Low Bit-Width Convolutional Neural Network on RRAM

Yi Cai, Tianqi Tang, Lixue Xia, *Student Member, IEEE*, Boxun Li, *Student Member, IEEE*, Yu Wang, *Senior Member, IEEE*, and Huazhong Yang, *Senior Member, IEEE*

***Abstract*—The emerging resistive random-access memory (RRAM) has been widely applied in accelerating the computing of deep neural networks. However, it is challenging to achieve high-precision computations based on RRAM due to the limits of the resistance level and the interfaces. Low bit-width convolutional neural networks (CNNs) provide promising solutions to introduce low bit-width RRAM devices and low bit-width interfaces in RRAM-based computing system (RCS). While open questions still remain regarding: 1) how to make matrix splitting when a single crossbar is not large enough to hold all parameters of one weight matrix; 2) how to design a pipeline to accelerate the inference based on line buffer structure; and 3) how to reduce the accuracy drop due to the parameter splitting and data quantization. In this paper, we propose an RRAM crossbar-based low bit-width CNN (LB-CNN) accelerator. We make detailed discussion on the system design, including the matrix splitting strategies to enhance the scalability, and the pipelined implementation based on line buffers to accelerate the inference. In addition, we propose a splitting and quantizing while training method to incorporate the actual hardware constraints with the training. In our experiments, low bit-width LeNet-5 on RRAM show much better robustness than multibit models with device variation. The pipeline strategy achieves approximately 6.0× speedup to process each image on ResNet-18. For low-bit VGG-8 on CIFAR-10, the proposed accelerator saves 54.9% of the energy consumption and 48.3% of the area compared with the multibit VGG-8 structure.**

***Index Terms*—Constrained training, low bit-width convolutional neural network (LB-CNN), parameter splitting, pipeline, resistive random-access memory (RRAM).**

I. Introduction

CONVOLUTIONAL neural networks (CNNs) have achieved great performance in various computer vision applications [1]–[3]. With the CNNs going deeper, the computing of CNNs has increased the demand on the communication bandwidth and hardware resources. Thus, efficiency is now perceived to be a growing problem in neural network computing systems. Many studies have made great efforts on accelerating the neural network computing [4]–[6]. However, conventional CMOS- and von Neumann architecture-based processors are facing the *memory wall*, and Moore's Law is dying [7], which impedes further improvement of speed and energy efficiency. Therefore, many researchers have turned attention to the novel processing-in-memory (PIM) architectures.

The emerging resistive random-access memory (RRAM)-based computing system (RCS) has been proven to be one of the most promising candidates for future CNN accelerators [8]–[10], resulting from the outstanding attributes of RRAM, such as high density, low power, etc. Moreover, the RRAM crossbars not only can store the weight parameters of CNN models, but also can implement the matrix-vector multiplications (MVMs) located in memory [8], [10]–[12], which is the so called PIM pattern. By combining RRAM crossbars with CNN computing systems, the data movements can be substantially reduced, and less bandwidth is required. The time complexity of the MVM (with $n \times n$ matrix) can be reduced from $O(n^2)$ to $O(1)$, resulting from the crossbar-level parallelism.

However, the RRAM-based CNN accelerators are not adept for high-precision computing. On the one hand, the resistance level of the RRAM device cannot support the available storage of high-precision data [13]. Furthermore, as the bit level increases, the impact of memory writing and reading variation will be greatly enlarged because the margin of two adjacent resistance states becomes quite smaller [14]. On the other hand, because RRAM crossbar performs the MVMs in analog mode, additional interfaces, including analog-to-digital converters (ADCs) and digital-to-analog converters (DACs), are required to implement the data conversions. High-precision ADC/DACs consume considerable area and power, which makes the entire RCS far less efficient than expected [11]. Therefore, the precision limit in the RCS has become the most challenging issue of RRAM-based CNN accelerators.

Researchers have proposed methods to train low bit-width CNNs (LB-CNNs), even with binary weights and activations. Although the performance of binary CNNs always drops rapidly, e.g., over a 10% accuracy drop in

Manuscript received September 13, 2018; revised March 4, 2019; accepted April 28, 2019. Date of publication May 20, 2020; date of current version June 18, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFA0207600, in part by the National Natural Science Foundation of China under Grant 61832007, Grant 61622403, and Grant 61621091, and in part by the Beijing National Research Center for Information Science and Technology. This paper was recommended by Associate Editor L. Benini. *(Yi Cai and Tianqi Tang contributed equally to this work.) (Corresponding author: Yu Wang.)*

Y. Cai, B. Li, Y. Wang, and H. Yang are with the Department of Electronic Engineering, Beijing National Research Center for Information Science and Technology, Beijing Innovation Center for Future Chips, Tsinghua University, Beijing 100084, China (e-mail: yu-wang@tsinghua.edu.cn).

T. Tang is with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA 93106 USA.

L. Xia is with the Department of Cloud Intelligence, Alibaba Group at Beijing, Beijing 100022, China.

Digital Object Identifier 10.1109/TCAD.2019.2917852

ImageNet classification was observed in [15] and [16], low-bit approaches have demonstrated impressive performance. For example, INQ [17] outperforms the floating-point baseline with the quantization of both the weights and activation to 5 bits; DoReFa-Net [18] also achieves comparable performance with full-precision networks with only 1-bit weight and 4-bit activation. Therefore, it is promising to overcome the precision limits by reducing the bit-width of the data, including all the weights and activations.

More challenges also appear when the scale of the CNN increases. The RRAM crossbar can only be manufactured with a limited size due to the IR-drop problem and other nonideal factors. Therefore, a single crossbar may not be able to hold the whole weight matrix of a large-scale convolution or a fully connected (FC) layer. To address this, matrix splitting strategies are required. While high-precision interfaces are required to obtain partial intermediate activation (IA) and maintain the computing precision. Then, the partial IA are accumulated to get the merged activation (MA). To save additional energy, it is also promising to introduce low bit-width interfaces in the partial intermediate data conversions. However, these interfaces will introduce more quantization error and cause decreasing accuracy.

Meanwhile, a larger time consumption and energy overhead will be introduced by the rapid growth of intermediate data buffers between convolutional layers. Because sliding operations exist to convolve the features, only a few registers are used in each cycle, so there exists a parallel potential between layers to reduce the buffer size while boosting the processing speed. As a result, a pipeline strategy between layers can be designed for both CNN and LB-CNN accelerators.

In this paper, we propose an RRAM crossbar-based LB-CNN accelerator. The main contributions are as follows.

1) An efficient structure is designed for the accelerator system, including the computing circuits and buffers. A detailed discussion on the parameter splitting is given.
2) We propose a pipeline strategy to reduce the processing cycles. Our experiments show that the pipelined implementation can achieve 6× speedup when processing the ResNet-18 compared with layer-by-layer implementation.
3) We introduce the splitting and quantizing during training method to incorporate the hardware limitations with the training, which achieves comparable performance with floating-point, nonsplitting models.
4) We demonstrated the robustness with device variation of the LB-CNNs on RRAM. Our experiments of LeNet over the MNIST dataset validate that, binary CNN achieves 0.75% error rate on 3-bit RRAM devices when considering device variation.
5) We conduct simulations to evaluate the performance of our design. Through introducing low-bitwidth interfaces, more than 60% of the interface area and power can be saved. Overall, when constructing the accelerator for VGG-8, 54.9% of the energy overhead, and 48.3% of the area consumption are saved.

II. Preliminaries

A. CNN Fundamentals

Typically, a standard CNN is constructed by a number of convolutional (Conv) layers and FC layers that run sequentially. Conv layers are usually optionally followed by nonlinear neuron layers, pooling layers, and normalization layers [1].

1) Conv Layer: The mathematical function of the Conv layer can be expressed as in

$$\vec{g}(x,y,z)=\sum_{i=0}^{h-1}\sum_{j=0}^{w-1}\sum_{k=0}^{C_{\text{in}}-1}\vec{f}(x+i,y+j,k)\cdot\vec{c}_z(i,j,k) \quad (1)$$

where the vector $\vec{f}$ represents the 3-D input feature map with the size of $H_{\text{in}}\times W_{\text{in}}\times C_{\text{in}}$; the vector $\vec{g}$ represents the 3-D output feature map with the size of $H_{\text{out}}\times W_{\text{out}}\times C_{\text{out}}$; the vector $\vec{c}_z$ is the zth convolution kernel with the size of $h\times w\times C_{\text{in}}$; C_{out} is the number of convolution kernels; and the remaining variables are all spatial coordinates of the feature maps and convolution kernels. In this way, a 4-D blob shaped as $H_{\text{out}}\times W_{\text{out}}\times C_{\text{in}}\times C_{\text{out}}$ corresponds to a Conv layer.

2) Neuron Layer: This layer is a one-by-one mapping function that is attached after the Conv layer ($y=f(x)$). In our LB-CNN design, the nonlinear neuron layer is set as the commonly used ReLU function. When quantizing the activations to 1 bit, binary neurons, as proposed in BinaryNet [15], are used. The forward function can be expressed in

$$y=\begin{cases}1, & x>0\\ -1, & x\le 0.\end{cases} \quad (2)$$

3) Max Pooling Layer: This layer is cascaded after the nonlinear neuron layer, which is a form of nonlinear down-sampling. Max pooling partitions the input feature map into rectangular regions and picks the maximum value of each region as the corresponding element of the output feature map to reduce the computation for the upper layers and maintain the local invariance.

4) FC Layer: This layer is the final layer in which all inputs and outputs are connected by weights as traditional neural networks. The operation can be represented as in

$$f_{\text{out}}(y)=\sum_{x=0}^{L_{\text{in}}-1}f_{\text{in}}(x)\cdot c(x,y) \quad (3)$$

where x is the index of the 1-D input feature map vector f_{in} with the length of L_{in}; y is the index of output feature map vector f_{out} with the length of L_{out}; and the 2-D matrix $\mathbf{c}$ is the weights with the size of $L_{\text{in}}\times L_{\text{out}}$.

