IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 28, 2019, accepted April 28, 2019, date of publication May 9, 2019, date of current version May 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2915787

Cascaded Architecture for Memristor Crossbar Array Based Larger-Scale Neuromorphic Computing

SHENG-YANG SUN, HUI XU, JIWEI LI, QINGJIANG LI, AND HAIJUN LIU
College of Electronic Science and Technology, National University of Defense Technology, Changsha 410073, China
Corresponding author: Haijun Liu (liuhaijun@nudt.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61471377, Grant 61604177, and Grant 61704191.

ABSTRACT Multiply-accumulate calculations using a memristor crossbar array is an important method to realize neuromorphic computing. However, the memristor array fabrication technology is still immature, and it is difficult to fabricate large-scale arrays with high-yield, which restricts the development of memristor-based neuromorphic computing technology. Therefore, cascading small-scale arrays to achieve the neuromorphic computational ability that can be achieved by large-scale arrays, which is of great significance for promoting the application of memristor-based neuromorphic computing. To address this issue, we present a memristor-based cascaded framework with some basic computation units, several neural network processing units can be cascaded by this means to improve the processing capability of the dataset. Besides, we introduce a split method to reduce the pressure of the input terminal. Compared with VGGNet and GoogLeNet, the proposed cascaded framework can achieve 93.54% Fashion-MNIST accuracy and 86.64% CIFAR-10 accuracy under the 4.15M parameters. The extensive experiments with Ti/AlOx/TaOx/Pt we fabricated are conducted to show that the circuit simulation results can still provide a high recognition accuracy, and the recognition accuracy loss after circuit simulation can be controlled at around 0.39%.

INDEX TERMS Cascaded neural networks, memristor, crossbar array, convolutional neural networks.

I. INTRODUCTION

Convolutional neural networks (CNNs) have been broadly used in computer vision tasks such as image classification [1]–[4], they are widely popular in industry for their superior accuracy on datasets. Some concepts about CNNs were proposed by Fukushima and Miyake in 1980 [5]. In 1998, LeCun *et al.* proposed the LeNet-5 architecture based on a gradient-based learning algorithm [6]. While moving ahead with deep learning algorithms, the complexity and dimension of network layers have grown markedly. Whereas, state-of-the-art CNNs demand a large number of parameters and compute at billions of FLOPs, which prevents them from being utilized in embedded applications. For instance, the AlexNet [2] proposed by Krizhevsky *et al.* in 2012, requires 60M parameters and 650K neurons, the ResNet [7], which is widely used to solve detection task [8], has a complexity of 7.8 GFLOPs and fails to accomplish real-time applications even with a powerful GPU.

The associate editor coordinating the review of this manuscript and approving it for publication was Ruqiang Yan.

Over the past few years, embedded neuromorphic processing systems have acquired crucial advantages, such as the ability to solve the image classification problems while consuming very little area and power consumption [9]. Besides these advantages, the functions of human brains can also be simulated by neuromorphic computing [10]. As develops the VLSI industry, more and more devices are beginning to be miniaturized [11]. As previously stated, CNNs lead to an exploding number of parameters in which the relevant hardware costs will be colossal [12], these complex calculations and high memory demands make it impractical for the embedded systems applications, such as robotics and real-time or mobile scenarios in general.

A few network architectures [13]–[16] which utilize memristor crossbar arrays to perform convolution computation have been proposed, these frameworks cost a great deal of time and require high memory. Moreover, the fabrication technology of larger-scale memristive arrays is still not mature [17]–[19], however useful cascaded frameworks in real applications with memristor-based architectures have seldom been reported. Therefore, cascading small-scale

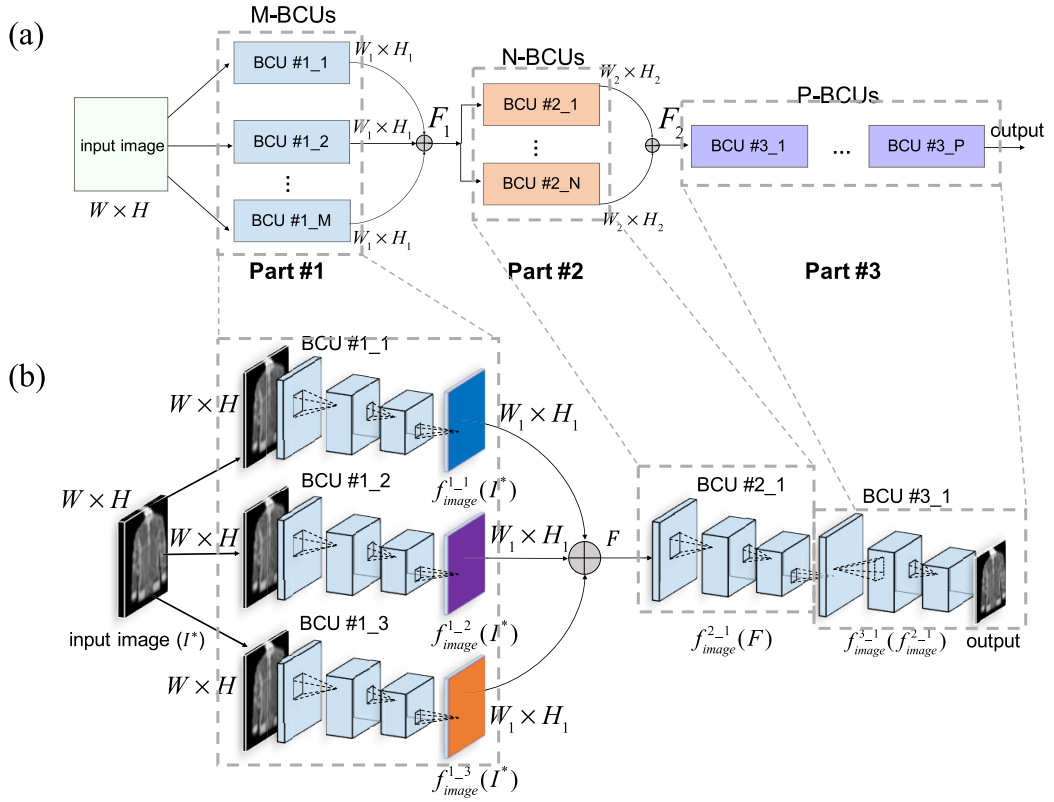

FIGURE 1. The diagram of the proposed cascaded framework. (a): Standard cascaded network framework "*M-N-P*". (b): The typical "3-1-1" cascaded type.

arrays to achieve the neuromorphic computational ability that can be achieved by large-scale arrays, which is of great significance for promoting the application of memristor-based neuromorphic computing. In this paper, we present a memristor-based cascaded framework with some basic computation units, several neural network processing units can be cascaded by this means to improve the processing capability of the dataset. The basic computation unit builds on our prior work [20], which has validated this simplified three-layer CNNs can get desired recognition accuracy.

