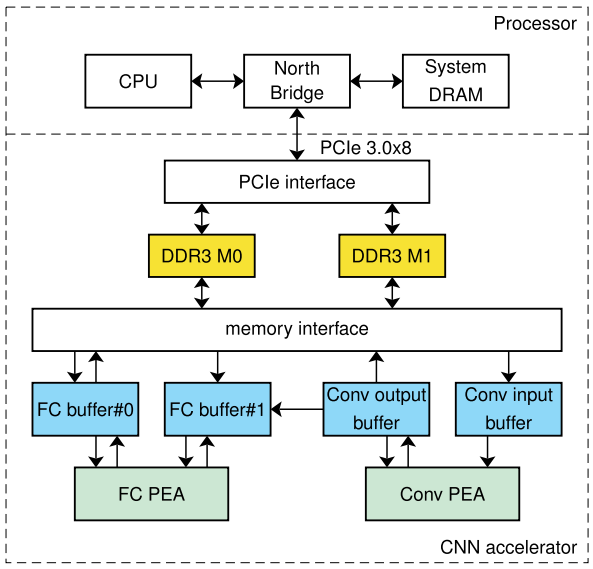



Fig. 3. High-level block diagram for the proposed CNN accelerator.

$$\sigma_{\alpha}^2 = \frac{2^{-2L_m}}{12} \cdot \sum_{i=1}^{N_{\xi}} p_{\xi_i} 2^{2\xi_i} \tag{8}$$

where $p_{\xi_i}(i = 1, \ldots, N_{\xi})$ is the probability mass function (PMF) of the block exponents. $N_{\xi} = 2^{L_e}$ is the number of available block exponent levels, and $L_e$ is the bit length of the block exponent.

According to our previous work [21], the signal-to-noise ratio (SNR) of the BFP formatted input matrix and weight matrix is

$$\text{SNR}_{\mathbf{I}} = 10 \cdot \log_{10} \frac{E(\mathbf{I}^2)}{\sigma_I^2} \tag{9}$$

$$\text{SNR}_{\mathbf{W}} = 10 \cdot \log_{10} \frac{\sum_{c=1}^{C_o} E\left(\mathbf{W}_c^2\right)}{\sum_{c=1}^{C_o} \sigma_{W_c}^2} \tag{10}$$

where $E(\mathbf{I}^2)$ is the mean square of the input matrix, $\sigma_I^2$ is the quantization error energy of input matrix $\mathbf{I}$. $E\left(\mathbf{W}_c^2\right)$ is the mean square of the weight matrix's $c$th row vector, and $\sigma_{W_c}^2$ is the quantization error energy of the $c$th row vector.

The calculations in convolutional layers can be written as inner products of the vectors. Given two vectors $\vec{\mathbf{I}}$ and $\vec{\mathbf{W}}$ of length $K$, we can obtain the BFP formatted versions $\vec{\mathbf{I}}_b = \vec{\mathbf{I}} + \vec{\mathbf{I}}_e$ and $\vec{\mathbf{W}}_b = \vec{\mathbf{W}} + \vec{\mathbf{W}}_e$. $\vec{\mathbf{I}}_e$ and $\vec{\mathbf{W}}_e$ are quantization errors. Here, we ignore the effect of biases on the error propagation. The energy of the BFP formatted inner product $E\left(\mathbf{O}_b^2\right)$ is

$$\begin{aligned} E\left(\mathbf{O}_b^2\right) &= E((\vec{\mathbf{I}}_b \cdot \vec{\mathbf{W}}_b)^2) \\ &= E((\vec{\mathbf{I}} \cdot \vec{\mathbf{W}})^2) + E((\vec{\mathbf{I}}_e \cdot \vec{\mathbf{W}})^2) \\ &\quad E((\vec{\mathbf{I}} \cdot \vec{\mathbf{W}}_e)^2) + E((\vec{\mathbf{I}}_e \cdot \vec{\mathbf{W}}_e)^2). \end{aligned} \tag{11}$$

Because $\vec{\mathbf{I}}_e$ and $\vec{\mathbf{W}}_e$ are independently distributed while ignoring the higher order components of the error, we can