B. RRAM Device, Crossbar Array, and Interface

An RRAM device is a passive two-port nonvolatile memory element whose resistance can be tuned within a certain range. Therefore, a multibit number can be represented by dividing the resistance range into multiple intervals. Moreover, multiple RRAM devices can be used to build the crossbar structure. If the weights are stored by the conductance of the RRAM devices and the data are represented by the input voltage signals, then the RRAM crossbar is able to perform as an analog convolution processing unit. When applying voltages in the

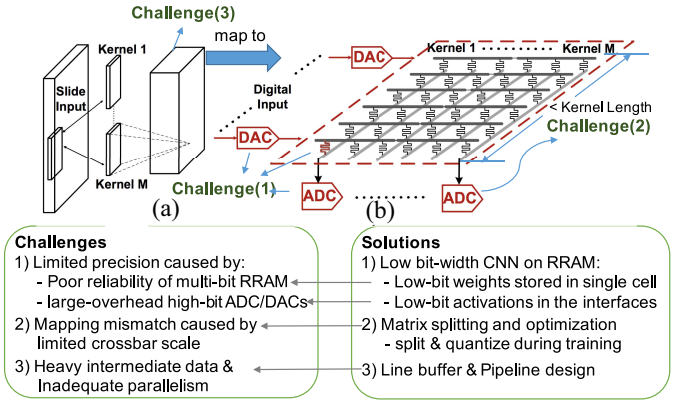


Fig. 1. Challenges and proposed solutions of RRAM-based neural network computing. This figure also shows (a) CNN structure. (b) Structure of the RRAM-based crossbar.

input interface, accumulated currents can be fetched on the output port. Specifically, the relationship between the input voltages and output currents can be expressed as in (4) [12]

$$i_{\text{out}}(k) = \sum_{j=0}^{N-1} g(k,j) \cdot v_{\text{in}}(j) \tag{4}$$

where $\vec{v}_{\text{in}}$ is the vector of input voltage (denoted by $j = 0, 1, \ldots, N-1$), $\vec{i}_{\text{out}}$ is the vector output current (denoted by $k = 0, 1, \ldots, M-1$), and g is the conductivity matrix of the RRAM device which represents the weights. Commonly in the input interfaces, DACs are required to convert the digital inputs into the voltages with multilevel amplitudes. In the output interface, sensing amplifiers (SAs) or ADCs are required to extract the calculation results, as shown in Fig. 1. Since the equation shows the same pattern as MVMs, the RRAM crossbars can implement MVMs with high parallelism and efficiency.

As the dominant computations of Conv layers and FC layers are composed of MVMs, the crossbar can be utilized to implement the Conv and FC operations in analog mode with high speed, small area, and low power [12]. For FC layers, the weight matrices are directly mapped to the RRAM crossbars [11]. As for Conv layers, the parameter tensors with the shape of $(H_{\text{out}}, W_{\text{out}}, C_{\text{in}}, C_{\text{out}})$ are reshaped to 2-D $(H_{\text{out}} \times W_{\text{out}} \times C_{\text{in}}, C_{\text{out}})$ matrices, then mapped to the RRAM crossbars [12], as shown in Fig. 1.

III. Motivation

The RRAM-based systems have shown great potential to achieve incredible performance regarding CNN computation. However, the RRAM device, crossbar manufacturing, and peripheral interfaces have some limits, which derives a number of obstacles to RRAM-based CNN implementation and makes the efficiency far less appealing than expected.

A. Limited Bit Level of RRAM Device

To the best of our knowledge, the state-of-the-art RRAM devices [13] can at most represent 7-bit values due to the limited conductance range and device variation. The situation will become even worse when constructing the crossbar structure. Thus, the quantized CNNs with 8-bit fixed-point weights cannot be deployed directly. To tackle the precision problem, previous work utilized multiple RRAM devices to represent a single weight [8], [10], accompanied with much more times of energy overhead. Moreover, multibit devices suffer from more variation and reliability problems than single-bit devices [14]. Therefore, CNNs with low bit-width weights are preferred to ensure the reliability and efficiency.

TABLE I
Performance Table of State-of-the-Art ADCs

	Resolution (bit)	Power (mW)	Sampling Freq. (Hz)	Area (mm^2)	E./Samp. (μJ/S)
[19]	19	0.278	1K	0.25	0.278
[20]	15	0.0045	1K	0.053	0.053
[21]	13	0.084	625K	0.024	1.34E-4
[22]	13	64.3	50M	0.25	1.29E-3
[23]	8	0.00005	5K	0.0039	1.0E-5
[24]	8	97.0	48G	0.15	2.02E-6

B. Limited Bit Level of Crossbar Interfaces

Currently, it is unavailable to directly operate on the analog output signals, and stably store and send the analog data to the next array for the follow-up calculations. Thus, the peripheral digital circuit and memories are required to implement the processing, buffering, and transmission of the intermediate results. In this scenario, interfaces are needed for the conversion between the digital signals in the peripheral digital units and the analog signals in the crossbar. A practical choice is to use ADC/DACs as the interfaces to convert from and to the digital signals, while large overheads are introduced by the high-precision ADC/DACs. Li *et al.* [11] noted that only 8-bit ADC/DACs contribute to more than 85% of the area and power consumption of the entire RCS. Previous work [9] has proposed to use spike-based input and output to eliminate the ADC/DACs. However, the drawback of such design is requiring much more cycles. We investigate the performance of recently reported ADCs as shown in Table I. It can be seen that the power of ADCs explosively increases with the sampling frequency and resolution. Therefore, it will contribute significantly to the energy efficiency if achieving a well-trained network model with lowest possible bit-width activation.

C. Limited Manufacturing Scale of the Crossbar

Reliable RRAM crossbars are usually manufactured with a limited size, generally up to 1024×1024. However, taking VGG [1] as an example, the Conv layer usually has 512 convolutional kernels, all of which are $3 \times 3 \times 512$, and will be reshaped to 4096×512 matrices. To successfully map the Conv layers onto the RRAM crossbars, the scale of parameter matrices must be smaller than, or at most equal to, that of the crossbar size. In this scenario, the crossbar size should also approach 4096×512 at least, which is obviously unmanufacturable under current technology. Previous work [8], [10] has proposed to handle the mapping problem by parameter matrix splitting.

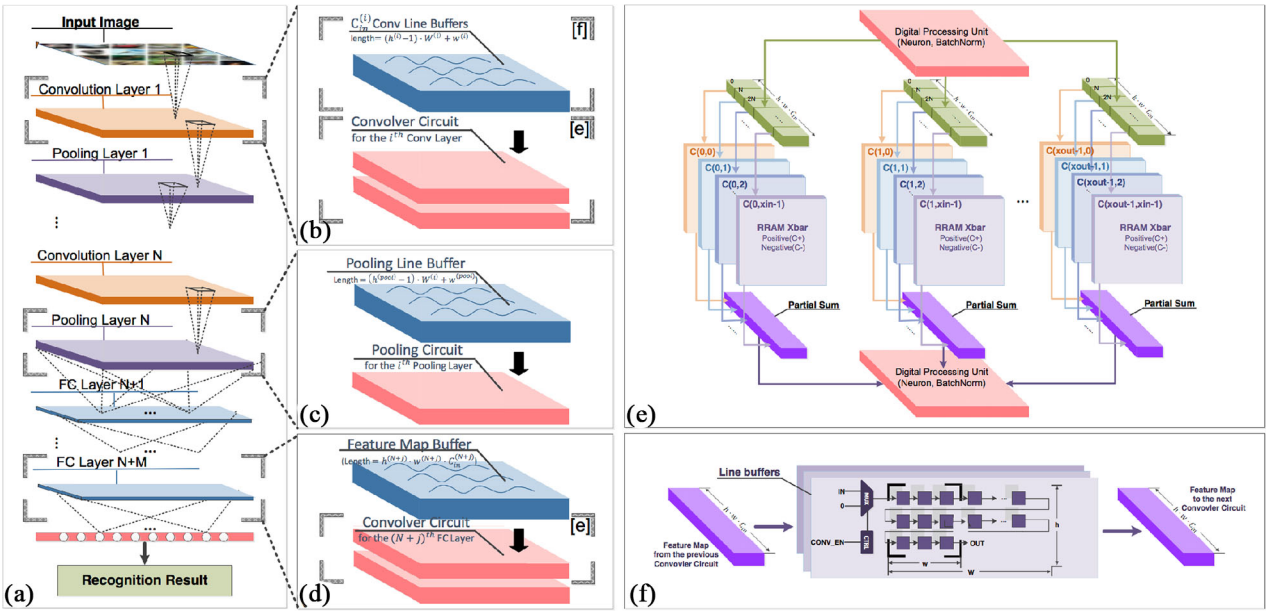


Fig. 2. (a) Overall structure of the RRAM-based BCNN accelerator: "N Conv layers + M FC layers" with the input image and the output recognition result, and each Conv layer optionally followed by the pooling layer. (b)–(d) Dataflow of the "Conv layer," "pooling layer," and "FC layer." (e) Convolver circuit for one Conv/FC layer on RRAM-based platform. (f) Conv line buffers. Each Conv layer requires $((h-1)\times(W+p)+w)\times C_{\text{in}}$ registers.

The matrix splitting is equivalent to dividing the original MVM into multiple sub-MVM blocks, then summing all the intermediate results. Mathematically, high-precision ADCs are needed to convert the partial IA to keep the final quantized results consistent with the expectation. Otherwise, large truncation errors will be introduced, and then the recognition accuracy of CNNs will substantially decrease. However, since the row-wise split block number is equal to the multiple of the ADCs to be increased, the power consumption introduced by matrix splitting will be unacceptable if high-precision ADCs are used. Therefore, it is of great importance to incorporate the splitting and quantization with training, to eliminate the mismatch of hardware constraints and the CNN models, so that the high-cost interfaces can be removed without causing the accuracy drop.

IV. RRAM-Based LB-CNN Accelerator Design

We propose an RRAM-based LB-CNN accelerator design. In this section, we first introduce the overall structure of processing circuits and buffer, including the computing circuit and line buffer structure. Then we discuss the problem of matrix splitting. In the final, we also propose a pipeline approach to accelerate the inference computing.

A. Computing Circuits and Buffer Design

Fig. 2(a) shows the overview of the LB-CNN accelerator structure. The entire accelerator is made up of a series of processing elements (PEs). Each PE consists of the computing circuit and buffers. The computing circuits contain two main parts: 1) the RRAM-based convolver circuit and 2) additional digital circuit. The digital computing circuit is also integrated to process the operations other than MVMs, including the element-wise adds, the (max) pooling layer, and the ReLU layer. The line buffers are set to buffer the feature maps, which are constructed by SRAM-based scratchpad memory. Besides, The data paths for the Conv, pooling, and FC layer are, respectively shown in Fig. 2(b)–(d).

