“© 2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.”

IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

CKFO: Convolution Kernel First Operated Algorithm with Applications in Memristor-based Convolutional Neural Network

Journal:	*Transactions on Computer-Aided Design of Integrated Circuits and Systems*
Manuscript ID	TCAD-2019-0318
Manuscript Type:	Full Paper-Machine Learning/AI Automation
Date Submitted by the Author:	28-Jul-2019
Complete List of Authors:	Wen, Shiping; University of Electronic Science and Technology of China Chen, Jiadong; University of Electronic Science and Technology of China Wu, Yingcheng; Huazhong University of Science and Technology - Main Campus Zheng, Yan; University of Technology Sydney Cao, Yuting; Hunan University Yang, Yin; Hamad Bin Khalifa University, College of Science, Engineering and Technology Huang, Tingwen; Texas A&MUniversityatQatar, Chen, Yiran; University of Pittsburgh, Department of Electrical and Computer Engineering Li, Peng; University of California Santa Barbara
Keywords:	CNN, Convolution, memristor, weight pruning

SCHOLARONE™
Manuscripts

CKFO: Convolution Kernel First Operated Algorithm with Applications in Memristor-based Convolutional Neural Network

Shiping Wen, Jiadong Chen, Yingcheng Wu, Zheng Yan, Yuting Cao, Yin Yang, Tingwen Huang, *Fellow, IEEE*, Yiran Chen, *Fellow, IEEE*, and Peng Li, *Fellow, IEEE*

***Abstract*—This paper presents a new convolution algorithm: Convolution Kernel First Operated (CKFO), which can solve the problem that the actual calculation is not reduced after pruning the weight of the convolution neural network. According to the convolution algorithm, this paper proposes a simulated memristor implementation of a Convolutional Neural Network (CNN). After that, we use the method of ex-situ training to train CNN in Tensorflow and then download the trained parameters to the Simulink system by compiling the conductance value of memristor to test the proposed simulation model. Finally, the effectiveness of the proposed model is verified. In addition, we prune the weights of CNN and retrain it, then adjust the simulation model according to the parameters after being pruned (remove memristors with the value of zero). We are surprised to find that the convolution layer designed according to the new convolution algorithm can apply the results of the pruned weight without any modification, which is very cumbersome in other memristor based CNN, because the distribution of the pruned weight is irregular. The parameters are reduced by 75.24%, while the accuracy is just reduced by 0.06%.**

***Index Terms*—CNN, Convolution, Memristor, Weight pruning**

I. Introduction

As a kind of feedforward neural networks, convolutional neural network (CNN) contains convolution computation, and works as one of the most representative algorithms of deep learning [1], for CNN has the ability of representation learning and can classify input information according to its hierarchical structure by shift-invariant classification. Therefore, CNNs are also called as Shift-Invariant Artificial Neural Networks. In the 21st century, with the advancement of deep learning theory and the improvement of numerical computing equipment, CNNs have been developed rapidly and applied in computer vision, natural language processing and other fields [1].

The research on convolutional neural networks can be traced back to the Neocognitron model proposed by Kunihiko Fukushima in his papers published in 1979 and 1980 [2]. Neocognitron is a neural network, whose hidden layer is composed of simple-layer and complex-layer to implement one of the earliest deep learning algorithms [3] to extract and screen features, and partially realize the convolution layer [4] and pooling layer [5] in convolution neural network [6]. Traditional convolution based on sliding window derive from the Neocognitron.

Memristor is a circuit element whose conductance can be compiled by voltage pulse. Its conductance persists even after the voltage pulse is turned off. At the same time, memristor can be used to implement multiplication. Since the convolution operation in CNN is composed of multiplication and addition, memristor has vast potential in the implementation of the CNN hardware circuits. In recent years, related methods have been proposed to implement a memristor-based CNN, such as memristor crossbar based CNN [7], parallel memristor crossbar based CNN [8], and memristor-based Full Convolutional Network (FCN) [9]. These works provide some methods for the implementation of memristor-based CNN.

In order to speed up the calculation of convolution in neural networks, the convolution operation was transformed into matrix multiplication [10]. The high-speed processing ability of GPU greatly accelerates the training process of neural network with matrix multiplication [11]. However, neither the traditional convolution based on sliding window nor the convolution in Caffe can reduce the calculation by pruning the weights [12]–[16], because even after the weight is pruned to zero, it still needs to participate in the calculation. To solve this problem, this paper presents a new convolution algorithm named Convolution Kernel First Operation (CKFO). At each time, those feature elements, which multiplied by a kernel element in the traditional convolution calculation, will be multiplied by this kernel element. Do the same for each kernel element, then the final convolution result is obtained by summing up the results of each operation. Because the

This work was supported by the Natural Science Foundation of China under Grant 61673187. This publication was made possible by NPRP grants: NPRP 9-466-1-103 from Qatar National Research Fund. The statements made herein are solely the responsibility of the author[s]. *(Corresponding author: Shiping Wen.)*

S. P. Wen and J. D. Chen, is with School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, Sichuan, 611731, China (wenshiping@uestc.edu.cn). Y. C. Wu is with School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, Hubei, 430074, China (email: yingcheng.wu@hust.edu.cn). Z. Yan is with Centre for Artificial Intelligence, University of Technology Sydney, Australia (yan.zheng@uts.edu.au). Y. T. Cao is with College of Mathematics and Econometrics, Hunan University, Changsha, Hunan, China (yuting.cao@hnu.edu.cn). Y. Yang is with College of Science, Engineering and Technology, Hamad bin Khalifa University, 5855, Doha, Qatar (email: yyang@hbku.edu.qa). T. W. Huang is with Science Program, Texas A&M University at Qatar, 23874, Doha, Qatar (email: tingwen.huang@qatar.tamu.edu). Y. R. Chen is with Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (email: yiran.chen@duke.edu). P. Li is with Department of Electrical and Computer Engineering, University of California, Santa Barbara CA 93106, USA (email: peng.li@tamu.edu).

elements in the kernel are calculated one by one, the elements whose values are pruned to zero can be skipped directly in the calculation process, thus the algorithm reduces the calculation.