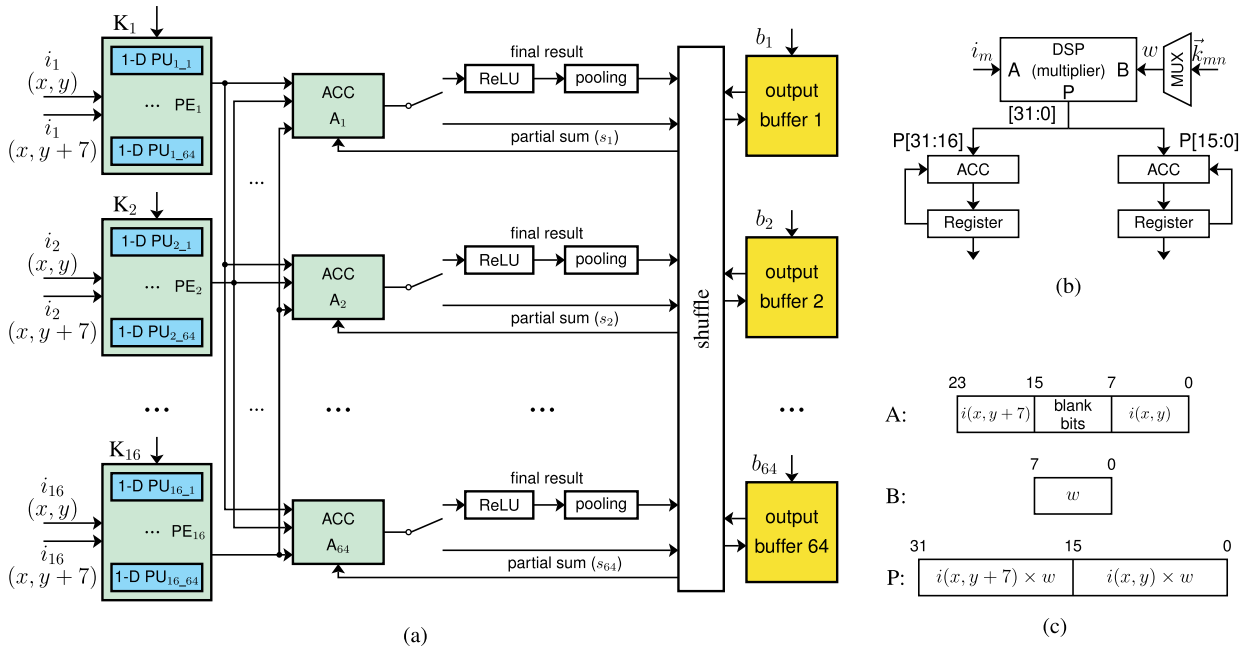



Fig. 4. Architecture of the convolution PEA. (a) Overall architecture. (b) Architecture of $\mathrm{PU}_{m\_n}$. (c) Data format of the DSP port ($i_m(x, y)$: the pixel in the position $(x, y)$ of the $m$th input channel, $\mathrm{K}_m$: the convolution kernels of 64 output channels in the $m$th input channel, $\mathrm{K}_m = [\vec{k}_{m1}, \vec{k}_{m2}, \ldots, \vec{k}_{m64}]$, $\vec{k}_{mn}$: the convolution kernel of the $n$th output channel in $m$th input channel, $b_n$: the bias of the $n$th output channel, $\mathrm{PE}_m$: the processing element (PE) performs convolution on the $m$th input channel, $\mathrm{PU}_{m\_n}$: the PU performs convolution on the $n$th output channel in the $\mathrm{PE}_m$, and ACC: accumulator).

obtain

$$E\left(\mathbf{O}_b^2\right) = E((\vec{\mathbf{I}}\cdot\vec{\mathbf{W}})^2) + E((\vec{\mathbf{I}}_e\cdot\vec{\mathbf{W}})^2) + E((\vec{\mathbf{I}}\cdot\vec{\mathbf{W}}_e)^2)$$
$$= \frac{1}{K}\left(1 + \frac{\|\vec{\mathbf{I}}_e\|^2}{\|\vec{\mathbf{I}}\|^2} + \frac{\|\vec{\mathbf{W}}_e\|^2}{\|\vec{\mathbf{W}}\|^2}\right)\cdot\|\vec{\mathbf{I}}\|^2\cdot\|\vec{\mathbf{W}}\|^2 \quad (12)$$

where $(\|\vec{\mathbf{I}}_e\|^2/\|\vec{\mathbf{I}}\|^2)$ and $(\|\vec{\mathbf{W}}_e\|^2/\|\vec{\mathbf{W}}\|^2)$ are the noise-to-signal ratios (NSRs) of $\vec{\mathbf{I}}_e$ and $\vec{\mathbf{W}}_e$ and are denoted as $\eta_{\mathbf{I}}$ and $\eta_{\mathbf{W}}$, respectively. From (9) and (10), we have

$$\eta_{\mathbf{I}} = 10^{-\frac{\mathrm{SNR}_{\mathbf{I}}}{10}} \quad \text{and} \quad \eta_{\mathbf{W}} = 10^{-\frac{\mathrm{SNR}_{\mathbf{W}}}{10}}. \quad (13)$$

The NSR of the BFP formatted output matrix can be expressed as

$$\eta_{\mathbf{B}} = \frac{\sigma_B^2}{E((\vec{\mathbf{I}}\cdot\vec{\mathbf{W}})^2)} = \frac{E(\mathbf{O}_b^2) - E((\vec{\mathbf{I}}\cdot\vec{\mathbf{W}})^2)}{E((\vec{\mathbf{I}}\cdot\vec{\mathbf{W}})^2)} = \eta_{\mathbf{I}} + \eta_{\mathbf{W}}. \quad (14)$$

Thus, the average SNR of the BFP formatted output matrix is

$$\mathrm{SNR}_{\mathbf{B}} = -10\cdot\log_{10}\eta_{\mathbf{B}}$$
$$= -10\cdot\log_{10}\left(10^{-\frac{\mathrm{SNR}_{\mathbf{I}}}{10}} + 10^{-\frac{\mathrm{SNR}_{\mathbf{W}}}{10}}\right). \quad (15)$$

The quantization errors of BFP2FP conversion $\gamma$ have zero mean, and variance $\sigma_\gamma^2 = (\Delta/12)$. $\Delta$ is the minimum value interval represented by the FP format after conversion. The SNR of BFP2FP conversion is

$$\mathrm{SNR}_\gamma = 10\cdot\log_{10}\frac{E\left(\mathbf{O}_b^2\right)}{\sigma_\gamma^2}. \quad (16)$$

The relationship between $\eta_{\mathbf{B}}$ and $\eta_\gamma$ is

$$\eta_\gamma = \frac{\sigma_\gamma^2}{E\left(\mathbf{O}_b^2\right)} = \frac{\sigma_\gamma^2}{E((\vec{\mathbf{I}}\cdot\vec{\mathbf{W}})^2) + E((\vec{\mathbf{I}}\cdot\vec{\mathbf{W}})^2)\cdot\eta_{\mathbf{B}}}. \quad (17)$$