1) Computing Circuit: As stated before, each PE contains a convolver circuit and a digital computing circuit. For Conv and FC layers, the *convolver circuit* is mainly constructed by the RRAM crossbar-based MVM modules, as shown in Fig. 2(e). For the pooling layer, we use the max pooling function for the downsampling operations. The pooling operations are performed on a pure digital circuit. If the activations before pooling operations are binarized, then the circuit of the pooling layer can be simply implemented by the multi-input OR gate in our design. As long as a partial rectangle contains at least an "1," the output result of pooling will also be "1," otherwise the output result will be "0." For other low-bit cases, the look-up-tables (LUTs) will be used to construct cascaded comparators to finish the pooling operations. We only use the typical implementation of LUTs because the pooling operations are not the bottleneck to improving system performance. For k-bit inputs, $(2^{2k}\times k)$-bit SRAMs are utilized to construct an LUT for comparing two inputs. For the batch normalization (BN) layer, since the function of BN is a kind of linear transform, it can be merged into the former Conv layer when processing the inference. For the ReLU layer, we put multiplexers to select the outputs from *zero* and original inputs according to the sign bits.

2) Input Buffer: In the computing process of the Conv and the pooling layers, a sliding window is applied to separate the connections of the different pixels in the feature maps, with a sliding stride s and zero padding being made at the edge. The result of a sliding window will not be computed until all the elements inside the window are fetched. For this reason, the structure of the *line buffer* is introduced for intermediate

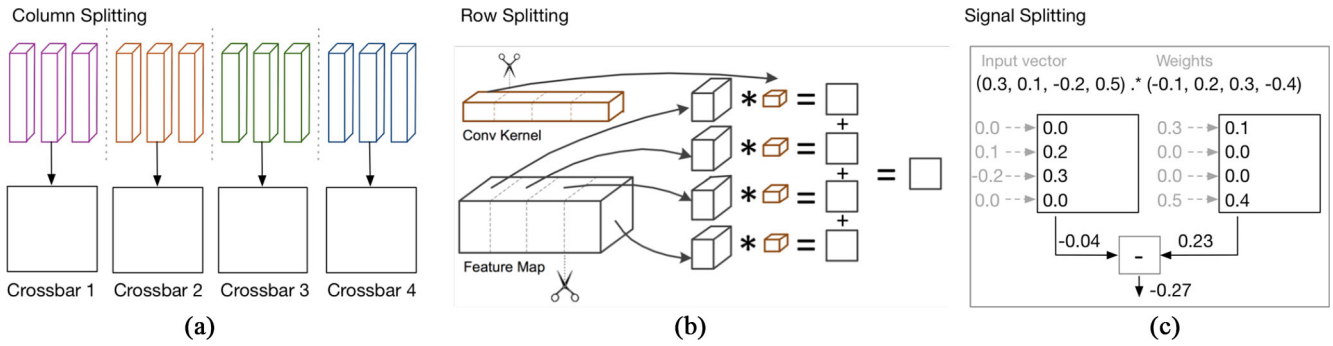


Fig. 3. Matrix splitting. Split the Conv kernels into multiple groups through (a) row-wise, (b) column-wise, and (c) signal-wise, and map to corresponding crossbars.

data buffering and fetching, as shown in Fig. 2(f). For the FC layers, the regular buffers are used since the nearby layers are fully connected.

B. Convolver Circuit: Problem of Matrix Splitting

The convolver circuit for one particular layer is shown in Fig. 2(e), which is designed on the basis of RRAM crossbar-based MVM described in Section II-B. The main structure of the convolver circuit is constructed by groups of crossbars for the Conv operations. Some digital peripheral units, including the specific circuits for neurons, etc., are also placed in the front of or at the back of the crossbar groups.

Considering both the large energy cost of the writing operation and the endurance limit of the RRAM device [25], it is inpractical to reuse the RRAM crossbar by repeatedly write operations. Because the high area density is one of the strong points of RRAM, all Conv kernels can be mapped onto the crossbars in the convolver circuit of the corresponding layer. In this way, each output channel is able to obtain one output element in one processing cycle, if enough data have been fed into this layer's line buffers by the former Conv layer. However, if the size of the Conv kernels is larger than that of the crossbar, then matrix splitting is necessary for the weight mapping. For FC layers, the splitting strategies are similar to those of the Conv layers.

1) Column Splitting: In most mapping methods, different crossbar columns represent their corresponding convolution output channels. Therefore, if the crossbar column count (W) is smaller than the Conv output channel count (C_{out}) of this layer, then the C_{out} Conv output channels need to be split into $X_{\text{out}}^{(\text{Conv})}$ groups of RRAM crossbars, as shown in (5) and Fig. 3(a). The copies of the input feature maps with the same channel out as one Conv kernel ($h \cdot w \cdot C_{\text{in}}$) are sent to each group of crossbars

$$X_{\text{out}}^{(\text{Conv})} = \left\lceil \frac{C_{\text{out}}}{W} \right\rceil. \tag{5}$$

2) Row Splitting: Fig. 3(b) shows the basic process of row splitting. Our mapping method reshapes a Conv kernel to a column vector and maps it onto one crossbar column. Therefore, if the cross-point count (H) in one RRAM column is smaller than the vector size ($h \cdot w \cdot C_{\text{in}}$), then the elements of one Conv kernel will be split into $X_{\text{in}}^{(\text{Conv})}$ groups, as calculated through (6). Each group will be mapped to one RRAM crossbar. Similar and necessarily, the input feature maps are also split into $X_{\text{in}}^{(\text{Conv})}$ groups, and are sent to corresponding crossbar input interfaces. The partial IAs are achieved from different groups of crossbars, so adder trees need to be cascaded after the crossbar groups to accumulate all the $X_{\text{in}}^{(\text{Conv})}$ IA and obtain the final MAs

$$X_{\text{in}}^{(\text{Conv})} = \left\lceil \frac{h \cdot w \cdot C_{\text{in}}}{H} \right\rceil. \tag{6}$$

Generally, the IAs before adder trees should be high-precision data. However, since the cascaded digital functions, i.e., the nonlinear function and BN function, are monotone increasing functions, the low-bit quantization can be merged with these functions by changing the threshold and output data range. Therefore, the results after addition can also be only low precision. Based on this observation, we also attempted to lower the requirements on the ADCs' precision, which can save a large amount of overhead, especially when the splitting amount is large.

3) Signal Splitting: The resistance of the RRAM device is definitely positive, so it is unable to store negative values. Commonly seen methods to handle the negative weights representation include the following: 1) using a column of reference RRAMs to shift the weight range to a totally positive range, i.e., add an offset to each weight $W'_{i,j} = W_{i,j} + W_{\text{offset}}$, to ensure that all weights larger than zero and 2) mapping one weight matrix onto a crossbar pair, namely, one crossbar for positive weights and the other for negative weights. Then, the exact result will be achieved by a subtraction operation of the two outputs, as shown in Fig. 3(c). Both methods have advantages and disadvantages. The latter requires more hardware resources because the crossbar pair is introduced, while it provides a doubled conductance range; thus, one more bit precision can be achieved. Moreover, this approach can alleviate the pressure of input voltage drivers and the current density of the bit lines since the inputs are distributed to two crossbars. For these reasons, we adopt the latter method in our design.

C. Line Buffer and Pipeline Implementation

However, our accelerator is still challenged by an irreducible buffer overhead. Because the RRAM crossbar uses multiple inputs in the same cycle, the processed data between layers need to be buffered. Therefore, thousands of registers and

corresponding multiplexers are required for large networks. ISAAC [8] provides a rough design for pipelining the Conv operations of different layers where weight duplication is introduced for balancing the load of the pipeline, but a thorough discussion on pipelined implementation is still lacking. Because only a few registers are used in each cycle, there exists parallel potential between layers to reduce the buffer size while boosting the processing speed. As a result, a pipeline design between layers is necessary for both CNN and LB-CNN accelerators.

As mentioned previously, the sliding window exists in the Conv layers. Data dependency analysis shows that the convolver circuit can awake (A) from sleep (S) mode once the input data in a Conv kernel sliding window is fetched. In this way, the structure of the line buffer is introduced for the following reasons. First, much fewer registers are required for data buffering because there is no need to buffer the whole input feature maps. Second, with line buffer introduced in every Conv/pooling layer, a pipeline can be implemented, which makes the inference process much faster than computing the Conv layers in one-by-one mode. This process is the same for the pooling layers.

As the pooling layer is optionally followed by the Conv layer, there exist *Conv–Conv* and *Conv-Pooling-Conv* two modes for the nearby layer relationships. Here, we use the dataflow behavior of CIFAR-10 on the VGG-8 experiment as a case study to show the line buffer-based pipeline implementation.

1) Conv–Conv: The Conv layers in VGG-Net filter the input image with kernels of size 3×3 with a stride of 1, and zero padding is introduced in order to keep the input and output feature map as the same size. Here, we set the kernel size to $h \cdot w$, the feature map size to $H \cdot W$ and the padding pixel count to p. When the feature map is fed in by following the row-major order, the line buffer of each channel only needs $(h-1) \cdot (W+p) + w$ registers. In the initial periods of a layer, zero padding in the length of $(W+p)$ and the first row of $x^{(k)}_{1,:}$ are sent sequentially into the kth Conv layer's line buffer before T_0. And in these cycles, the convolver circuit of the kth is in the sleep (S) mode. Finally, at the T_0 cycle, $x^{(k)}_{2,1}$ is sent into the line buffer, as the dataflow shown in Fig. 4. In the next cycle, the input line buffer shown in Fig. 2(f) is fulfilled by data, and therefore kth Conv layer starts at time T_1. Additionally, at the end of the layer's computation, $(W+p)$ cycles are needed for computing the last row just like the initial cycles.