This paper is structured as follows. In Section II we examine how basic computation units cascade. Section III demonstrates the circuits implemented based on memristor crossbar arrays, a general algorithm mapping method and shows what a split method is. We exhibit the experimental results in Section IV, and conclude this paper in Section V.

II. PROPOSED CASCADED FRAMEWORK

Taking advantage of the proposed basic processing unit, we present the standard "M-N-P" cascaded framework, and more details are demonstrated.

A. THE PROPOSED CASCADED METHOD

The standard "M-N-P" cascaded architecture is demonstrated in Fig. 1(a). The framework consists of several basic computation units (BCU) in series or in parallel.

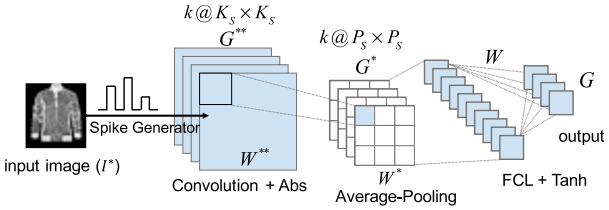

FIGURE 2. The basic computation unit architecture. It consists of three layers, behind the input layer is convolution layer which consists of k kernels, followed by an average-pooling layer and a fully connected layer.

As shown in Fig. 2, each BCU is a simplified three-layer CNNs architecture [20]. In the cascaded framework, the BCU is taken as an image transformator. In detail, each computation unit includes three parts to complete the convolution, pooling (to extract the features from the input images and produce the feature maps) and reconstruction. The first part includes k kernels with kernel size of $K_s \times K_s$, the absolute nonlinearity function (Abs) is used for activating. And the $P_s \times P_s$ average-pooling is used for obtaining spatial invariance while scaling feature maps. The third part, fully connected layer (FCL) performs the classification or reconstruction operations, it takes the extracted feature maps and multiplies a weight matrix following a dense matrix vector multiplication pattern.

We use the $H(\cdot)$ to describe the process of image transformation in a BCU. Assuming that the I^* denotes an input image, a BCU will output a reconstructed output G. This process can be described as

$$\begin{aligned} G = H(I^*) &= \sigma(W \cdot G^* + b) \\ &= \sigma(W \cdot (G^{**} * W^*) + b) \\ &= \sigma(W \cdot (\delta(W^{**} * I^* + b^*) * W^*) + b) \quad (1) \end{aligned}$$

where $H : \mathbb{R}^{C\times W\times H} \rightarrow \mathbb{R}^{C^*\times W^*\times H^*}$ is the transformation function in the BCU, C is the number of chanels of input image, σ indicates the hyperbolic tangent function, δ indicates the Abs function, $*$ represents the convolution operator, b, b^* are bias value (usually not required), and W, W^*, W^{**} represent the weight matrix of each layer, respectively (refer to Fig. 2).

As shown in Fig. 1(a), the cascaded architecture includes three parts. The **Part #1** has M BCUs, which extract features from the input image and produce the reconstructed image for the subsequent subnetworks. The output of first part $F_1 \in \mathbb{R}^{C_1\times W_1\times H_1}$ is generated from the BCU outputs $G_1^i \in \mathbb{R}^{C_1\times W_1\times H_1}$ ($i \in [1, M]$). The number of **Part #2** BCUs is N, and this part takes the outputs of the first part as the inputs to generate the $F_2 \in \mathbb{R}^{C_2\times W_2\times H_2}$. And the final **Part #3** includes P BCUs, the first BCU takes the F_2 as the input to produce the $G_3^1 \in \mathbb{R}^{C_3\times W_3\times H_3}$, and the next BCU utilizes G_3^1 to produce the G_3^2, and so on until $G_3^P \in \mathbb{R}^{\widetilde{C}_3\times \widetilde{W}_3\times \widetilde{H}_3}$ is generated. This cascaded framework is called the "M-N-P" type.

Fig. 1(b) shows a demo example of the typical "3-1-1" cascaded architecture, this cascaded framework includes five BCUs which three BCUs in parallel and two in series to generate the classification output.

B. THE RECONSTRUCTION OPERATION

The reconstruction operation is designed to combine feature maps generated of BCUs and produce the new feature maps to the next part. Given the output $G \in \mathbb{R}^{C\times W\times H}$ generated from a part, a reconstruction transformation $f : \mathbb{R}^{C\times W\times H} \rightarrow \mathbb{R}^{C\times W\times H}$ is applied to aggregate outputs over all BCUs of this part, where C denotes the number of chanels of input image, W and H are the spatial dimensions. The output of k^{th} part is described as

$$F_k = (\alpha_k^1 \cdot G_k^1) \oplus (\alpha_k^2 \cdot G_k^2) \oplus \ldots \oplus (\alpha_k^n \cdot G_k^n) \quad (2)$$

$$\begin{aligned} F_{k+1} &= (\alpha_{k+1}^1 \cdot G_{k+1}^1) \oplus \ldots \oplus (\alpha_{k+1}^m \cdot G_{k+1}^m) \\ &= (\alpha_{k+1}^1 \cdot H_{k+1}^1(F_k)) \oplus \ldots \oplus (\alpha_{k+1}^m \cdot H_{k+1}^m(F_k)) \quad (3) \end{aligned}$$

where G_k^i is the output of the i^{th} BCU of the k^{th} part, and $\oplus$ represents the reconstruction operator which combined with the several original outputs using an element-wise addition to produce a new output. Equation (2) demonstrates the process of generating the output of k^{th} part by reconstruction operation, and α_k^i is the weight coefficient of each BCU output in k^{th} part. The value of α_k^i can be obtained through the training process of neural networks, and it can also be regarded as a set of superparameters.

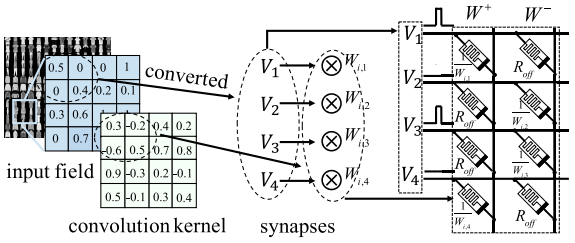

FIGURE 3. Diagram displaying a memristor crossbar that is capable of completing a convolution computation.

The output F_k is treated as input to feed into the $k+1$ part. Equation (3) describes the calculation process of the next part output from the k^{th} part, here the $H(\cdot)$ is the transformation of BCU. The BCU in the same part has the same input, and it is worth noting that the last part does not contain reconstruction operations.