Thus, the average NSR of the current layer output feature map $\eta_{\mathbf{O}}$ is

$$\eta_{\mathbf{O}} = \frac{\sigma_B^2 + \sigma_\gamma^2}{E((\vec{\mathbf{I}}\cdot\vec{\mathbf{W}})^2)}$$
$$= \frac{\eta_{\mathbf{B}}E((\vec{\mathbf{I}}\cdot\vec{\mathbf{W}})^2) + \eta_\gamma\left(E((\vec{\mathbf{I}}\cdot\vec{\mathbf{W}})^2) + \eta_{\mathbf{B}}E((\vec{\mathbf{I}}\cdot\vec{\mathbf{W}})^2)\right)}{E((\vec{\mathbf{I}}\cdot\vec{\mathbf{W}})^2)}$$
$$= \eta_{\mathbf{B}} + \eta_\gamma + \eta_{\mathbf{B}}\eta_\gamma. \quad (18)$$

In the VGG-16 network, the result is processed by a ReLU layer after each convolutional layer. We assume that error is uniformly distributed in the entire output feature map space. Thus, the impact of the ReLU layer can be ignored. The output feature map of the upper layer carrying the error is used as the input feature map of the current layer; thus, we can analyze the multilayer propagation error.

Let $\eta_{\mathbf{O}}$ and $\eta_{\mathbf{I}}$ represent the NSR of the upper layer output feature map and the NSR of the FP2BFP conversion for the current layer input feature map, respectively. Similar to the analysis process described by (17) and (18), the overall NSR $\eta'_{\mathbf{I}}$ of the current layer input feature map is

$$\eta'_{\mathbf{I}} = \eta_{\mathbf{O}} + \eta_{\mathbf{I}} + \eta_{\mathbf{O}}\eta_{\mathbf{I}}. \quad (19)$$

The detailed error analysis and performance comparisons of our BFP arithmetic are described in Section IV.

## III. Hardware Implementation

In this paper, a reconfigurable architecture that supports a kernel size of $K_W \times K_H$ $(1 \le K_W, K_H \le 7)$ and an input image size of $14F_W \times 14F_H$ $(1 \le F_W, F_H \le 16)$ is proposed. Fig. 3 illustrates the high-level block diagram for

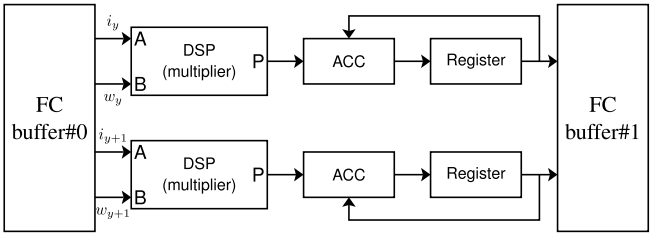



Fig. 5. Architecture of the FC PEA ($i_y$ and $w_y$: the input pixel and the weight of the $y$th output channel, respectively).

the proposed CNN accelerator. Our accelerator is composed of three primary components: a processing element array (PEA), an on-chip buffer, and external memory. In the initialization stage, the input images and network parameters are transferred from the host computer to the on-board DDR3 modules via PCIe3.0x8. To reduce the on-chip memory requirements, we apply the tile-grain convolution scheme (the tile size is $14\times14$).

The BFP arithmetic can significantly improve the performance of our accelerator by reducing the off-chip bandwidth requirements and the on-chip resources, and it introduces only a small accuracy loss. By analyzing the influence of mantissa length in the BFP arithmetic on classification accuracy (the detailed results are presented in Section IV-A), an 8-bit mantissa is applied in our FPGA accelerator implementation.

### A. PEA Architecture

According to the dataflow peculiarities of convolutional and fully connected (FC) layers, the image grain pipeline scheme [23] is adopted for the convolution PEA and FC PEA in this paper. When the FC layers of the current image are processed by the FC PEA, the convolution PEA processes the convolution computation of the next image. By analyzing the computational loads of the convolutional and FC layers, 16 (input channels)$\times$64 (output channels) BFP PUs are employed in the convolutional layer, while only two 16-bit FP multiply-accumulates (MACs) are adopted in the FC PEA.

*1) Convolution PEA Architecture:* The overall architecture of the convolution PEA is shown in Fig. 4(a). Sixteen PEs are devoted to their corresponding input channels' convolution. The input pixels $i_m$ and convolution kernels $K_m$ are fed into $PE_m$. The 64 PUs in one PE share the same input pixels, while they use the kernels of the corresponding output channels. The products of the 16 PUs($PU_{1\_n}$, $PU_{2\_n}$, ..., $PU_{16\_n}$) are summed in accumulator $A_n$, and the results are accumulated with the partial sum ($s_n$) of the former input channels. $PU_{m\_n}$ performs the calculations of the kernel $\vec{k}_{mn}$, and the architecture is shown in Fig. 4(b). The convolution in one PU employs a two-stage pipeline, including multiplication and accumulation (the register of the first stage is in the DSP). In each cycle, the pixels in the receptive field of the input feature map are input from Port A of the DSP in turn and are multiplied by the corresponding weights. The result of the multiplier is accumulated in the next cycle. The convolutional product of one filter is output after $(K_W \times K_H)$ cycles.