A main challenge for the Conv–Conv pipeline design is the sleep control for the "line feed" problem. When the computation of a row is accomplished, the input data need to be changed from the end of current line to the front of the next line, which means at least $(w-1)$ data (usually we have $w > 3$) in the next layer need to be prepared. However, for the line-buffer-based pipeline design, the input field shown in Fig. 2(f) is invalid during the preparing cycles, e.g., $(x^{(k)}_{i-3,1}, 0, x^{(k)}_{i-2,W}; x^{(k)}_{i-2,1}, 0, x^{(k)}_{i-1,W}; x^{(k)}_{i-1,1}, 0, x^{(k)}_{i,W})$. In these cycles, the Convolver of the kth layer is also in the S mode; while for the line buffer of the $(k+1)$th layer, there is no valid input. Fortunately, we find that the zero padding of next

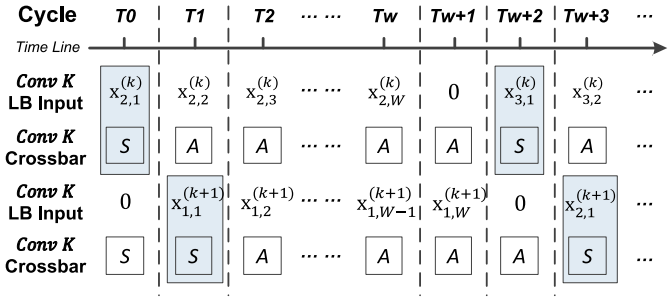


Fig. 4. Dataflow of Conv–Conv. The first line shows the line buffer's input data of previous Conv layer in each cycle, and the third line shows the input data of next Conv layer. The second and forth line show whether the Convolvers are awake (A) or sleep (S).

Conv layer can just exploit this cycle. And in the next cycle, i.e., cycle $T_{m(W+1)+2}$ $(m = 0, 1, \ldots)$, the Convolver of the kth layer recovers to awake (A); while the Convolver of the $(k+1)$th layer begins to sleep (S) for line feed. In this way, the line feed problem is solved by utilizing the extra sleep cycle in each layer for zero padding, and the works in fully pipelined parallelism without waiting. Based on this structure, we achieve the fewest possible cycles for Conv–Conv pipeline connections.

2) Conv-Pooling-Conv: For the pooling layers in most neural networks, the "kernel size" is set as 2×2, and the "stride" s is set as 2. As stride is larger than 1, the pooling circuit will work for one row when every s rows are ready. In Conv-Pooling pipeline, just as shown in Fig. 5, the kth pooling circuit sleep from cycle T_{W+3} to T_{2W+1}. For the awaken row, the pooling circuit will work once every s data are sent into the pooling line buffer. As shown in Fig. 5, the kth pool circuit awakens one cycle and sleeps one cycle from cycle T_3 to T_{W+1}. Although the problem of line feed also exists, the sleep cycles can be hidden into with the "sleep row," and zero padding is not introduced in pooling layer, as shown in cycle T_1 and cycle T_{W+2}. While for the Pooling-Conv line buffer of the next layer, it is just the turn of zero padding in this cycle like Conv–Conv pipeline.

Finally, in the pipeline implementation, the total cycle amount for one complete forward process is shown as

$$T_{\text{pip}} = \left(W^{(1)} + p\right) \cdot \left(H^{(1)} + 2p\right) + \sum_{i \in \text{Conv}}^{i>1} \left(W^{(i)} + p\right) + \sum_{i \in \text{Pool}} 1 + \sum_{j \in \text{FC}} 1 \tag{7}$$

$(W^{(1)} + p) \cdot (H^{(1)} + 2p)$ is the computation cycle amount for the first Conv layer. After that, once the cascaded layer is a Conv layer, $(W+p)$ cycles are needed for computing the last row. Otherwise, only one extra cycle is needed for computing the last pixel of next pooling layer, or to perform an FC layer. The pipelined cycle amount is much fewer than the straight forward layer-by-layer design whose cycle amount is

$$\sum_{i \in \text{Conv}} \left(W^{(i)} + p\right) \cdot \left(H^{(i)} + 2p\right) + \sum_{i \in \text{Pool}} \left(W^{(i+1)} H^{(i+1)}\right) + \sum_{j \in \text{FC}} 1. \tag{8}$$

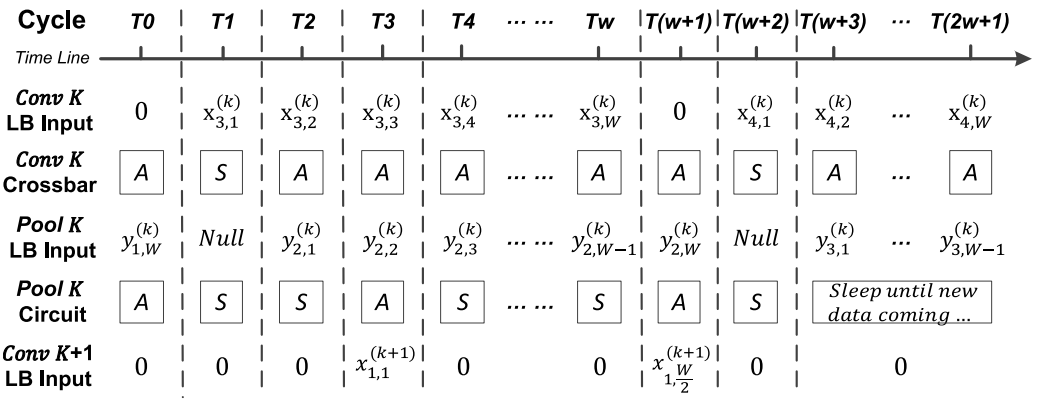


Fig. 5. Dataflow of Conv-Pooling. The first line shows the line buffer's input data of previous Conv layer in each cycle; the third line shows the input data of next pooling layer; the second and forth line show whether the convolvers are awake (A) or sleep (S).

Data dependency exists between the pooling operation and the Conv operation. For the pooling operation, it has to get the data in the window sized of the pooling kernel $h^{(\text{pool})} \times w^{(\text{pool})}$. While for the convolver circuit ahead, it can only output one element for each output channel in one processing cycle. Therefore, the classic structure of line buffer proposed in [26] is introduced as pooling line buffer to collect enough data for pooling from the convolver circuit. When feature map goes through the buffer in row-major layout, the line buffer releases a window selection function on the input feature map. The delay of each line is determined by the width $W^{(\text{pool})}$ of the feature map; the size of the selected window is determined by the pooling kernel $h^{(\text{pool})} \times w^{(\text{pool})}$; and the size of the whole pooling line buffer is $h^{(\text{pool})} \times W^{(\text{pool})} \times C$ where C is the channel count of the feature maps. The circuit of the "max" function, which followed the selected window, makes the max pooling operation.

Similar line buffer structure is also introduced as Conv line buffer, as shown in Fig. 2(e), to fill the input feature map buffer of the next Conv layer with the output of the max pooling operation. For Conv line buffer, the delay of each line is determined by the width of the input feature map of the $(i+1)$th Conv layer $W^{(i+1)}$ (the same as that of the output feature map of the pooling layer ahead); the size of the selected window is the same as that of the Conv kernel $h^{(i+1)} \times w^{(i+1)}$; and the size of the whole pooling line buffer is $h^{(\text{pool})} \times W^{(\text{pool})} \times C_{\text{in}}^{(i+1)}$ where $C_{\text{in}}^{(i+1)}$ is the channel count of the feature maps.

Besides, skip connections are frequently applied in state-of-the-art neural architectures since ResNet [27] was presented. Unlike the straightforward structures as Conv–Conv or Conv-Pooling-Conv, additional buffers are introduced for the *element-wise* addition operations. The buffer size is dependent on the number of skip layers. The corresponding buffered pixels can only be freed when the addition calculations are completed. For example, the residual block in ResNet, which usually skips two Conv layers to construct a shortcut path, needs to put the same size of buffer as the first Conv layer for the element-wise additions. This is because the data coming from the shortcut-connected path need to wait for the output results coming from the Conv path.

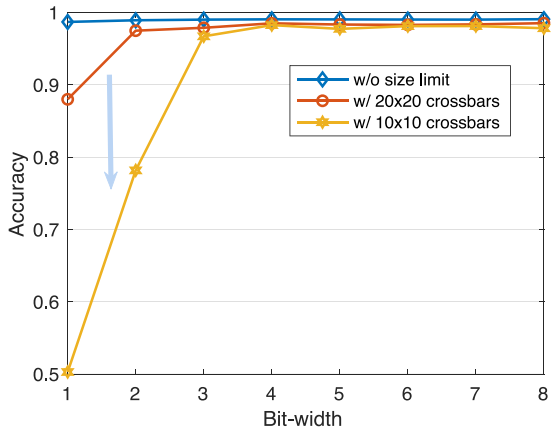


Fig. 6. Test accuracy of LeNet-5 on MNIST under different crossbar size configurations: without size limit, with 10×10 crossbars, and with 20×20 crossbars. As can be seen, the accuracy substantially drops when directly mapping the models onto the size-limited crossbars.

V. Splitting and Quantizing While Training

As described in Section III, larger convolution kernels and input feature maps should be split and mapped to multiple crossbars. Correspondingly, the convolutional operations are also divided into multiple sub-MVM blocks. In this scenario, all the partial IAs of these blocks need to be merged to get the final output feature maps. From the principle of the numerical calculation, m-bit input activations multiplied with n-bit weights and accumulated by k times will lead to $\log_2(k) + m + n$-bit results. That is, the ADCs with the same resolution are required to get the right results, as shown in Fig. 7(a). Thus even in the simplest case with 1-bit activation, 1-bit weights, and 512 long crossbar columns, the bit-width of IAs should approach $\log_2(512) + 1 + 1 = 11$ bits. While as the energy cost of ADCs is positively correlated with the resolution, as observed in Table I, such high-precision converters will greatly degrade the energy efficiency.