III. MEMRISTOR BASED IMPLEMENTATION

In this section, we first describe the basic computation unit implemented on memristor crossbar arrays. Next, a mapping method of general algorithm with BCUs is described. And then, the BCU split method is presented for relieving the pressure of input terminals. At last, the details about memristor crossbar programmed are introduced.

A. BCU ON MEMRISTOR CROSSBAR ARRAY

As Section II described, the basic computation unit is treated as an image transformator in cascaded framework. The output G of each BCU is generated by multiply-accumulate computation on memristor crossbarp arrays.

The BCU is a simplified three-layer CNNs architecture [20]. The trained weight matrix contains both positive and negative weights, so a synapse converted mapping method is needed to be applied. Assuming that W represents a $N \times M$ weights matrix of original neural networks weight matrix, the first step is that converts W to the $N \times 2M$ matrix, this is for the convenience of obtaining the final computation results through differential output. The W^+ denotes the positive values and W^- represents the negative values. If one synapse weight in original matrix is a positive value, then the value is defined in W^+, and the value of W^- is zero. In contrast, the negative element value is defined in W^- and the W^+ is zero. Similarly, if the arrangement of the memristor in the array corresponds to the converted matrix, the R_{off} takes the place of the zero element.

Fig. 3 demonstrates a simple demo of performing convolution computations by memristor crossbar arrays. The gray information of the field to be convoluted in the input image is converted into a set of voltage values ($V_1, V_2 \ldots$), and the weight of the convolution kernel is mapped to the resistance of the device on memristor crossbar arrays. Therefore, the differential output of W^+ and W^- is the final convolution output.

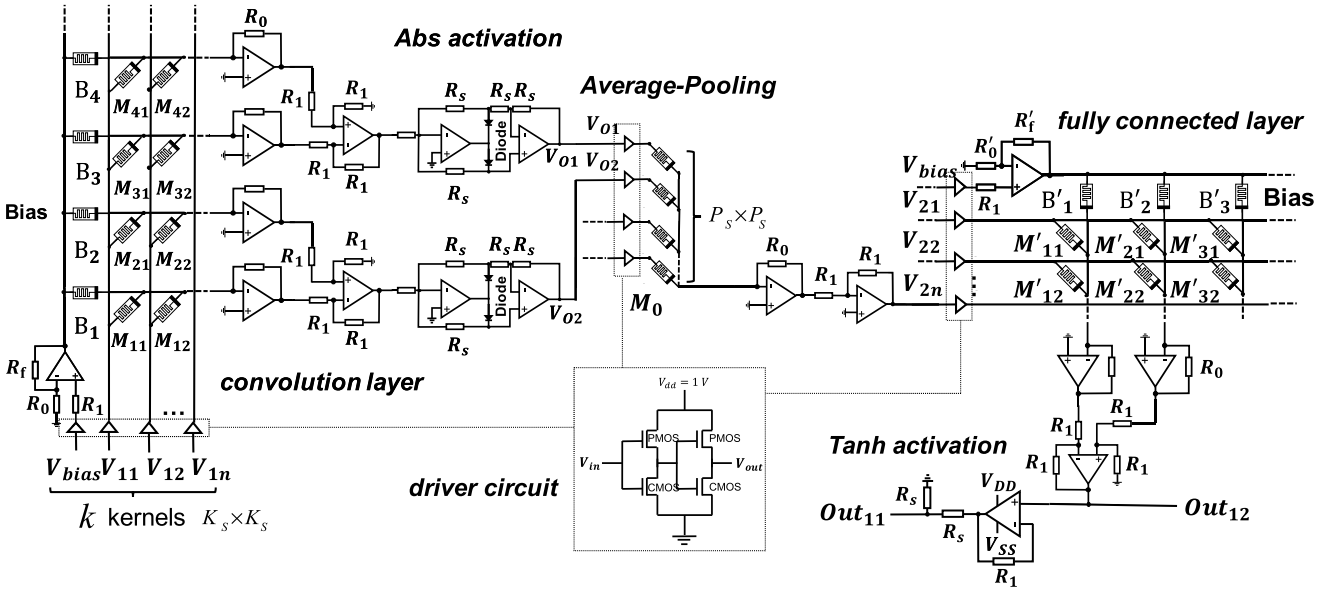

FIGURE 4. Memristor-based basic computaion unit architecture.