The multiplier is realized by the DSP48E1 Slice in VC709, and the max bit widths of the input ports are 25 and 18. In our

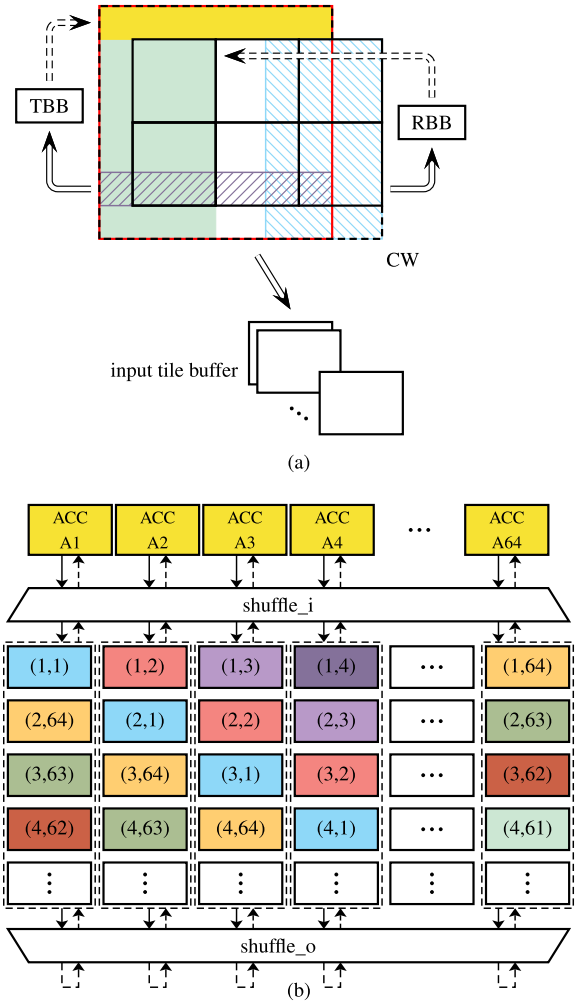



Fig. 6. Architecture of convolution buffers. (a) Layout of input buffer (TBB: top block buffer, CW: convolution window, RBB: right block buffer). (b) Layout of output buffer [the coordinate $(p, q)$ of the pixels in the BRAM: $p$ denotes the label of the pixel in a $7 \times 7$ block, $q$ indicates that this pixel is located in the $q$th output channel].

TABLE VIII
COMPUTATIONAL COMPLEXITY OF TYPICAL CNNS FOR CONV AND FC LAYERS

| Model | AlexNet | VGG-16 | GoogLeNet | ResNet |
|---|---|---|---|---|
| Conv layers (MACs) | 666M | 15.3G | 1.43G | 3.86G |
| FC layers (MACs) | 58.6M | 124M | 1M | 2M |

design, the bit length of the input pixels and weights is equal to 8, and thus two multiplication operations can be carried out in one DSP slice. Two pixels in the different receptive field of one input feature map are dispatched to one DSP, and the blank bits are used to isolate the two multiplication operations, as shown in Fig. 4(c). Every $(K_W \times K_H)$ cycles, two convolution results are produced by one PU, and the latency of one convolution round equals $14 \times 7 \times K_W \times K_H$ cycles.

The ReLU activation function and the max-pooling function are involved after completing the convolution of the last

TABLE IX
BFP QUANTIZATION RESULTS USING DIFFERENT CNN MODELS ($L_W$ AND $L_I$ REPRESENT THE BLOCK MANTISSA BIT LENGTH OF THE WEIGHT AND INPUT MATRICES, RESPECTIVELY)

| | | VGG-16 top-1 | | | | GoogLeNet top-1 | | | | ResNet-50 top-1 | | | | LeNet top-1 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $L_I$ | | | | $L_I$ | | | | $L_I$ | | | | $L_I$ | | |
| | | 6 | 7 | 8 | | 6 | 7 | 8 | | 6 | 7 | 8 | | 3 | 4 | 5 |
| $L_W$ | 6 | 2.02% | 0.60% | 0.38% | 6 | 0.89% | 0.51% | 0.45% | 6 | 6.83% | 3.05% | 2.38% | 3 | 0.57% | 0.13% | 0.13% |
| | 7 | 1.56% | 0.17% | 0.08% | 7 | 0.50% | 0.15% | 0.05% | 7 | 4.54% | 0.95% | 0.21% | 4 | 0.38% | 0.08% | 0.08% |
| | 8 | 1.48% | 0.15% | 0.03% | 8 | 0.45% | 0.09% | 0.07% | 8 | 4.38% | 0.76% | 0.11% | 5 | 0.30% | 0.12% | 0.00% |
| | | VGG-16 top-5 | | | | GoogLeNet top-5 | | | | ResNet-50 top-5 | | | | Cifar10 top-1 | | |
| | | $L_I$ | | | | $L_I$ | | | | $L_I$ | | | | $L_I$ | | |
| | | 6 | 7 | 8 | | 6 | 7 | 8 | | 6 | 7 | 8 | | 5 | 6 | 7 |
| $L_W$ | 6 | 1.35% | 0.50% | 0.36% | 6 | 0.62% | 0.42% | 0.36% | 6 | 4.74% | 1.85% | 1.31% | 5 | 1.46% | 0.87% | 0.86% |
| | 7 | 1.07% | 0.15% | 0.04% | 7 | 0.29% | 0.13% | 0.05% | 7 | 3.06% | 0.62% | 0.26% | 6 | 1.06% | 0.16% | 0.23% |
| | 8 | 1.05% | 0.13% | 0.02% | 8 | 0.22% | 0.12% | 0.06% | 8 | 3.01% | 0.54% | 0.12% | 7 | 0.91% | 0.00% | -0.04% |

input channel set. At the beginning of each convolution layer, the biases are input into the output buffer as the initial values of the partial sums. To ameliorate the loading latency during the convolutional procedure, a ping-pong access mode is applied for input tiles. The shuffle is used to rearrange the storage locations of partial sums in BRAM. A detailed introduction of our optimized memory system is provided in Section III-B.