However, the reduction of the bit-width will also lead to the network information loss, thereby leading to the accuracy degradation, which is mainly caused by the mismatch of algorithm and hardware. As shown in Fig. 6, our experiments

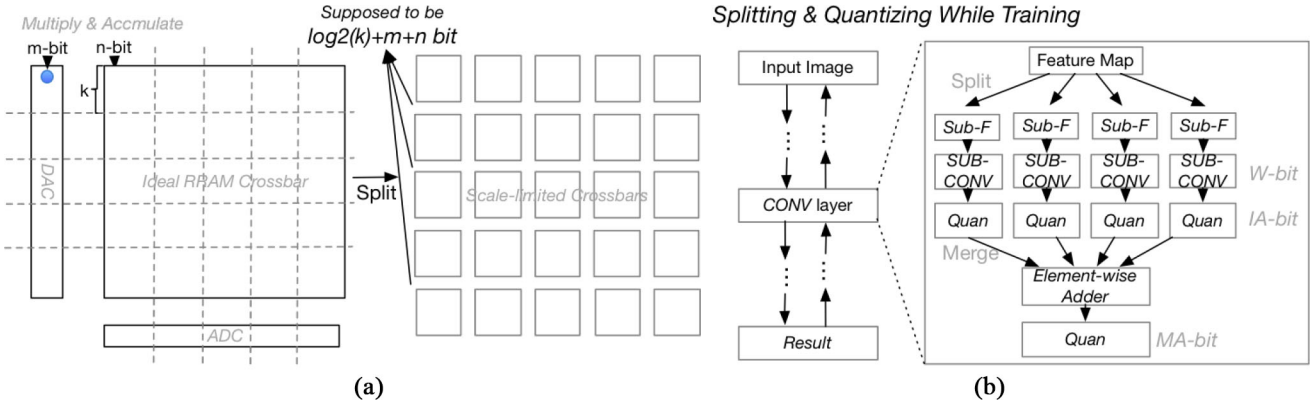


Fig. 7. (a) Bit-width setup and requirements of deploying CNNs on ideal and nonideal scale-limited RRAM crossbars. Here, m represents the bit-width of input feature maps, n represents the bit-width of weights, and k represents the column length of crossbars. (b) Splitting and quantizing while training framework based on RRAM crossbars. W, IA, and MA are introduced to represents the bit-width of weights, IA, and MA, respectively.

have demonstrated that if the model parameters are directly mapped onto the RCS, then using low-bit interfaces for the intermediate data conversion will lead to a significant accuracy decreasing. In the 1-bit case, the accuracy of the recognition task decreases from approximately 99% to 50.29%. Therefore, we propose a specific algorithm which incorporates the splitting and quantization with training, to constructing a bridge for connecting the algorithm and hardware.

A. Training Algorithm

As shown in Fig. 7(a), the regular nonsplitting RRAM crossbar-based convolution framework shifts data from high-precision to low-bit fixed-point after the sums of all the IA are obtained. In our convolution framework, as shown in Fig. 7(b), the IA of all blocks will first be quantized to low bit-width numbers, and then be accumulated by element-wise adders. Then, the MA will be quantized again and sent to the next layer. The forward pass will go through the quantized neural network models with low-bit weights and activation, and the backward pass will propagate floating-point gradients. The training flow is shown as the following steps.

Step 1: Analyze the defined model structure and decide the number of blocks that need to be split.

Step 2: Generate a new model with multiple sub-Convs in each convolutional layer. Add the quantization function for the forward pass.

Step 3: Train the neural network based on the generated model using a normal SGD optimizer.

B. Quantization Methodology

As stated above, quantizations are required since there exist limits on the precision of RCSs. In this paper, the quantization methodology can be decomposed as the following components.

1) Scaling: We use linear scaling for normalizing the activation vectors with irregular values to vectors with values in the range $[-1, 1]$. The quantizations are performed before the ReLU function, so the normalized vectors can lie in a negative range. For each quantization, we will first search the minimum element α which is the power of 2 and able to cover the largest absolute value in the vector v_{in}. Then the scaling function can be formulated as $\text{Scale}(v_{\text{in}}) = (v_{\text{in}}/|\alpha|)$.

2) Uniform Quantization: There are various methods for quantizing numbers. To avoid complex quantizing operations, we simply use uniform quantization in our training and inference process. The quantization function with k bits will be defined as

$$\hat{Q}(x) = \frac{\text{round}\big((2^{k-1}-1)x\big)}{2^{k-1}-1}. \tag{9}$$

Then, the total quantization function can be defined as below. Because the scaling factor α is the power of 2, the multiplication with α can be simply implemented by shifting

$$v_{\text{out}} = Q(v_{\text{in}}) = \alpha \hat{Q}\Big(\frac{v_{\text{in}}}{\alpha}\Big). \tag{10}$$

3) Gradient Backpropagation: Mathematically, the quantization function with continuous inputs and discrete outputs would have zeros gradient. While the training requires backward-propagated gradients to search the optimization space. Therefore, we utilize the straight-through estimator (STE, as also adopted in [18]) to generate the gradients

$$\frac{\partial \text{Cost}}{\partial v_{\text{in}}} = \frac{\partial \text{Cost}}{\partial v_{\text{out}}}. \tag{11}$$

In our accelerator system, there are three types of data that need to be quantized by the above function, the weights, the intermediate output data of split blocks, and the merged final output (i.e., the activation of the corresponding layer). We note them as W (weights), IA (intermediate activations), and MA (merged activations), respectively.

VI. Experimental Evaluation

In this section, we first introduce the setup of our experiments, including the benchmark, configurations, and parameters of simulation. Then, we analyze the experimental results from several concerned aspects, including the accuracy, speed, area, and energy efficiency, to evaluate the system performance.

A. Experimental Setup

1) Benchmark: We construct the accelerator designs for three typical CNN models, namely, LeNet [28], ResNet [27], and VGGNet [1]. LeNet is a simple yet efficient network for the recognition of handwriting digits. ResNet is a powerful network that has been widely utilized in many computer vision applications. VGGNet is a relatively large neural network whose Conv weight matrix can reach $3 \times 3 \times 512 \times 1024$. We also selected two popular classification datasets to demonstrate the accuracy, MNIST [29] for LeNet-5 and CIFAR-10 [30] for ResNet-18 and VGG-8.

2) Modeling Hypothesis: For inference processing, the multibit model is achieved by dynamically quantizing [4] the well-trained floating-point model into 8 bits, and the LB-CNN models are trained by our training algorithms. The crossbars are set to different sizes to evaluate the effects of splitting. If one crossbar pair is not large enough to store all parameters of one layer, then the matrix splitting strategies are performed. In default, 8-bit RRAM devices and 8-bit interfaces are introduced in multibit CNN (MB-CNN) models.

For the crossbar, the design of the RRAM cell can be simpler if binary weights are chosen. For the multibit mode, the 1T1R RRAM cell is introduced. This is because we have to tune each RRAM cell to a particular resistance before the crossbar is used for computing. When tuning, the transistor of the to-be-tuned RRAM-cell is on and others are off. In this way, the decoder is able to support the RRAM cells opening cell-by-cell. For the binary weights, only two values of the RRAM resistance (the ON/OFF state) are used. In this way, the 0T1R RRAM cell, which each cell only takes up the area of $4F^2$, can be used. The one-by-one tuning method is introduced as "half-selection" in [31].

3) Simulation Parameters: To estimate the area and energy overhead, we used reliable data from relevant papers as a basis [32]–[41]. For the RRAM crossbar part, NVSim [40] is used to estimate the overhead because it provides complete RRAM device data and key indicators. For the ADC part, we refer to 2 work for 8-bit [31] and 4-bit [36] ADC designs because the resolutions and frequencies satisfy the experimental demand. For the adders, [37] provides the power and area of the adders. For the LUT and the SRAM part, [37] also provides the power overhead of 32-bit SRAM. The power estimation is based on the required size of the LUTs and line buffer. We estimate the system performance by combining the costs of all circuit elements. The simulation parameters of the power and area cost for each circuit element are shown in Table II. Based on the device data, we construct a spreadsheet, which works in the similar way as MNSIM [41], to estimate and sum up the overhead of all system parts.

TABLE II
POWER AND AREA COST OF EACH CIRCUIT ELEMENT

	Area	Power(mW)
1T1R RRAM device	$(1+\frac{W}{L}) \cdot 3F^2$	0.052[b]
0T1R RRAM device	$4F^2$	0.06[b]
8bit DAC	$3096T$[a] [32]	30 [35]
Sense Amplifier	$244T$ [32]	0.25 [34]
8bit ADC	$2550T + 1k\Omega(\approx 450T)$ [32]	35 [33]
4bit ADC	$72T$ [36]	12.4 [36]
8bit SUB	$256T$	$2.5 \cdot 10^{-6}$ (0.025pJ [37])
8bit ADD	$256T$	$2.5 \cdot 10^{-6}$ (0.025pJ [37])
32bit SRAM SpadMem	$192T$	0.064 (5pJ [37])[c]

[a] $T = W/L \cdot F^2$, where $W/L = 3$, and the technology node $F = 45nm$.
[b] The power consumption of RRAM cell is estimated by $V_{avg}^2 g_{avg}$, where $g_{avg} = \sqrt{g_{on} g_{off}}$ [40].
[c] The energy consumption of digital arithmetic logics and memory access refer to the energy table under 45nm CMOS technology node [37]. The system clock is assumed to be 100MHz, which is determined based on the speed of ADC/DACs and the latency of RRAM crossbar [39].

B. Accuracy: Effects of Device Variation Under Different Bit-Levels

Variation exists when mapping the weight parameters to the RRAM devices since it is one conductance range (not a specific conductance value) that represents one fixed-point number. When one RRAM device is able to represent N bits, i.e., 2^N conductance ranges represent 2^N fixed-point weights, respectively. For the kth conductance range, $g_{(k)}$ represents the center conductance, and $(g_{(k)} - \Delta g, g_{(k)} + \Delta g)$ represents the conductance range, i.e., the device variation δg ranges from $(-\Delta g, \Delta g)$. In previous work [42], a typical model of the RRAM resistance variation satisfies the distribution of $\Delta R/R = 9\%$, which means that the variation varies at different conductance levels. In our experiments, we simplify the model and assume that the variation range Δg is the same for each conductance range. When the RRAM device is used in the *binary mode*, only two conductance ranges are picked from 2^N ones. In this way, the expectation of $(\Delta g/g)$ can be smaller than in the case that 2^N ranges are all in use (we just name it as *full bit-level mode*), thus introducing less computing error for MVMs. There are relative studies which have made attempts to resolve the variation problem. For example, in physical level by inserting a TiN buffer layer [43] or in algorithm level by leveraging the self-healing capability of CNNs [44], etc. Our experiments also demonstrate that the of the low-bit neural networks has stronger variation-tolerant ability.

LeNet on the MNIST dataset is demonstrated as a case study to show the effects of device variation under different weight bit-levels. Without considering device variation, a precise mapping is made from the quantized fixed-point weight parameters to the RRAM conductances in the full bit-level mode. In this way, the increasing recognition error rate mainly results from the quantization error. While in the binary mode, the recognition performance remains the same for RRAM of different bit-levels when neglecting device variation, though the recognition error is a bit higher than that of full bit-level mode in the case of 7-bit and 5-bit RRAM, as listed in Table III. When considering device variation, the recognition performance in the binary mode shows better robustness: in binary mode, device variation introduces less than 0.01% error rate increase in the case of 3 bit (or larger bit-level) RRAM, while in full bit-level mode, the recognition performance in 3-bit RRAM becomes worse than that in binary mode due to the larger effect of device variation.