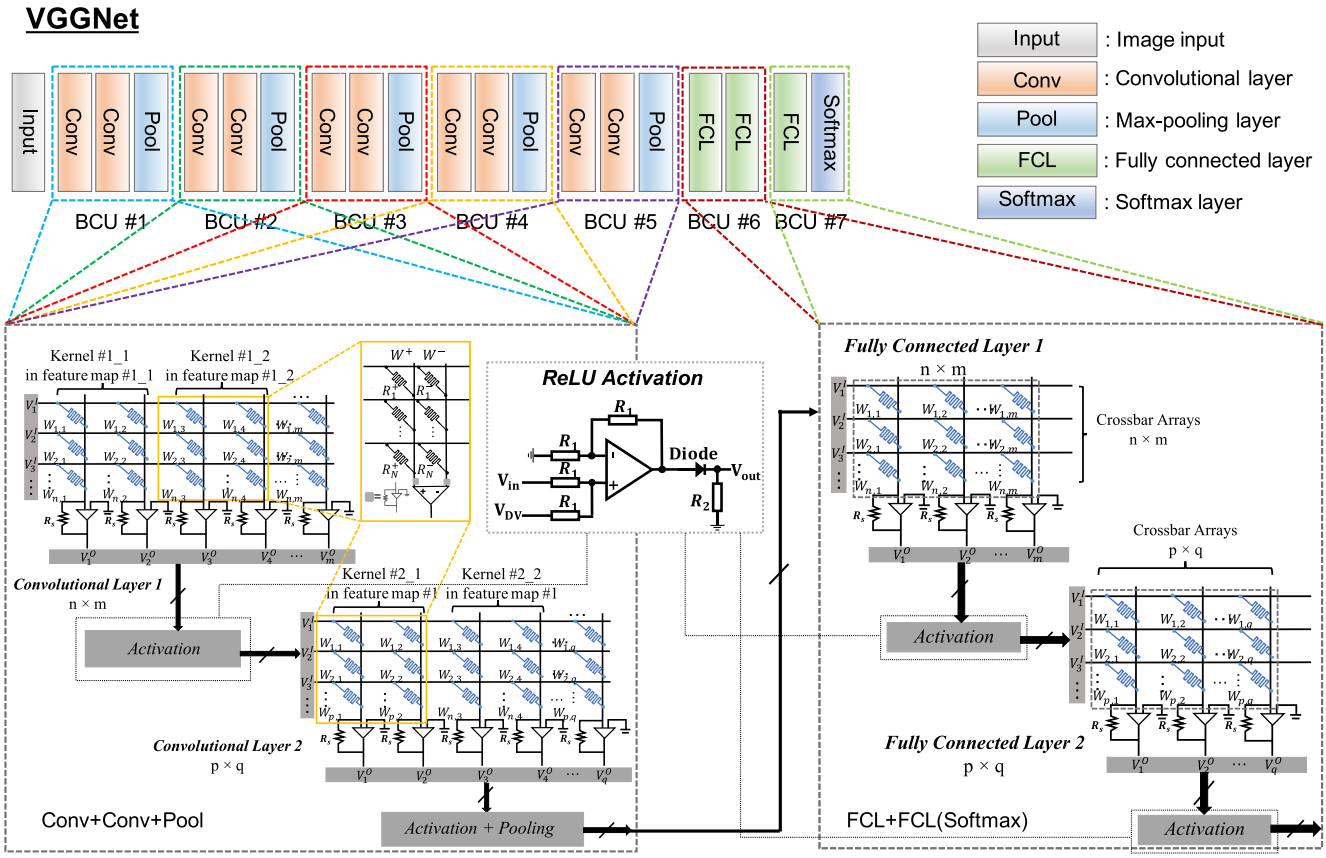

FIGURE 6. A demo example of mapping a VGGnet.

A memristor-based computation unit architecture is shown in Fig. 4. A $K_s^2 \times P$ matrix $M_{(1)}(M_{ij})$ corresponds to all outputs of convolutional kernels (here $P = k \times 2$, two memristors represent one synapse), so the convolution layer consists of some kernels crossbar. The Abs activation module follows the kernels crossbar, it consists of two op-amps and

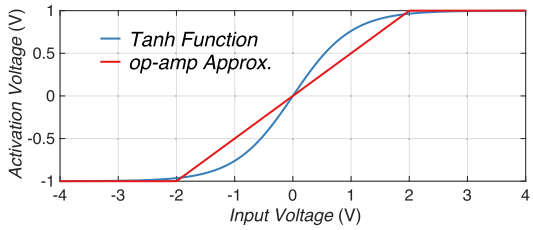

FIGURE 5. Comparison of the tanh activation function and the amplifier alternative.

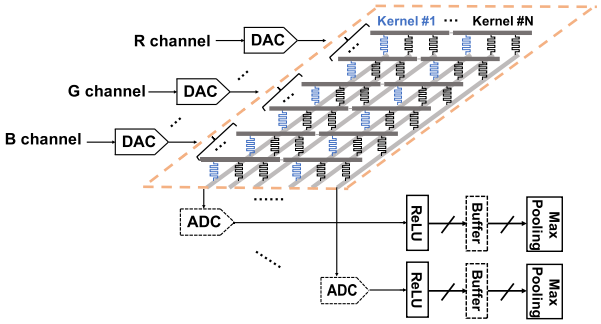

FIGURE 7. A method of multi-channel mapping to convolution kernels.

two diodes. The activation module generates signals (like V_{O1} or V_{O2} in Fig. 4) and sents these signals to the Average-Pooling modules. The pooling module consists some $P_s^2 \times 1$ crossbar which matrix weight is $1/P_s^2$, the multiply-addition calculation is applied to complete the pooling operations. The fully connected layer receives the signals from pooling module, a $L \times N$ matrix $M_{(2)}(M'_{ij})$ corresponds to N neurons and L inputs. The Tanh activation circuit at the end of fully connected layer in Fig. 4 is used to both scale the output voltage and implement the Tanh activation function.

$$Out_{11}(x) = \begin{cases} 1, & x > 2 \\ mx, & |x| \leqslant 2 \\ -1, & x < -2 \end{cases} \approx \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4}$$

There is an alternative calculation to complete the Tanh activation function by referring the Eq. (4). A linear amplifier activation function (with bounded upper and lower voltage rails) matches the Tanh relatively closely, and the simulation results show that this is an effective alternative (in Fig. 5). To obtain the optimal linear, $m = 1/2$ fits the Tanh function in Eq. (4). It should be noted that we use the Out_{11} to cascade the next network, the Out_{12} is treated as the output when we use the single BCU for classification.

To enhance the driving capability of crossbar arrays, driver circuits are needed so that low current communication signals can drive the large resistive crossbar structures, they are placed at each row input on the crossbar. These driver circuits have a finite impedance and errors will occur if the crossbar becomes too large [21]. Driver circuits are shown in Fig. 4, and MOSFET based CMOS inverter circuits is used.

B. GENERAL ALGORITHM MAPPING METHOD

According to what we have described above, the BCU mainly has two crossbar arrays, which are generally used for mapping the convolutional layer and the fully connected layer. When mapping to a general algorithm, such as VGGnet, the second array is mapped to a convolutional layer. The feature maps generated by the first array will be used as input to the second array [22]. Similarly, the first crossbar array can also be mapped to a fully connected layer.

Fig. 6 demonstrate an example of mapping a VGGnet [3]. It consists of seven BCUs, which are cascaded by "Conv+Conv+Pool" and "FCL+FCL(Softmax)", a ReLU activation circuit was designed to implement the nonlinearity of the function. The "Conv+Conv+Pool" BCU performs convolution operations with two arrays, and the output of the second array is sent to the Max-Pooling module which usually implemented using Single-Pole-Double-Throw switches (SPDT) [20]. In the same way, the fully connected layer of VGGnet is implemented by cascading the "BCU #6" and "BCU #7".

Fig. 7 shows the method of mapping each channel in the convolution kernel to the array (taking RGB channel as an example). Analogously, a convolution kernel is mapped into two columns of the array. It is worth noting that each channel is arranged in sequence corresponding to the weight in the convolution kernel. Assuming that the input size of each RGB channel is W×W, then each convolution kernel is mapped to an array size of (W·W·3)×2. The "ReLU" and "Max-Pooling" [20] in the figure can be implemented by circuit, so ADC and Buffer can be omitted.

C. BCU SPLIT METHOD

As we discussed in the previous section, the BCU can generate entire output feature maps in one processing cycle. It means that BCU is a highly parallel systems, it will improve the time efficiency of the classification. However, this calculation method will put pressure on the input terminal. Assumed the $W \times H$ input image, each of the images is converted to a $W \cdot H \times 1$ voltage output vector by DAC as the input dataflow. Each $K_s \times K_s$ field of the images will be sent to the K_i^{convX} module for the convolution computation.