*2) FC PEA Architecture:* As shown in Table VIII, the computation loads of the FC layers are less than 1% of the loads of the convolutional layers except for that of the smaller AlexNet. Therefore, only two PUs are allocated for the FC PEA. Taking the FC6 layer in the VGG-16 network as an example, the input feature map is a 25088-D vector, and the output feature map has 4096 dimensions. Thus, a weight filter of size 25088×4096 is needed to connect them. In our accelerator, each PU performs a vector multiplication of the corresponding output channel, as shown in Fig. 5. Every 25 088 cycles, the final results of two output channels can be generated.

### *B. Memory System*

After thoroughly considering the DRAM burst access properties, we define a $7 \times 7$ block as the basic storage cell. The off-chip DDR3 Module 0 stores the model parameters and input feature maps of odd convolutional layers, while the model parameters and input feature maps of even layers are stored in DDR3 Module 1. During the convolution procedure of one layer, one DRAM only performs the read operation, and the other maintains the write mode. This approach can eliminate the transition latency and power consumption between the write and read modes.

The on-chip buffer is composed of a convolution input buffer group, weight buffer, convolution output buffer, and FC buffer group. The convolution input buffer group consists of one convolution window (CW), one input tile buffer, one top block buffer (TBB), and one right block buffer (RBB), as shown in Fig.6(a).

CW is used to rearrange the input pixels from the DDR3 memory and later write the pixels into the input tile buffer. The rectangular enclosed by the red line will be stored in the BRAM of the corresponding input channel in the input tile buffer. A RBB and a TBB are used to cache the overlapped pixels of neighboring tiles, as shown in Fig. 6(a). The input pixels of the area filled with blue slashes are cached in RBB, and the pixels are written back to the green area from RBB when the CW slides to the right to a new tile. Similarly, TBB can write back the input pixels in the area filled with purple slashes to the yellow area when the CW slides down to a new tile. By adopting this method, the edge data can be read from the RBB and TBB without repeatedly accessing the external memory when processing the next tile.

The input tile buffer is composed of 16 160×40 bit BRAMs, and each BRAM stores the input tile of one input channel. The top half and bottom half of the BRAM realize the ping-pong access mode, and thus the data loading can be merged in the computation. After reading the input feature maps of an input channel set, all convolution operations related to these input feature maps are performed and generate as many intermediate results as possible. Each input feature map is read into the tile buffer only once, and it can reduce the data traffic with external memory.

The weight buffer consists of 16 512×2304 bit BRAMs and is used to store the weights of one layer. This approach can avoid cyclic weight reading from external memory. The convolution PEA will start after the first weight set has been read into the buffer; therefore, the PEA will not introduce high initial input latency.

The convolution output buffer consists of 64 1568×64 bit BRAMs, as shown in Fig.6(b). During the convolution procedure, the convolution output buffer is used to store the partial sums. The barrel shifting storage scheme is applied for the output buffer in this paper. Every time a new value is obtained from ACC, $shuffle\_i$ is used to shift the partial sum of the current output channel by one position and store the sum in another BRAM. $Shuffle\_o$ is used to rearrange the partial sums reading from the buffer. By adopting this method, the partial sums of different coordinates are stored in different BRAMs, and one 7×7 output block can be read out each cycle with true dual-port memories. The output buffer also adopts

TABLE X
SNR COMPARISONS BETWEEN OUR THEORETICAL ERROR ANALYSIS MODEL AND THE EXPERIMENTAL DATA FOR THE VGG-16 MODEL

| Layer | | theoretical SNR(dB) | experimental SNR(dB) | SNR change(dB) |
|---|---|---|---|---|
| conv1_1 | input | 41.8047 | 40.1767 | -1.6280 |
| | weight | 44.3538 | 44.1945 | -0.1593 |
| | output | 39.8824 | 37.1781 | -2.7043 |
| conv1_2 | input | 26.7226 | 28.8192 | 2.0966 |
| | weight | 37.3569 | 37.4007 | 0.0438 |
| | output | 26.3625 | 37.5498 | 11.1873 |
| conv2_1 | input | 28.8074 | 29.4291 | 0.6217 |
| | weight | 35.3470 | 35.3351 | -0.0119 |
| | output | 27.9372 | 31.7514 | 3.8133 |
| conv2_2 | input | 23.6998 | 26.2094 | 2.5096 |
| | weight | 34.9562 | 34.9732 | 0.0170 |
| | output | 23.3862 | 28.3326 | 4.9464 |
| conv3_1 | input | 25.7492 | 26.6441 | 0.8949 |
| | weight | 32.8990 | 32.8764 | -0.0226 |
| | output | 24.9836 | 28.3401 | 3.3565 |
| conv3_2 | input | 21.5140 | 24.5199 | 3.0059 |
| | weight | 32.1746 | 32.1773 | 0.0027 |
| | output | 21.1561 | 26.6088 | 5.4527 |
| conv3_3 | input | 19.4097 | 23.7736 | 4.3639 |
| | weight | 31.3544 | 31.3372 | -0.0172 |
| | output | 19.1406 | 25.4855 | 6.3449 |
| conv4_1 | input | 23.0923 | 24.4796 | 1.3873 |
| | weight | 32.5038 | 32.5263 | 0.0225 |
| | output | 22.6214 | 25.6961 | 3.0747 |
| conv4_2 | input | 20.1958 | 23.8786 | 3.6828 |
| | weight | 32.3566 | 32.3611 | 0.0045 |
| | output | 19.9395 | 24.6341 | 4.6946 |
| conv4_3 | input | 18.5201 | 23.4241 | 4.9040 |
| | weight | 31.6326 | 31.6401 | 0.0075 |
| | output | 18.3130 | 23.8303 | 5.5173 |
| conv5_1 | input | 21.5108 | 23.6883 | 2.1775 |
| | weight | 32.2420 | 32.2421 | 0.0001 |
| | output | 21.1584 | 24.8994 | 3.7410 |
| conv5_2 | input | 19.1991 | 23.7123 | 4.5132 |
| | weight | 33.9193 | 33.9279 | 0.0086 |
| | output | 19.0550 | 24.5182 | 5.4632 |
| conv5_3 | input | 17.8867 | 23.6408 | 5.7541 |
| | weight | 33.6540 | 33.6565 | 0.0025 |
| | output | 17.7731 | 23.2563 | 5.4832 |
| min output change(dB) | | | -2.7043 | |
| max output change(dB) | | | 11.1873 | |
| average output change(dB) | | | 4.6439 | |