Meanwhile, the error rate rises to around 90% when quantizing the full-bit weights into 2 bits, i.e., the model absolutely fails for both with and without considering variation (since the

TABLE III
ERROR RATE OF LENET ON MNIST: DEVICE VARIATION EFFECTS UNDER DIFFERENT WEIGHT BIT-LEVELS

Weight Bit Level	RRAM Bit Level[a]	RRAM Used in Full Bit-level Mode		RRAM Used in Binary Mode	
		No Variation	With Variation	No Variation	With Variation
8 bit	7 bit	0.58%	0.58%	0.73%	0.74%
6 bit	5 bit	0.60%	0.59%		0.75%
4 bit	3 bit	0.80%	1.21%		0.75%
2 bit	1 bit	90.67%	89.10%		0.86%

[a] The bit-level of RRAM devices is 1-bit less than that of the weight parameters because of the signal splitting.

TABLE IV
CLASSIFICATION ACCURACY FOR MNIST AND CIFAR-10 TEST DATASET WITH DIFFERENT COMBINATIONS OF BITWIDTH AND DIFFERENT CROSSBAR SIZE LIMITS, TESTED ON LENET-5, RESNET-18, AND VGG-8, RESPECTIVELY. W, IA, AND MA ARE BIT-WIDTH OF WEIGHTS, IA, AND MA

Model	Dataset	Crossbar Size	W Bitwidth	IA Bitwidth	MA Bitwidth	Accuracy
LeNet-5	MNIST	10×10	8	8	8	99.08%
			4	4	4	99.06%
			2	2	2	**98.93%**
			1	1	1	98.70%
		20×20	8	8	8	99.10%
			4	4	4	99.08%
			2	2	2	**99.01%**
			1	1	1	98.81%
ResNet-18	CIFAR-10	128×128	32[a]	32	32	89.12%
			4	4	8	85.14%
			4	4	4	82.78%
			2	**4**	**4**	**81.22%**
		256×256	4	4	8	85.13%
			4	4	4	81.10%
			2	**4**	**4**	**81.03%**
VGG-8	CIFAR-10	128×128	32[a]	32	32	93.40%
			4	4	8	92.06%
			4	4	4	89.44%
			2	**4**	**4**	**89.08%**
		256×256	4	4	8	92.20%
			4	4	4	89.72%
			2	**4**	**4**	**88.83%**
		512×512	4	4	8	91.95%
			4	4	4	89.88%
			2	**4**	**4**	**90.03%**

[a] Bitwidth=32 represents corresponding models were trained with 32-bit floating-point numbers and unlimited crossbar size.

MNIST dataset has only ten categories). The reason is that the direct quantization of neural network will introduce large computation error under extremely low-bit weights. Therefore, the incorporation of quantization with training is also important to enhance the recognition accuracy.

C. Accuracy: Effects of Matrix Splitting

As stated before, we propose the splitting-and-quantizing-while-training algorithm to overcome the accuracy drop caused by matrix splitting. To demonstrate feasibility, experiments are constructed to evaluate the performance of different low-bit CNNs with different crossbar sizes. As the LeNet model is not a large network, we set the length of the RRAM crossbar column to 10 and 20 to study the effect of matrix splitting. For the larger CNN model, we choose ResNet-18 and VGG-8 to train on a more complex dataset CIFAR-10.

Table IV shows the test accuracy of the trained neural network models. We first evaluate the simpler dataset MNIST, the accuracy can still maintain around 99% even under 1-bit weight, 1-bit IA, and 1-bit merge activation. For the more complex dataset CIFAR-10, the low-bitwidth models lose some accuracy compared to the floating-point & nonsplitting baseline. We can also conclude that VGG-8 shows much better adaptability than ResNet-18. For example, with (W, IA, MA) = (2, 4, 4) quantization and 128×128 crossbar size, the accuracy of VGG-8 drops approximately 3.4% while the ResNet-18 drops 8% accuracy. This is because there exists more redundancy in VGG-8, thus it still has a strong feature extraction ability at low bitwidth quantization.

There are also interesting trade-offs between the test accuracy and different bitwidth combinations. Because the low-bit mode of the RRAM device and the low bit-width interfaces can provide many benefits for circuits design and resistance tuning, the weight and activation with lowest possible bitwidth are preferred for the deployment. However, the accuracy is positively correlated with the bitwidth of weights and activation. Taking ResNet-18 as an example, the accuracy decreases significantly with the bitwidth reduction (from 89.12% to 81.22%). Therefore, the bitwidths need to be carefully adjusted according to the application scenarios.

We have also explored the impact of different crossbar array sizes. We train multiple models under the different crossbar size configurations, ranging from 128 to 512. The experimental results demonstrate that our training algorithm is robust and performs well under various size configurations. As shown in Table IV, with the crossbar size as a variable, there is almost no fluctuation (up to approximately 1%) in the test accuracy.

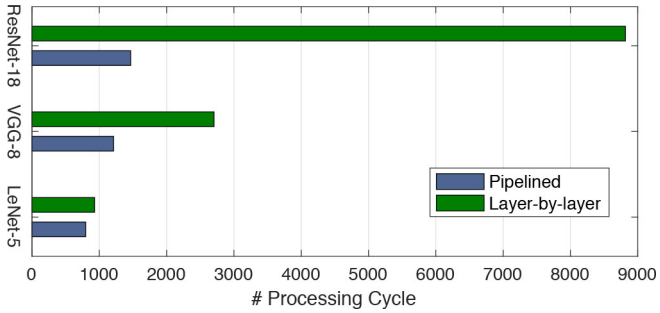


Fig. 8. Processing cycle amount of pipelined processing and layer-by-layer processing.

D. Pipeline Evaluation

We evaluate the performance of the pipeline strategy. Theoretically, the improvement in cycle amount is independent on the input image but dependent on the network structure. To quantitatively evaluate the cycle savings, we take several datasets with different image sizes as examples. For LeNet-5, we demonstrate the cycle amount on the MNIST dataset, with a 28×28 input image to the neural network. For ResNet-18 and VGG-8, 32×32 CIFAR-10 images are fed into the neural network.

The processing cycle amount can be obtained by 7 and 8 for pipeline implementation and layer-by-layer processing, respectively. Fig. 8 shows the cycle amount comparisons. The speedup ratio achieves approximately 1.16×, 2.23×, and

TABLE V
AMOUNT AND PROCESSING COUNT OF COMPUTING UNITS, INTERFACES, AND BUFFERS

Module	Layer	Amount	Processing Count
RRAM cell	Conv	$(h \cdot w \cdot C_{\text{in}}) \cdot C_{\text{out}} \cdot X_{\text{out}} \cdot X_{\text{out}}$	$H_{\text{out}} \cdot W_{\text{out}}$
DAC	Conv	$(h \cdot w \cdot C_{\text{in}}) \cdot X_{\text{out}}$	$H_{\text{out}} \cdot W_{\text{out}}$
SA&ADC	Conv	$C_{\text{out}} \cdot X_{\text{in}}$	$H_{\text{out}} \cdot W_{\text{out}}$
Feature Map Buffer	Conv	$h \cdot w \cdot C_{\text{in}}$	$H_{\text{out}} \cdot W_{\text{out}}$
Line Buffer	Conv	$h \cdot W_{\text{in}} \cdot C_{\text{in}}$	$H_{\text{out}} \cdot W_{\text{out}}$
Line Buffer	Pooling	$h \cdot W_{\text{in}} \cdot C_{\text{in}}$	$H_{\text{out}} \cdot W_{\text{out}}$
RRAM Cell	FC	$C_{\text{in}} \cdot C_{\text{out}} \cdot X_{\text{out}} \cdot X_{\text{out}}$	1
DAC	FC	$C_{\text{in}} \cdot X_{\text{out}}$	1
SA&ADC	FC	$C_{\text{out}} \cdot X_{\text{in}}$	1
Feature Map Buffer	FC	C_{in}	1
Adder	Conv	$C_{\text{out}} \cdot X_{\text{out}}$	$H_{\text{out}} \cdot W_{\text{out}}$
LUT/OR GATE	Pooling	$C_{\text{in}} \cdot X_{\text{out}}$	$H_{\text{out}} \cdot W_{\text{out}}$

TABLE VI
ENERGY AND AREA ESTIMATION OF DIFFERENT RRAM-BASED CROSSBAR PES

Network	Performance	MB-CNN	LB-CNN	Saving
LeNet-5	Energy (uJ/img)	18.39	7.83	57.4%
	Area (mm^2)	0.082	0.024	70.7%
ResNet-18	Energy (uJ/img)	271.22	118.64	58.6%
	Area (mm^2)	0.104	0.047	54.8%
VGG-8	Energy (uJ/img)	4600.99	2076.52	54.9%
	Area (mm^2)	2.34	1.21	48.3%

The energy and area were estimated under different CNNs and input image sizes: 28x28x1 MNIST images for LeNet-5, 32x32x3 CIFAR-10 images for VGG-8 and ResNet-18. The bitwidth configuration of the systems are 2-bit W, 4-bit IA, and 4-bit MA. The bitwidth of the first Conv layer inputs are set to 8-bit.

6.01× on LeNet-5, VGG-8, and ResNet-18, respectively. The improvement varies among the different types of CNNs when applying the proposed pipeline method. For the LeNet-like neural networks, the feature map size reduced quickly through layers, so the system spends many cycles in the first layer and the speed will not be greatly improved. While for ResNet or VGG-like CNNs, the feature map size halves in two adjacent convolution stages, so the forward process can obtain more benefits by applying the pipeline because the parallelism between layers will be fully utilized. It is also worth mentioning that deeper neural networks (e.g., ResNet-18) are expected to achieve more speedup.

E. Area and Energy Estimation Under Different Bit-Levels

Network models of LeNet, ResNet-18, and VGG-8 are demonstrated in the area and energy estimation. Moreover, we also profile the area and energy distribution among different system parts and among different layers on VGG-8. In our estimation, the crossbar-based computing units and the buffers are considered, while the consumption of interconnections (routers) is neglected. The amount and processing count of each module are listed as in Table V. Because of the sliding window operation, modules in Conv layers process $H_{\text{out}} \cdot W_{\text{out}}$ times in one forward process. The area and power consumption of each system part are listed in Table II.