K_i^{convX} indicates that when using the i^{th} convolution kernel to complete the X^{th} convolution computation, K_i^{convX} is the kernel crossbar discussed above. A $K_s \times K_s$ convolution computation of a $W \times H$ image requires a total of $(W - K_s + 1) \times (H - K_s + 1)$ operations. The modules of each convolution kernel are independent, so the K_i^{convX} does not need to be reconfigured.

In order to reduce the pressure on the input terminals, the BCU split method is need to be considered. According to Eq. (1), the input image I^* will be sent to the BCU to complete the convolutional calculation. As we can see, the input I^* can be described as $I^* = I_1^* + I_2^* + \ldots + I_N^*$, so the Eq. (1) can

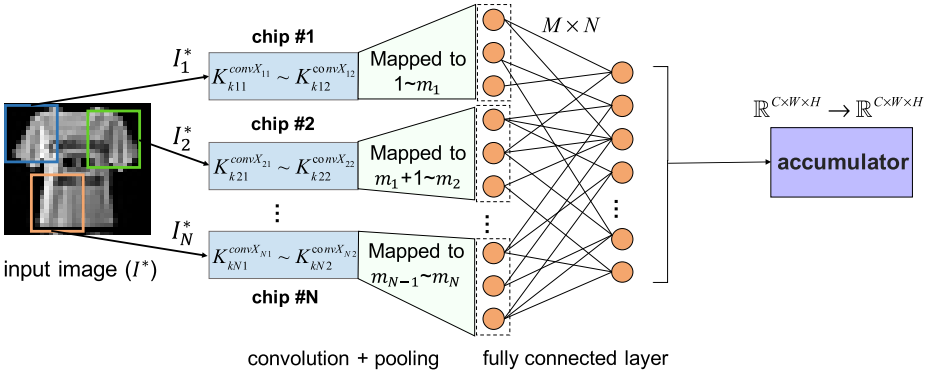

FIGURE 8. The diagram of BCU split method of one part.

be rewritten as

$$\begin{aligned} G &= \sigma(W \cdot (\delta(W^{**} * I^* + b^*) * W^*) + b) \\ &= \sigma(W \cdot (\delta(W^{**} * (I_1^* + \ldots + I_N^*) + b^*) * W^*) + b) \\ &= \sigma(W \cdot (\delta(W^{**} * I_1^* + b^*) * W^*) + b) \\ &\quad + \sigma(W \cdot (\delta(W^{**} * I_2^* + b^*) * W^*) + b) + \ldots \\ &\quad + \sigma(W \cdot (\delta(W^{**} * I_N^* + b^*) * W^*) + b) \\ &= H(I_1^*) \oplus H(I_2^*) \oplus \ldots \oplus H(I_N^*) \end{aligned} \tag{5}$$

where $H : \mathbb{R}^{C\times W\times H} \rightarrow \mathbb{R}^{C^*\times W^*\times H^*}$ is the transformation in the BCU, and $\oplus$ represents the reconstruction operator which combined with the output of different split chip using an element-wise addition, and N is the number of subimages.

Fig. 8 demonstrates how a BCU in one part distributes input to multiple chips. Take the *chip #i* for example, this chip will perform the convolution calculation $K_{i11}^{convX_{i1}} \sim K_{i12}^{convX_{i2}}$ in the portion of I^*. As we mentioned above, the fully connected layer is a process that calculates $W \cdot G^*$ (here G^* can be regarded as $\mathbb{R}^{1\times m_N}$), and the *chip #i* will map the output to the $m_{i-1}+1 \sim m_i$ in G^*. Several $\mathbb{R}^{C\times W\times H}$ outputs are generated by these chips and accumulated by $\oplus$ operation.

D. PROGRAMMED ON MEMRISTOR CROSSBAR

The basic computation unit is presented in Fig. 4, memristor crossbars are utilized to implement the network layer. As the number of cascaded networks increases, many memristor crossbars would be required.

The objective of the programming process is to set each memristor in the crossbar to a specific memristance. In the 0T1R crossbar (refer to Fig. 3 and Fig. 4) programmed process, writing or reading an individual memristor resistance is challenging due to sneak paths. To solve the issue, placing a transistor at each cross-point will ensure that only the memristance of the target memristor is impacting the column current during the programmed process as in a 1T1R crossbar.

The 1T1R programmed circuit is shown in Fig. 9. In this circuit, only the *ctrl* line ($ctrl_i$) corresponding to the target memristor is enabled. This circuit utilizes two Digital-to-Analog converter (DAC) to control a bounded voltage range for successful programming. Assumed the memristance we want to set is w, the DAC is able to convert each floating point weight to a value within a set of 2^B predefined conductivity states, where B corresponds to the bit width of the DAC used. The α is the approximate value 2^k, and the δ is the programming precision, that means the V_T will be less than V_{C1} and greater than V_{C2} when the programmed finished.

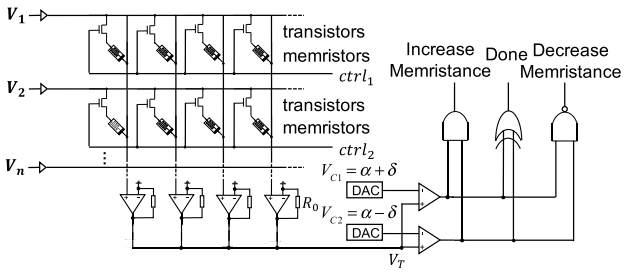

FIGURE 9. Circuit used to program the 1T1R memristor crossbar to target memristance.

The logic circuit determines whether the memristance should be decreased or increased based the output voltage V_T of the two comparators. If the AND gate has a high output, a negative write pulse will be applied to the target memristor, and if the NAND gate has a high output, a positive write pulse will be applied. This programmed process will be repeated until the XOR gate gets a high output, representing a successfully programmed device.

IV. EXPERIMENTS

A. EXPERIMENTAL SETTINGS

All experiments are conducted using an Intel Xeon E7 (2.6 GHz), 16GB DDR4 and a NVIDIA Titan XP graphics card. This work is implemented with the Theano python open-source library to train the neural network models for 100 epochs with a batch size of 128 and a learning rate of 0.05, and HSPICE is also used for some circuit simulations. To reflect the practical application of algorithm performance better, we do not use any data augmentation technologies.

We compare the performance of the cascaded network with some other state-of-the-art architectural units designed

for classification on the MNIST [6], Fashion-MNIST [23], SVHN [24] and CIFAR-10 [25] datasets.