the ping-pong access mode, and thus the data output can be merged in the convolution process. The above-optimized approaches can minimize the output latency.

The FC buffer group consists of two FC buffers, which are alternately set as the input and output buffers of one FC layer.

## IV. EXPERIMENTAL RESULTS

### A. Data Quantization Results

The proposed BFP arithmetic is conducted on the Caffe [24] scheme, and several typical deep convolution neural networks, including VGG-16 [5], GoogLeNet [2], and ResNet-50 [25], are used to evaluate our method. Furthermore, smaller network models, such as LeNet and Cifar10, are also tested. We only rewrite the convolution function of the inference model in Caffe according to the instructions shown in Fig. 2. The mantissa bit length is an important factor affecting the quantization result. Nine sets of experiments with different bit lengths of the weight and input mantissa are run, and 50 000 images in the ImageNet classification data set [26] are used to test the performance of the proposed BFP arithmetic, as shown in Table IX. The 32-bit FP result is set as the baseline.

$L_W$ and $L_I$ denote the bit lengths of the BFP formatted weight and input mantissa (including the sign bit), respectively. For typical CNN models (VGG-16, GoogLeNet, and ResNet-50), when $L_W$ and $L_I$ are equal to 8, the classification accuracy loss is less than 0.12%. For GoogLeNet, in particular, the 6-bit mantissa can also maintain high performance with top-1 and top-5 accuracy losses of only 0.89% and 0.62%, respectively. In addition, the 4-bit and 6-bit mantissas can achieve accuracy losses of only 0.08% and 0.16% for LeNet [28] and Cifar10 [29], respectively. As the mantissa length decreases, both the data traffic and hardware cost of the accelerator will be significantly reduced. For example, savings of more than 50% for computing resources and 42.8% for memory resources can be achieved for an 8-bit mantissa compared with a 16-bit mantissa. Therefore, we choose the 8-bit mantissa for our hardware implementation in this paper. Furthermore, as our BFP arithmetic does not require retraining and fine-tuning, the time cost of development will be reduced. The experiments show that our BFP arithmetic can apply to most CNN models with only negligible accuracy and time cost.

The error analysis model is tested in the VGG-16 network on the Caffe scheme, and the SNR comparisons between our theoretical error analysis model and the experimental data are shown in Table X. The max-pooling layer is used in the VGG-16 network, which will improve the overall SNR. However, the error propagation of the max-pooling layer is difficult to fit with mathematical models. Thus, we use the output SNR of the pooling layer as the input SNR of the next layer in this paper. The results show that the average deviation is 4.64 dB, and the theoretical data are in good agreement with the experimental results. Our error analysis model is sufficiently close to guide hardware design.

TABLE XI
RESOURCE UTILIZATION IN XC7VX690T

| Resource | LUT | FF | BRAMs | DSP | PCIe |
|---|---|---|---|---|---|
| Used | 231761 | 140971 | 913 | 1027 | 1 |
| Available | 433200 | 866400 | 1470 | 3600 | 3 |
| Utilization | 53.5% | 16.3% | 62.1% | 28.5% | 33.3% |

### B. Hardware Performance

The hardware implementation is described in Verilog-HDL with the Xilinx Vivado 2017. 4 Design Suite environment. The platform is the VC709 evaluation board, which consists of a Xilinx Virtex-7 XC7VX690T FPGA and two 4-GB