The area and energy estimation is shown in Table VI. On the overall system performance, the LB-CNN on RRAM saves 54.9% of the energy and 48.3% of the area consumption for VGG-8 on CIFAR-10 compared with the MB-CNN. Significant enhancement can also be achieved on the ResNet-18. The area and energy distribution is shown in Fig. 9. Either for low-bit or MB-CNNs, the input and output interface still takes up dominant part of the energy and area consumption. In terms of energy consumption, the Conv layers consume most because the sliding window of each Conv layer has to sweep through the whole feature map in multiple cycles, but FC layers only process once. Comparing the area and energy distribution between the LB-CNN and MB-CNN, the overhead of the input and output interfaces is massively saved (≥60%) due to the reduction of bitwidth.

F. Comparison to Related Studies

The throughput and power efficiency are also evaluated, as shown in Table VII. We make comparisons with several advanced related accelerators, including pure CMOS-based DaDianNao and RRAM-based ISAAC and PipeLayer. Our system constructed for VGG-8 achieves 51.68 TOPS/s throughput, higher than DaDianNao and ISAAC. The power efficiency of our system is 301.7 GOPS/s/W, lower than ISAAC while higher than PipeLayer and DaDianNao. Most importantly, the contributions presented in this paper are orthogonal to the related studies. Whether in ISAAC or PipeLayer, the weights and activation used are with relatively high bitwidth, and the matrix splitting problems are not fully discussed. Therefore, it is valuable to combine our approaches with other RRAM-based accelerator designs.

VII. RELATED WORK

A. Low-Precision CNNs

The researchers have found that lowering the bit-width of weights and neurons have little impact on the performance. Meanwhile, reducing the precision of CNN models can boost the efficiency of inference computations. The methodologies of approaching low-precision CNNs have been well discussed. Previous studies, such as INQ[17], etc., have aimed to reduce the precision of the weights in neural networks. Furthermore, investigations of XNOR-Net [16], BNN [15], etc., have targeted in the quantization of both weights and activations in CNNs. The DoReFa-Net [18] approach introduces the quantization of gradients while training. In industry, NVIDIA has also launched TensorRT [45], which can automatically quantize the neural networks from FP32 to INT8 without accuracy loss. All the above algorithms provide guidance for training LB-CNNs, but they are not fully adept to the RCS, since the quantization mechanism of RCS differs from that of conventional digital systems.

B. Divide and Conquer

With the scale of NN models growing rapidly, *divide and conquer*-type methods are frequently used to extend the scalability of computing systems. Generally, dividing applied in large-scale networks can bring the following two benefits. On the one hand, it can increase the scalability of neural network computing systems. For example, Krizhevsky *et al.* [46] introduced group convolution, splitting the whole AlexNet onto

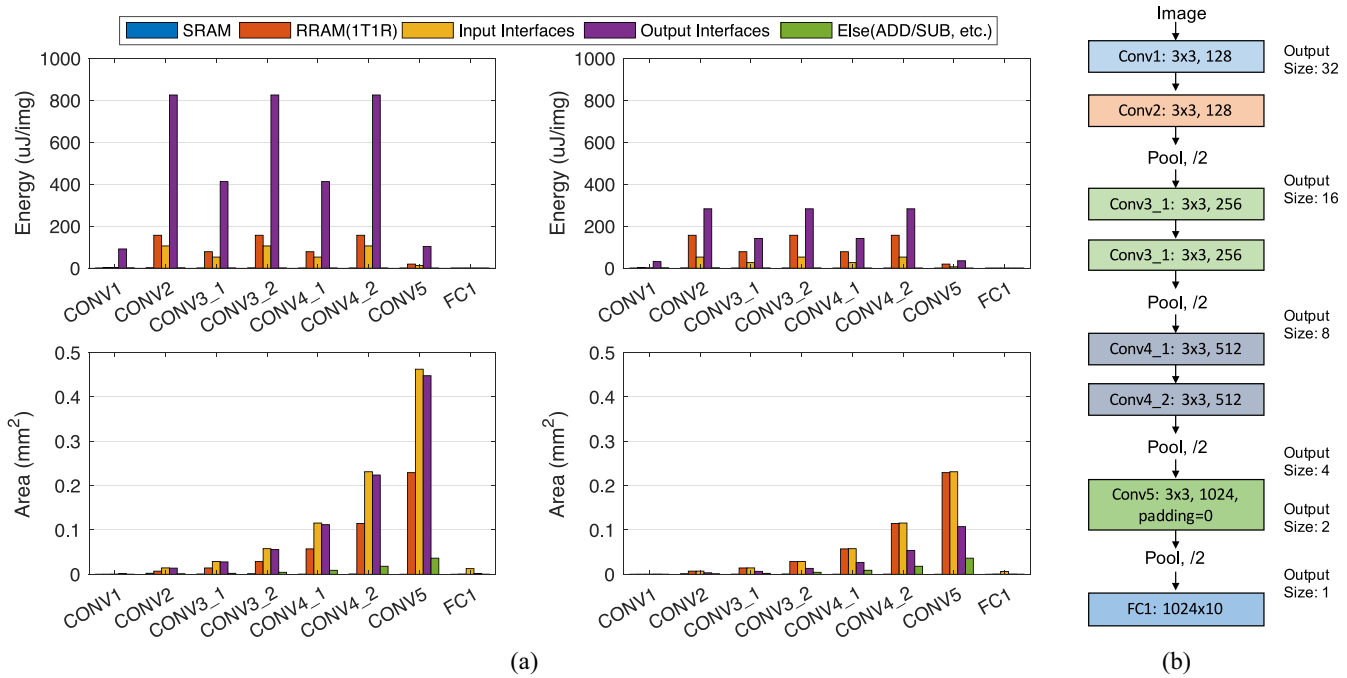


Fig. 9. (a) Energy and area overhead distribution of processing VGG-8 with 32 × 32 input images. Left: multibit VGG-8 with (W, IA, MA) = (8, 8, 8). Right: low-bit VGG-8 with (W, IA, MA) = (2, 4, 4). The cost of RRAMs (1T1R) is assumed as the same under full-bit mode and low-bit mode. (b) Structure of the VGG-8 network.

TABLE VII
COMPARISON TO ISAAC, PIPELAYER, AND DADIANNAO IN TERMS OF THROUGHPUT AND POWER EFFICIENCY

Accelerator	Crossbar Size	ADC Config. Resolution	ADC Config. Power	Peak Throughput ($TOps/s$)	Power Efficiency ($GOps/s/W$)
DaDianNao [6]	N/A	N/A		5.58	286.4
ISAAC [8]	128x128	8 bits	16 mW	41.3	627.5
PipeLayer [9]	128x128	N/A (Spiking Output)		122.7	142.9
Ours(VGG-8)	128x128	4 bits	12 mW	51.68	301.7

two GPUs while training due to the limited graphical memory of the GTX 580 device that they used. In a finer-grained level, Li *et al.* also applied a blocking strategy in their design focusing on FPGA-based sparse MVM (SpMV) acceleration. The authors regularized a sparse matrix into small matrix blocks and then mapped these blocks into highly parallel PEs that enable simultaneous multiplication-accumulation computations. On the other hand, some variables are highly correlated, showing similar computing patterns and impacting the network performance in similar ways. Therefore, processing these variables as a group will not compromise the network performance but will simultaneously reduce the computation complexity. For example, Wen *et al.* [47] divided the weight parameters into groups, implementing structured sparsity learning to obtain regularized network structures. They proposed a group lasso-based approach that tended to remove less important parameter groups so as to obtain structured sparsity. It aggregated parameters with high correlation into same groups. However, such methods have high complexity, which limits the usefulness of the RCS.

C. RRAM-Based Neural Network Accelerators

In previous studies, designs were proposed to accelerate neural network computing based on RRAM, ranging from the circuit designs to the high-level architecture designs. The researchers focused their attention on the efficient mapping and scheduling of deep learning algorithms. PRIME [10] enables in-memory neural network computing based on RRAM-based main memory, achieving 895× the energy efficiency across the benchmarks. ISAAC [8] was also proposed as a full-fledged NN accelerator design, accompanied by a pipeline architecture, encoding techniques, and supporting digital circuits to improve the efficiency. PipeLayer [9] also demonstrated the feasibility of efficient neural computing based on RRAM. However, all of these studies did not fully discuss the limitations of the RCS as stated in Section III, which limits the applicability in real world scenarios.

VIII. CONCLUSION

This paper aims to improve the speed, the energy efficiency, and the area efficiency of RRAM-based neural network inference system. We observe that the high-precision interfaces take up the dominant overhead of the entire system, and the bit-width of weights is also limited due to the variation and limited conductance intervals of the RRAM devices. Therefore, we design an LB-CNN accelerator based on RRAM with all the weights and activations quantized. By introducing low-bit

activations, more than 60% of the interface power and area overhead can be saved. For the overall system constructed for accelerating VGG-8, 54.9% of the energy consumption and 48.3% of the area can be saved. A pipeline strategy is also designed to accelerate the process and reduce the buffer demands. Deeper neural networks are expected to gain more speedup from the pipeline processing, e.g., 6.0× speedup for ResNet-18. Furthermore, to tackle the limited crossbar size problem, we fully discuss the matrix splitting and introduce a training method to incorporate the quantization and splitting with training.

In future work, it is foreseeable that low bit-width quantization will become a key technique in the accelerator designing. Moreover, further exploration on the hardware-aware training algorithms is crucial and valuable for the optimization of RCS.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015. [Online]. Available: https://arxiv.org/abs/1409.1556

[2] J. Fan, W. Xu, Y. Wu, and Y. Gong, "Human tracking using convolutional neural networks," *IEEE Trans. Neural Netw.*, vol. 21, no. 10, pp. 1610–1623, Oct. 2010.

[3] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Boston, MA, USA, 2015, pp. 3128–3137.

[4] J. Qiu *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, Monterey, CA, USA, 2016, pp. 26–35.

[5] T. Chen *et al.*, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 269–284, 2014.

[6] L. Tao *et al.*, "DaDianNao: A neural network supercomputer," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 73–88, Jan. 2016.

[7] M. M. Waldrop, "The chips are down for Moore's law," *Nat. News*, vol. 530, no. 7589, p. 144, 2016.