1) SIMULATION PROCESS

The weights of neural network are converted to conductivity values of memristor device by Eq. (6). The C_{max} and C_{min} indicate the maximum and minimum conductances, respectively. The W is the original weights set, and the W_{max} represents the maximum absolute value of the weights set, and the C_i is the conductance of the memristor crossbar array.

$$C_i = \frac{C_{max} - C_{min}}{W_{max}} \cdot |W_i| + C_{min} \tag{6}$$

The 1T1R memristor crossbar array which stores the network weights is used for simulation. The weights obtained from training with the theano are programmed into a crossbar where each wire segment has a resistance of 2Ω, and 5% programmed erros were generated in memristor programming simulation process. The circuit structure in HSPICE is shown in Fig. 4. The process of simulation implemented is illustrated as below.

1) Convert test image samples to voltage (0-1 V).
2) Execute software simulation (used Python) and circuit simulation (used HSPICE) based on step 1.
3) Obtain difference of the potential output (Difference = HSPICE potential - Python output) from Python and HSPICE.
4) Get the labels of test samples from dataset, and determine whether the results of circuit simulation are correct (the label that corresponds the maximum voltage is the output).

The Python (uses theano library) is applied to complete the software simulation, the model weights are converted to memristance, and the Python output is compared with the output of the HSPICE simulation. We select the number which has the maximum output voltage in the fully connected layer as the HSPICE output, the number is compared with the label of the dataset.

2) MEMRISTOR MODEL

We then evaluated the cascaded neural network based on the Ti/AlOx/TaOx/Pt electronic synapse, which was proposed in our previous work [26]. The scanning electron microscopy (SEM) image and the cross-sectional transmission electron microscopy (TEM) is shown in Fig. 10(a).

During the fabrication process, the bottom electrode (Pt, 25nm) and the resistive switching layer (AlOx/TaOx, 5nm/3nm) were deposited by electron beam evaporation, respectively. And the top electrode (Ti, 30nm) was grown by magnetron sputtering. In all, three lithography processes were performed to form the patterns of the three different layers.

The Ti/AlOx/TaOx/Pt memristor proposed shows high uniformity, and over 200 states were achieved by applying triangular pulses (0→0.7 V, 28 KV/s for SET and 0→-1.2 V, 600 KV/s for RESET). The device which has the multilevel

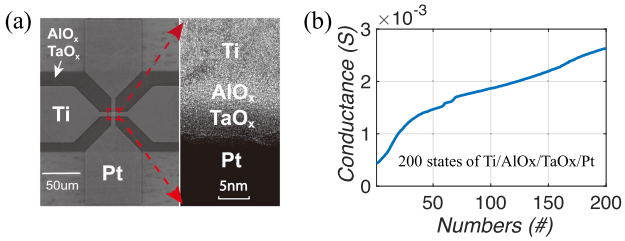

FIGURE 10. (a): SEM image and the Cross sectional TEM image of the Ti/AlOx/TaOx/Pt device. (b): 200 different states for our cascaded network simulation, achieved by applying triangular pulses.

characteristic can be repeatedly programmed to different target resistance states from 1KΩ to 12KΩ, indicating the great potential for neuromorphic computing applications.

Fig. 10(b) demonstrates that 200 states achieved by applying triangular pulses are utilized for circuits simulations, the trained matrix weights are converted to conductivity values that fall within the bounded range of states. Equation (6) introduces the method of weight mapping. In the actual simulation, the closest value is selected from the 200 states.

B. EXPERIMENTS RESULTS

In this section, we compare the cascaded model with other state-of-the-art architectures to verify the classification performance. Circuit simulation experiments are used to verify the correctness of the circuit design and the impact of the device on the recognition accuracy.

1) COMPARISON WITH OTHER STATE-OF-THE-ART ARCHITECTURES

Based on basic computation unit, we test the cascaded network with different architecture on MNIST, Fashion-MNIST, SVHN and CIFAR-10 datasets, the configuration details and Fashion-MNIST accuracy performance is shown in Table 1. The BCU which has 14 kernels with 9×9 kernel size and 2×2 pooling size, and Abs activation function applied can achieve 89.68% Fashion-MNIST accuracy, while Cascaded×14 achieves improvements of 3.18%, 3.44%, 3.57% and 3.86% under 4.15M parameters. The output sizes of the Part#1 and Part#2 are both 20×20. 7-Cascaded×14(4-1-1) provides 93.12% Fashion-MNIST accuracy which uses 9×9 kernel. 8-Cascaded×14(4-2-2) slightly outperforms "4-2-1" (∼0.13%) with 10,000 more parameters. It can be seen that cascaded models can achieve 93.54% accuracy under the 4.15M parameters.

We further compare the cascaded architecture with the building structures of four state-of-the-art models, VGGNet [3], GoogLeNet [4], ResNet [7] and CapsuleNet [27]. The BCU architecture in Table 2 includes 14 kernels, 2×2 average-pool with Abs activation function. A "4-3-2" cascaded type is implemented, "3×3, 5×5, 7×7, 9×9 conv." is in Part #1, "3×3, 5×5, 7×7 conv." is in Part #2, and "9×9, 9×9 conv." in Part #3. VGGNet-16 provides 99.22% MNIST accuracy, 93.35% Fashion-MNIST

TABLE 1. Cascaded architecture for Fashion-MNIST under the 4.15 M parameters. The number before the "Cascaded" is the number of the BCUs. "×14" indicates the cascaded network uses 14 convolution kernels.

Stage	Output Size	6-Cascaded×14 (4-1-1)	7-Cascaded×14 (4-2-1)	8-Cascaded×14 (4-2-2)	9-Cascaded×14 (4-3-2)
	28 × 28	image			
Part #1	20 × 20	3×3, 5×5, 7×7, 9×9 conv, Abs 2×2 average-pool, FC+Tanh			
Part #2	20 × 20	9×9 conv, Abs 2×2 average-pool, FCL+Tanh	9×9, 9×9 conv, Abs 2×2 average-pool, FC+Tanh	9×9, 9×9 conv, Abs 2×2 average-pool, FC+Tanh	3×3, 5×5, 7×7 conv, Abs 2×2 average-pool, FC+Tanh
Part #3	6 × 6	9×9 conv, Abs 2×2 average-pool	9×9 conv, Abs 2×2 average-pool	9×9, 9×9 conv, Abs 2×2 average-pool	9×9, 9×9 conv, Abs 2×2 average-pool
Classifier	1 × 1	FCL, softmax			
Parameters		3.26 M	3.45 M	3.46 M	4.15 M
Accuracy (%)		92.86	93.12	93.25	93.54

TABLE 2. Accuracy rates (%) on the MNIST, Fashion-MNIST, SVHN, and CIFAR-10 datasets.