TABLE XII
COMPARISONS WITH OTHER FPGA ACCELERATORS

| | [8] | [23] | [33] | [9] | [32] | [34] | Ours |
|---|---|---|---|---|---|---|---|
| Year | 2015 | 2017 | 2017 | 2018 | 2018 | 2018 | 2018 |
| Platform | Zynq XC7Z045 | Virtex7 VX690T | Zynq XC7Z045 | Zynq XC7Z020 | Arria10 GT1150 | Zynq XC7Z045 | Virtex7 VX690T |
| Network | VGG-16 | VGG-16 | VGG-16 | VGG-16 | VGG-16 | VGG-16 | VGG-16 |
| Clock(MHz) | 150 | 200 | 100 | 214 | 231 | 200 | 200 |
| Quantization Strategy | 16-bit fixed | 16-bit floating | 16-bit fixed | 8-bit fixed | 8/16-bit fixed | 8-bit fixed | 8-bit BFP |
| Power(W) | 9.63 | 10.81 | 9.4 | 3.5 | N/A | 7.2 | 9.18 |
| CNN Size(GOP) | 30.76 | 30.76 | 30.76 | 30.76 | 30.76 | 9.45(pruned) | 30.76 |
| Performance(GOP/s) | 136.97 | 202.42 | 229.6 | 84.3 | 715.9 | 524.0 | 760.83 |
| Power Efficiency (GOP/s/W) | 14.22 | 18.72 | 24.42 | 24.09 | N/A | 72.8 | 82.88 |
| DSP Used | 780 | 1728 | 824 | 780 | 1500 | 680 | 1027 |
| DSP Efficiency (GOP/s/DSP) | 0.176 | 0.117 | 0.279 | 0.444 | 0.472 | 0.771 | 0.741 |

DDR3 memory modules. VGG-16 is deployed on the accelerator to analyze the hardware performance. Two hundred consecutive images are tested on the accelerator to measure the processing latency. We use a Hyelec HY-001 digital power meter to measure the power consumption.

The resource utilization of our accelerator is exhibited in Table XI, and the proposed accelerator is compared with previously reported accelerators in Table XII.

The earliest CNN accelerator adopted the 32-bit FP format [7], and it only achieved a performance of 61.62 GOP/s because of the high hardware costs. All the following accelerators apply a bit length reduction mechanism. The work in [23] proposed a 16-bit FP CNN accelerator based on the ping-pong memory access mode and propagate partial MAC (PPMAC) processor. The computational performance of the accelerator is 202.42 GOP/s under 200-MHz working frequency, but the low hardware efficiency of the FP arithmetic degrades the DSP efficiency to only 0.117 GOP/s/DSP.

Fixed-point operations have been used instead of FP operations in recent studies. Qiu *et al.* [8] proposed an embedded FPGA accelerator that consisted of efficient dynamic precision data quantization and data arrangement methods to improve the hardware utilization. The accelerator achieved a performance of 136.97 GOP/s under a 150-MHz working frequency. A fusion architecture consisting of a faster algorithm using Winograd's minimal filtering theory was proposed in [33]. An automated toolchain was developed to convert the Caffe model to the FPGA bitstream, and it achieved a performance of 229.6 GOP/s.

To improve the hardware utilization, researchers began to explore the further reduction of data bit length. A programmable CNN accelerator architecture with a data quantization strategy was proposed in [9]. The accelerator can reduce the bit length of input feature maps and weights down to 8 bits with an accuracy loss of less than 6%. The accelerator achieved a power efficiency of 24.09 GOP/s/W and a DSP efficiency of 0.444 GOP/s/DSP. Through an in-depth analysis of the convolution loop optimization techniques, a new dataflow and hardware architecture was proposed to minimize the data traffic, and it achieved a performance of 715.9 GOP/s [32]. However, the classification accuracy loss of this method still reached 2%. An automated tool for building high-performance DNN hardware accelerator was proposed in [34]. The accelerator achieved a high performance of 72.8 GOP/s/W and 0.771 GOP/s/DSP. However, the top-1 accuracy loss of this method on the VGG-16 model was 4.6% without retraining and fine-tuning.

To further improve the hardware efficiency while maintaining the classification accuracy, we deploy the proposed BFP arithmetic on the FPGA platform. With the proposed three-level parallel convolution engine and optimized storage scheme, our accelerator achieves a high performance of 760.83 GOP/s, which is at least 6.28% faster than that of the previous accelerators for the VGG-16 model. Because of the BFP arithmetic, the power consumption is 9.18 W, which saves 15% energy consumption compared to the previous accelerator on the same Virtex7 VX690T platform [23]. The power efficiency of our work is 82.88 GOP/s/W, which is at least 13.8% higher than that of previous accelerators. By deploying two multiplication operations on one DSP and the data prereading mechanism, the DSP efficiency is 0.741 GOP/s/DSP.

## V. CONCLUSION

A block-FP arithmetic-based CNN accelerator is proposed in this paper to reduce the hardware costs and data traffic of CNN models. The accuracy loss of our method on different CNN models is less than 0.12% without retraining. An optimized rounding and zero setting algorithm maintains the computational accuracy while reducing the mantissa bit length efficiently. The FPGA-based CNN accelerator is deployed on the Xilinx VC709 evaluation board. Leveraging three-level parallel convolution engines, a ping-pong memory access mode, and an optimized on-chip buffer scheme, the proposed

architecture achieves a throughput of 760.83 GOP/s and a power efficiency of 82.88 GOP/s/W under 200-MHz working frequency, significantly outperforming previous architectures.

## REFERENCES


[1] A. Krizhevsky, I. Sutskever, and G.-E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2012, pp. 1097–1105.

[2] C. Szegedy *et al.*, "Going deeper with convolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[3] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.

[4] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun. (2016). "Very deep convolutional networks for text classification." [Online]. Available: https://arxiv.org/abs/1606.01781

[5] K. Simonyan and A. Zisserman. (2014). "Very deep convolutional networks for large-scale image recognition." [Online]. Available: https://arxiv.org/abs/1409.1556

[6] N. Corporation. *NVIDIA cuDNN GPU Accelerated Deep Learning*. Accessed: Feb. 2019. [Online]. Available: https://developer.nvidia.com/cudnn

[7] C. Zhang *et al.*, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2015, pp. 161–170.

[8] J. Qiu *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, pp. 26–35, 2016.

[9] K. Guo *et al.*, "Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2018.