[8] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, 2016.

[9] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Austin, TX, USA, 2017, pp. 541–552.

[10] P. Chi *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 27–39, 2016.

[11] B. Li, L. Xia, P. Gu, Y. Wang, and H. Yang, "Merging the interface: Power, area and accuracy co-optimization for RRAM crossbar-based mixed-signal computing system," in *Proc. 52nd Annu. Design Autom. Conf.*, San Francisco, CA, USA, 2015, p. 13.

[12] L. Xia *et al.*, "Switched by input: Power efficient structure for RRAM-based convolutional neural network," in *Proc. 53rd Annu. Design Autom. Conf.*, Austin, TX, USA, 2016, p. 125.

[13] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, 2012, Art. no. 075201.

[14] R. Degraeve *et al.*, "Causes and consequences of the stochastic aspect of filamentary RRAM," *Microelectron. Eng.*, vol. 147, pp. 171–175, Nov. 2015.

[15] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4107–4115.

[16] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, Amsterdam, The Netherlands, 2016, pp. 525–542.

[17] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017. [Online]. Available: https://openreview.net/forum?id=HyQJ-mclg

[18] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[19] S. Karmakar, B. Gònen, F. Sebastiano, R. Van Veldhoven, and K. A. A. Makinwa, "A 280μw dynamic-zoom ADC with 120dB DR and 118dB SNDR in 1kHz BW," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2018, pp. 238–240.

[20] H. Chandrakumar and D. Markovic, "A 15.2-ENOB continuous-time $\delta\sigma$ ADC for a 7.3 μW 200mv$_{pp}$-linear-input-range neural recording front-end," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2018, pp. 232–234.

[21] S. Li, B. Qiao, M. Gandara, D. Z. Pan, and N. Sun, "A 13-ENOB second-order noise-shaping SAR ADC realizing optimized NTF zeros using the error-feedback structure," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2018, pp. 234–236.

[22] T. He, M. Ashburn, S. Ho, Y. Zhang, and G. Temes, "A 50MHZ-BW continuous-time $\delta\sigma$ ADC with dynamic error correction achieving 79.8dB SNDR and 95.2dB SFDR," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2018, pp. 230–232.

[23] M. El Ansary *et al.*, "50nW 5kHz-BW opamp-less $\delta\sigma$ impedance analyzer for brain neurochemistry monitoring," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2018, pp. 288–290.

[24] L. Kull *et al.*, "A 24-to-72GS/s 8b time-interleaved SAR ADC with 2.0-to-3.3pJ/conversion and >30dB SNDR at nyquist in 14nm CMOS FinFET," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2018, pp. 358–360.

[25] Y. Y. Chen *et al.*, "Understanding of the endurance failure in scaled HFO2-based 1T1R RRAM through vacancy mobility degradation," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, San Francisco, CA, USA, 2012, pp. 20.3.1–20.3.4.

[26] B. Bosi, G. Bois, and Y. Savaria, "Reconfigurable pipelined 2-D convolvers for fast digital signal processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 3, pp. 299–308, Sep. 1999.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 770–778.

[28] Y. LeCun *et al.*, "Comparison of learning algorithms for handwritten digit recognition," in *Proc. Int. Conf. Artif. Neural Netw.*, vol. 60. Perth, WA, Australia, 1995, pp. 53–60.

[29] Y. LeCun, C. Cortes, and C. Burges, *MNIST Handwritten Digit Database*, AT&T Labs, Florham Park, NJ, USA, vol. 2, 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist

[30] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009.

[31] C. Xu *et al.*, "Overcoming the challenges of crossbar resistive memory architectures," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Burlingame, CA, USA, 2015, pp. 476–488.

[32] R. S. Amant *et al.*, "General-purpose code acceleration with limited-precision analog computation," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 505–516, 2014.

[33] J. Proesel, G. Keskin, J.-O. Plouchart, and L. Pileggi, "An 8-bit 1.5GS/s flash ADC using post-manufacturing statistical selection," in *Proc. CICC*, San Jose, CA, USA, 2010, pp. 1–4.

[34] S. Gupta *et al.*, "Simulation and analysis of sense amplifier in submicron technology," *Int. J. Eng. Tech. Res.*, vol. 2, no. 2, pp. 177–179, 2014.

[35] S. Y.-S. Chen, N.-S. Kim, and J. Rabaey, "A 10b 600MS/s multi-mode CMOS DAC for multiple Nyquist zone operation," in *Symp. VLSI Circuits Dig. Tech. Papers*, Honolulu, HI, USA, 2011, pp. 66–67.

[36] S. S. Chauhan, S. Manabala, S. C. Bose, and R. Chandel, "A new approach to design low power CMOS flash A/D converter," *Int. J. VLSI Design Commun. Syst.*, vol. 2, no. 2, pp. 100–108, 2011.

[37] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, Montreal, QC, Canada, 2015, pp. 1135–1143.

[38] Siddharth, M. Garg, and A. Gahlaut, "Comparative study of CMOS op-amp in 45nm and 180 nm technology," *Int. J. Eng. Res. Appl.*, vol. 4, no. 7, pp. 64–67, 2014.

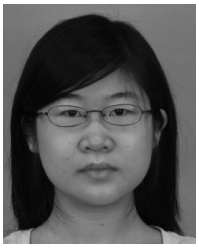
[39] S.-S. Sheu *et al.*, "A 4Mb embedded SLC resistive-RAM macro with 7.2ns read-write random-access time and 160ns MLC-access capability," in *Proc. ISSCC*, San Francisco, CA, USA, 2011, pp. 200–202.

[40] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[41] L. Xia *et al.*, "MNSIM: Simulation platform for memristor-based neuromorphic computing system," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 5, pp. 1009–1022, May 2018.
[42] S. Yu, B. Gao, Z. Fang, H. Yu, J. Kang, and H.-S. P. Wong, "A neuromorphic visual system using RRAM synaptic devices with sub-pJ energy and tolerance to variability: Experimental characterization and large-scale modeling," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, San Francisco, CA, USA, 2012, pp. 10.4.1–10.4.4.
[43] Y. Fang *et al.*, "Improvement of HFO_x-based RRAM device variation by inserting ALD tin buffer layer," *IEEE Electron Device Lett.*, vol. 39, no. 6, pp. 819–822, Jun. 2018.
[44] L. Chen *et al.*, "Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar," in *Proc. Conf. Design Autom. Test Europe*, Lausanne, Switzerland, 2017, pp. 19–24.
[45] (2018). *GTC Talk—8-Bit Inference With TensorRT*. [Online]. Available: http://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf
[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[47] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

Yi Cai received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2017, where he is currently pursuing the Ph.D. degree with the Department of Electronic Engineering.

His current research interests include on-chip neural network system and emerging nonvolatile memory technology.



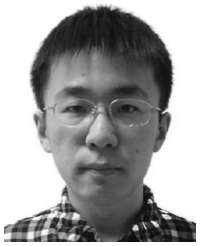
Tianqi Tang received the B.S. and M.S. degrees in electronic engineering from Tsinghua University, Beijing, China, in 2014 and 2017, respectively. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA, USA.

Her current research interests include on-chip neural network system and emerging nonvolatile memory technology.

Lixue Xia (S'14) received the B.S. and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 2013 and 2018, respectively.

He has authored and coauthored over 30 papers. His current research interests include deep learning hardware and parallel circuit analysis.

Dr. Xia was a recipient of the Best Paper Nominations in DAC 2017 and ITC 2018. He was invited to give talks in ACA 2016 and CTC 2016.



Boxun Li (S'13) received the B.S. and M.S. degrees in electronic engineering from Tsinghua University, Beijing, China, in 2013 and 2016, respectively.

His current research interests include energy efficient hardware computing system design and parallel computing based on GPU.

Yu Wang (S'05–M'07–SM'14) received the B.S. and Ph.D. degrees (Hons.) from Tsinghua University, Beijing, China, in 2002 and 2007, respectively.

He is currently a Tenured Professor with the Department of Electronic Engineering, Tsinghua University. He has authored and coauthored over 200 papers in refereed journals and conferences. His current research interests include brain inspired computing, application specific hardware computing, parallel circuit analysis, and power/reliability aware system design methodology.

Dr. Wang was a recipient of the Best Paper Award in ASPDAC 2019, FPGA 2017, NVMSA 2017, ISVLSI 2012, the Best Poster Award in HEART 2012 with nine Best Paper Nominations (DATE18, DAC17, ASPDAC16, ASPDAC14, ASPDAC12, 2 in ASPDAC10, ISLPED09, and CODES09), the DAC under 40 innovator award in 2018, and the IBM X10 Faculty Award in 2010. He served as the TPC chair for ICFPT 2019–2011 and ISVLSI 2018, the Finance Chair of ISLPED 2012–2016, the Track Chair for DATE 2017–2019, and GLSVLSI 2018, and served as the Program Committee Member for leading conferences in these areas, including top EDA conferences, such as DAC, DATE, ICCAD, ASP-DAC, and top FPGA conferences, such as FPGA and FPT. He serves as the Co-Editor-in-Chief for ACM SIGDA *E-Newsletter*, an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, the *Journal of Circuits, Systems, and Computers*, and the Special Issue Editor for *Microelectronics Journal*. He also serves as the Guest Editor for *Integration*, *VLSI Journal*, and the IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS. He has given over 60 invited talks in industry/academia. He is currently with ACM Distinguished Speaker Program. He is a Senior Member of ACM.

Huazhong Yang (M'97–SM'00) was born in Ziyang, China, in 1967. He received the B.S. degree in microelectronics and the M.S. and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 1989, 1993, and 1998, respectively.

In 1993, he joined with the Department of Electronic Engineering, Tsinghua University, where he has been a Full Professor since 1998. He has authored and coauthored over 400 technical papers, 7 books, and over 100 granted Chinese patents. His current research interests include wireless sensor networks, data converters, energy-harvesting circuits, nonvolatile processors, and brain-inspired computing.

Dr. Yang was a recipient of the Distinguished Young Researcher by NSFC in 2000 and the Cheung Kong Scholar by Ministry of Education of China in 2012. He has been in charge of several projects, including projects sponsored by the National Science and Technology Major Project, 863 Program, NSFC, 9th Five-Year National Program and Several International Research Projects. He also served as the Chair of Northern China ACM SIGDA Chapter.