Models	Parameters	MNIST Acc.	Fashion-MNIST Acc.	SVHN Acc.	CIFAR-10 Acc.
VGGNet-16 [3]	26 M	99.22	93.35	96.86	88.52
GoogLeNet [4]	6.8 M	99.45	93.52	93.74	85.58
ResNet-50 [7]	25.5 M	99.55	94.24	97.31	92.82
CapsuleNet [27]	11.3 M	99.35	93.55	94.76	88.73
Proposed	**4.15 M (<500K per BCU)**	**99.55**	**93.54**	**94.14**	**86.64**

accuracy, 96.86% SVHN accuracy and 88.52% CIFAR-10 accuracy, GoogLeNet achieves 99.45% MNIST accuracy, 93.52% Fashion-MNIST accuracy, 93.74% SVHN accuracy and 85.58% CIFAR-10 accuracy with 6.8M parameters, CapsuleNet provides 99.35% and 93.55% accuracy on MNIST and Fashion-MNIST datasets respectively, and 94.76% SVHN accuracy and 88.73% CIFAR-10 accuracy, while ResNet-50 produces remarkably better Fashion-MNIST accuracy of 94.24%, SVHN accuracy of 97.31% and CIFAR-10 accuracy of 92.82%. However, they are all computationally intensive (approximately 26M parameters of VGGNet). In comparison, 9-Cascaded×14(4-3-2) achieves 99.55% MNIST accuracy, 93.54% Fashion-MNIST accuracy, 94.14% SVHN accuracy and 86.64% CIFAR-10 accuracy under 4.15M parameters. From the table it can be seen that the cascaded network significantly surpasses VGGNet-16 by 0.33% on MNIST accuracy and 0.19% Fashion-MNIST accuracy with 6.26× fewer parameters, and slightly outperforms GoogLeNet 0.02% Fashion-MNIST accuracy, 0.40% SVHN accuracy and 1.06% CIFAR-10 accuracy with 1.64× fewer parameters.

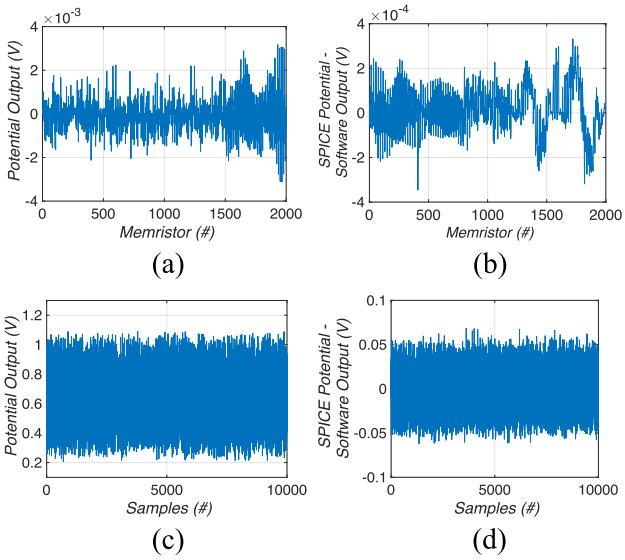

FIGURE 11. (a) and (c): Potential output of kernels and fully connected layer. (b) and (d): Difference of the potential output of kernels and fully connected layer.

2) CIRCUITS SIMULATION PEFORMANCE

Plots in Fig. 11 show the potential output of BCU obtained by the HSPICE simulation and the software simulation output (used Python) in 10,000 test samples of CIFAR-10 dataset.

Fig. 11(a) demonstrates the first 2000 potential outputs of all kernels in one test sample, which takes from 8064 convolution operations applied by 14 kernels. Similarly, Fig. 11(c) shows the potential output of 10,000 samples, each potential is the maximum voltage of 10 outputs in fully connected layer. It can be seen that the outputs of each samples are between 0.20 V to 1.20 V. Fig. 11(b) and Fig. 11(d) depict the difference of the corresponding potential outputs in the crossbars, obtained from HSPICE and the Python software based simulations. From Fig. 11(d) we can observe that the corresponding output result from HSPICE and Python simulations are close (absolute values of the difference of the corresponding values are less than 1×10^{-1}).

Fig. 12(a) and Fig. 12(b) depict the boxplot of simulation difference for the output of kernels and fully connected layers. As shown in Fig. 12(a), the differences are concentrated in the range of -2×10^{-4} to 3×10^{-4}. In Fig. 12(b), the differences are concentrated in the range of -5×10^{-2} to 5×10^{-2}. This is one order of magnitude different for the kernels and fully connected layer output. Simulation experiments demonstrate that the output of the circuit is close to the software simulation, indicating that the circuit implementation is feasible.

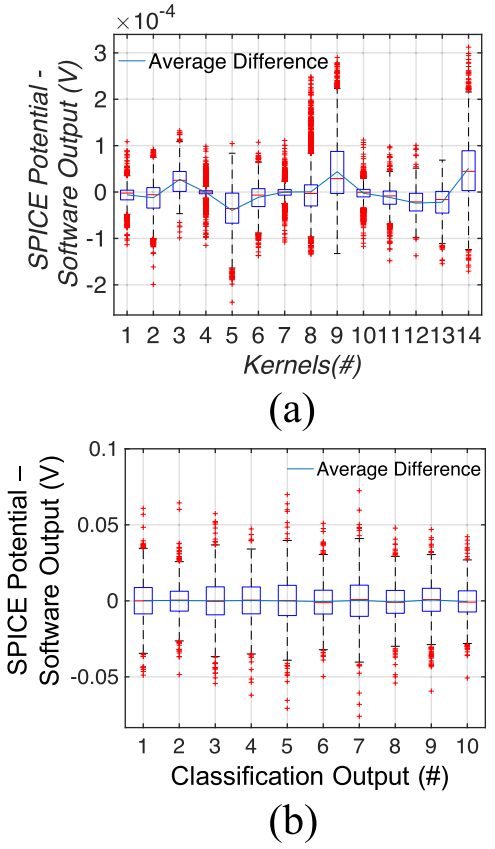

FIGURE 12. Boxplots of simulation difference for the output of (a) kernel and (b) fully connected layers.

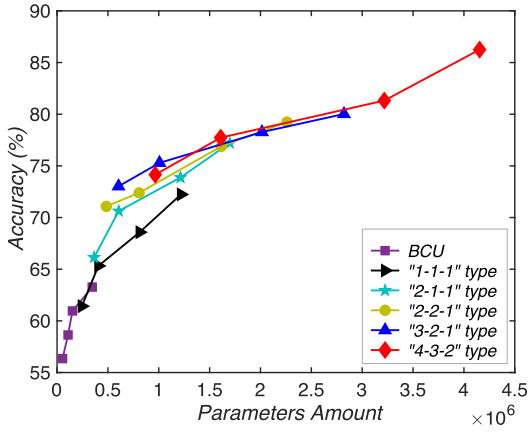

FIGURE 13. Comparison of different cascaded architecture with one BCU on CIFAR-10 dataset based on HSPICE simulation.