[10] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance FPGA-based accelerator for large-scale convolutional neural networks," in *Proc. 26th Int. Conf. Field Program. Logic Appl.*, Aug./Sep. 2016, pp. 1–9.

[11] H. Sharma *et al.*, "From high-level deep neural models to FPGAs," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2016, pp. 1–12.

[12] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[13] V. Gokhale, A. Zaidy, A. X. M. Chang, and E. Culurciello, "Snowflake: An efficient hardware accelerator for convolutional neural networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[14] A. Aimar *et al.*, "NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 3, pp. 644–656, Mar. 2019.

[15] S. Zhang *et al.*, "Cambricon-X: An accelerator for sparse neural networks," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchitecture (Micro)*, Oct. 2016, pp. 1–12.

[16] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 8, pp. 2220–2233, Aug. 2017.

[17] S. Han, H. Mao, and W.-J. Dally. (2015). "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding." [Online]. Available: https://arxiv.org/abs/1510.00149

[18] S. Gupta *et al.*, "Deep learning with limited numerical precision," in *Proc. 32nd Int. Conf. Mach. Learn.(ICML)*, Jun. 2015, pp. 1737–1746.

[19] S. Migacz. *8-Bit Inference With TensorRT*. Accessed: May 2017. [Online]. Available: https://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf

[20] U. Köster *et al.*, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 1742–1752.

[21] Z. Song, Z. Liu, and D. Wang, "Computation error analysis of block floating point arithmetic oriented convolution neural network accelerator design," in *Proc. 30th AAAI Conf. Artif. Intell.*, Apr. 2018, pp. 816–823.

[22] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis. (IJCV)*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[23] C. Mei, Z. Liu, Y. Niu, X. Ji, W. Zhou, and D. Wang, "A 200MHZ 202.4GFLOPS@10.8W VGG16 accelerator in Xilinx VX690T," in *Proc. IEEE Global Conf. Signal Inf. Process. (GlobalSIP)*, Nov. 2017, pp. 784–788.

[24] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, Nov. 2014, pp. 675–678.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[26] *Imagenet*. Accessed: Aug. 2016. [Online]. Available: https://www.image-net.org

[27] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, Feb. 2014, pp. 10–14.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[29] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep. 1(4):7, 2009.

[30] K. Kalliojarvi and J. Astola, "Roundoff errors in block-floating-point systems," *IEEE Trans. Signal Process.*, vol. 44, no. 4, pp. 783–790, Apr. 1996.

[31] N. Corporation. *CUDA Toolkit Documentation: Floating Point and IEEE 754*. Accessed: Mar. 2019. [Online]. Available: https://docs.nvidia.com/cuda/floating-point/

[32] Y. Ma, T. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 7, pp. 1354–1367, Jul. 2018.

[33] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y.-W. Tai, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs," in *Proc. 54th Ann. Design Autom. Conf. (DAC)*, Jun. 2017, pp. 1–6.

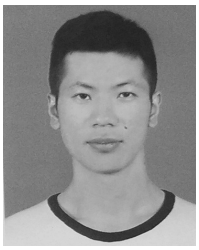
[34] X. Zhang *et al.*, "DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2018, pp. 1–8.


**Xiaocong Lian** (S'15–M'17) received the B.S. and Ph.D. degrees in electronic science and technology from Northwestern Polytechnical University, Xi'an, China, in 2013 and 2017, respectively.

He is currently a Post-Doctoral Researcher at the Automation Department, Tsinghua University, Beijing, China. His current research interests include algorithms and VLSI implementation for video coding and hardware acceleration of neural network.

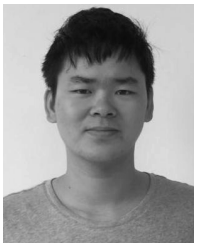
**Zhenyu Liu** (M'07) received the B.E., M.E., and Ph.D. degrees in electrical engineering from the Beijing Institute of Technology, Beijing, China, in 1996, 1999, and 2002, respectively.

From 2002 to 2004, he held a post-doctoral position at Tsinghua University, Beijing, where he was involved in the embedded processor architecture design. From 2004 to 2009, he was a Visiting Researcher at the Graduate School of IPS, Waseda University, Tokyo, Japan. In 2009, he joined Tsinghua University, where he is currently an Associate Professor at RIIT&TNList. His current research interests include signal processing, energy-efficient real-time video encoding, and application-specific processor.



**Zhourui Song** received the B.S. degree in electronic science and technology from the Beijing University of Posts and Telecommunications, Beijing, China, in 2016, where he is currently working toward the M.S. degree of computer science.

His current research interests include computer vision and simultaneous location and mapping.



**Jiwu Dai** received the B.S. degree in electronic science and technology from Northwestern Polytechnical University, Xi'an, China, in 2017, where he is currently working toward the master's degree.

His current research interests include deep learning.

**Wei Zhou** (M'11) received the B.E., M.S., and Ph.D. degrees from Northwestern Polytechnical University, Xi'an, China, in 2001, 2004 and 2007, respectively.

He is currently a Professor at Northwestern Polytechnical University. His current research interests include video coding and associated VLSI architecture design, intelligent visual computing.

**Xiangyang Ji** (M'10) received the B.S. degree in materials science and the M.S. degree in computer science from the Harbin Institute of Technology, Harbin, China, in 1999 and 2001, respectively, and the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.

In 2008, he joined Tsinghua University, Beijing, where he is currently a Professor at the Department of Automation, School of Information Science and Technology. He has authored more than 100 refereed conference and journal papers. His current research interests include signal processing, image/video compression and communication, and intelligent imaging.