Fig. 13 illustrates the relationship between the amount of model parameters and CIFAR-10 recognition accuracy. The amount of convolution kernels in the BCU is between 5 to 32, 9×9 kernel size and 2×2 average pooling are applied. As shown in figure, the BCU can achieve 56.35% accuracy under 0.5M parameters (when the amount of convolution kernels is 32). We tested the five cascaded type for their performance. It can be observed that as the number of parameters increases, the recognition accuracy of the network gradually increases. The "4-3-2" cascaded type provides 86.25% accuracy with 4.15M parameters. It is worth noting that the accuracy of the BCU is better than"1-1-1" cascaded type under 0.5M parameters, the feature information may not be effectively transmitted due to the lack of the cooperation of the parallel structure. The specific reason remains to be studied in the future work.

V. CONCLUSION

In this paper, we propose a memristor-based cascaded framework, which can use several basic computation units in cascaded mode to improve the processing capability of the dataset. We demonstrate the architecture of the BCU and corresponding circuit structure. Based on the proposed BCU architecture, we present a "M-N-P" cascaded specially designed for improving recognition accuracy of the datasets, and we introduce a split method about BCU to reduce pressure of input terminal. Compared with VGGNet and GoogLeNet, the proposed cascaded framework can achieve desired accuracy with fewer parameters. Extensive circuits experiments are conducted show that the circuit simulation results can still provide a high recognition accuracy. For future work, we consider to further evaluate the cascaded framework on other tasks such as object detection.

REFERENCES

[1] J. Gu *et al.*, "Recent advances in convolutional neural networks," *Pattern Recognit.*, vol. 77, pp. 354–377, May 2018.
[2] 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.
[3] K. Simonyan and A. Zisserman. (2014). "Very deep convolutional networks for large-scale image recognition." [Online]. Available: https://arxiv.org/abs/1409.1556
[4] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 1–9.
[5] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and Cooperation in Neural Nets*. Berlin, Germany: Springer, 1982, pp. 267–285.
[6] 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.
[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.
[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.
[9] B. Belhadj, A. Joubert, Z. Li, R. Héliot, and O. Temam, "Continuous real-world inputs can open up alternative accelerator designs," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 1–12, 2013.
[10] S. Park *et al.*, "Nanoscale RRAM-based synaptic electronics: Toward a neuromorphic computing device," *Nanotechnology*, vol. 24, Sep. 2013, Art. no. 384009.
[11] Y. Taur and T. H. Ning, *Fundamentals of Modern VLSI Devices*. Cambridge, U.K.: Cambridge Univ. Press, 2013.
[12] B. Kim, M. Lee, J. Kim, J. Kim, and J. Lee, "Hierarchical compression of deep convolutional neural networks on large scale visual recognition for mobile applications," *Adv. Sci. Technol. Lett.*, vol. 140, pp. 189–194, 2016.
[13] 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.

[14] 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, Jun. 2016.
[15] S. Sun, J. Li, Z. Li, H. Liu, Q. Li, and H. Xu, "Low-consumption neuromorphic memristor architecture based on convolutional neural networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2018, pp. 1–6.
[16] B. Li, Y. Wang, Y. Wang, Y. Chen, and H. Yang, "Training itself: Mixed-signal training acceleration for memristor-based neural network," in *Proc. IEEE 19th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2014, pp. 361–366.
[17] S.-Y. Sun, J. Li, Z. Li, H. Liu, H. Liu, and Q. Li, "Quaternary synapses network for memristor-based spiking convolutional neural networks," *IEICE Electron. Express*, vol. 16, no. 5, 2019, Art. no. 20190004.
[18] S. Kim *et al.*, "NVM neuromorphic core with 64k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous *in-situ* learning," in *IEDM Tech. Dig.*, Dec. 2015, pp. 1–17.
[19] S. Yu, P.-Y. Chen, Y. Cao, L. Xia, Y. Wang, and H. Wu, "Scaling-up resistive synaptic arrays for neuro-inspired architecture: Challenges and prospect," in *IEDM Tech. Dig.*, Dec. 2015, pp. 3–17.
[20] S.-Y. Sun, Z. Li, J. Li, H. Liu, H. Liu, and Q. Li, "A memristor-based convolutional neural network with full parallelization architecture," *IEICE Electron. Express*, vol. 16, no. 3, 2019, Art. no. 20181034.
[21] R. Uppala, C. Yakopcic, and T. M. Taha, "Methods for reducing memristor crossbar simulation time," in *Proc. Nat. Aerosp. Electron. Conf. (NAECON)*, Jun. 2015, pp. 312–319.
[22] C. Yakopcic, M. Z. Alom, and T. M. Taha, "Extremely parallel memristor crossbar architecture for convolutional neural network implementation," in *Proc. Int. Joint Conf. Neural Netw.*, 2017, pp. 1696–1703.
[23] H. Xiao, K. Rasul, and R. Vollgraf. (2017). "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms." [Online]. Available: https://arxiv.org/abs/1708.07747
[24] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," Tech. Rep., 2011.
[25] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009, vol. 1, no. 4, p. 7.
[26] Y. Sun *et al.*, "A Ti/AlO_x/TaO_x/Pt analog synapse for memristive neural network," *IEEE Electron Device Lett.*, vol. 39, no. 9, pp. 1298–1301, Sep. 2018.
[27] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3856–3866.

SHENG-YANG SUN received the B.S. degree in information engineering from the National University of Defense Technology, Changsha, China, where he is currently pursuing the M.S. degree in electronic science and information technology.

His current research interests include neuromorphic computing, memristor modeling, neuromorphic hardware, and deep learning.

HUI XU received the Ph.D. degree in information engineering from the National University of Defense Technology, Changsha, China, in 1995.

He is currently a Professor with the College of Electronic Science, National University of Defense Technology. His current research interests include circuits and signal processing systems.

JIWEI LI received the M.S. degree in electrical engineering from the National University of Defense Technology, Changsha, China, where he is currently pursuing the Ph.D. degree in electronic science and technology.

His current research interests include circuit design and spiking neural networks.

QINGJIANG LI received the Ph.D. degree in electronic science from the National University of Defense Technology, Changsha, China, in 2014.

He is currently an Associate Professor with the College of Electronic Science, National University of Defense Technology. His current research interest includes memristor characterization and applied circuits.

HAIJUN LIU received the Ph.D. degree in information and telecommunication systems from the National University of Defense Technology, Changsha, China, in 2010.

He is currently an Associate Professor with the College of Electronic Science, National University of Defense Technology. His current research interests include the study of memristor and memristive systems, and applications based on two-terminal resistive switches.