With the rapid development of neural networks, people begin to devote themselves to deploying neural networks to mobile devices. Because of the large scale of the network, the memory occupied and the resources consumed by computing are massive, so it is necessary to compress the large-scale network [17]–[19]. At the same time, with the emergence of memristors and thyristor [20], [21], it is possible to design neural networks based on hardware circuits. At present, many design schemes of neural network based on memristor have been proposed [7], [8], [22]–[26], but there exists a problem that the actual calculation is not reduced after weight pruned in convolution neural network based on memristor. In order to reduce the consumption of memristors, the neural network is pruned to remove the unimportant weights, so the corresponding memristors used to store these weights can also be removed. However, due to the irregular distribution of the removed weights, the actual operation is very cumbersome, which can not even be achieved for large-scale neural networks. Fortunately, this problem can be solved by the convolution algorithm proposed in this paper. Section III of this paper describes the convolution circuit based on this algorithm. Section IV implements the memristor-based CNN recognition system. CNN is trained in Tensorflow by ex-situ [27], and then the trained parameters are written into the Simulink system to verify the simulation model. After pruning and retraining the CNN, the kernel elements with the value of zero can be skipped directly, because the kernel elements are written into the memristors one by one for calculation. So the pruning results can be applied to the simulation model without deleting the memristor, and the calculation can be reduced.

This paper focuses on the new convolution algorithm and its application in the CNN implementation based on memristor. The main contributions of this article are listed as follows:

i) A new convolution algorithm is proposed to solve the problem that the calculation is still not reduced after the weights of CNN are pruned.
ii) Based on the new convolution algorithm, a new implementation method of memristor based CNN is proposed.
iii) The pruned weight result of CNN can be applied to the CNN recognition system without any modification to the circuit.

II. A New Algorithm of Convolution in CNN

A. Background

In 1989, Yann LeCun constructed a convolutional neural network for image classification, which was the initial version of LeNet, and the word convolution was used for the first time when its network structure was discussed [28]. The calculation method of convolution is using the convolution kernel as a sliding window to slide on the input feature map. After each sliding, the kernel elements and the currently covered feature map elements are multiplied and then added together, every such operation will get an element of the final output feature map. When the sliding window completes the scanning of the entire input feature map, the output feature map is obtained. Afterward, in the deep learning framework Caffe developed by Yangqing Jia of Berkeley university, a new convolution calculation method was adopted, which converted input feature map and kernel into the unrolled matrix and a column vector, then the sliding window convolution was simplified into matrix multiplication. The specific calculation process is shown in Fig. 1. This method makes Caffe convolution operation on GPU three times faster than conventional convolution operation and makes good use of the parallelism of GPU calculation.

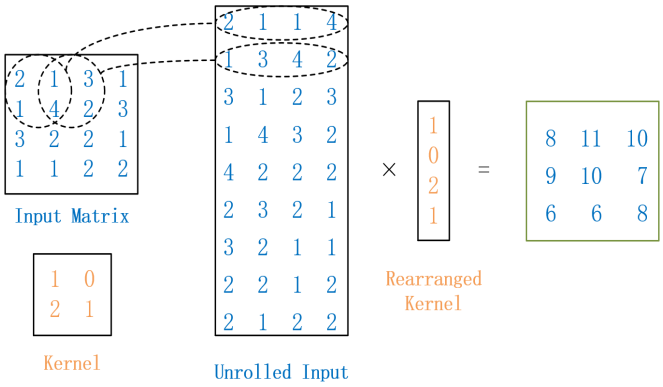

Fig. 1: The Compute Process of Convolution in Caffe

B. A New Convolution Algorithm: Convolution Kernel First Operated (CKFO)

It can be seen from the Caffe algorithm diagram shown in Fig. 1, in matrix multiplication, the first-row element 1 of the unrolled kernel matrix needs to be multiplied with all the elements of the first column of the extended input matrix. Similarly, the second-row element 2 of the unrolled kernel matrix needs to be multiplied with all the elements of the second column of the expanded input matrix. Inspired by it, this paper proposes a new convolution algorithm: Convolution Kernel First Operated (CKFO). CKFO bypasses the conventional requirement for obtaining an element of the output feature map after every operation and transfers the priority of computation to the convolution kernel.

Fig. 2 shows an example of convolution by CKFO. We can see from the picture that the first element $I(1,1) = 1$ of the kernel needs to be multiplied by the first-row element of the unrolled input matrix. Similarly, the second element $I(1,2) = 0$ of the kernel needs to be multiplied by the second-row element of the unrolled input matrix. Repeat this operation for the remaining kernel elements. The final output feature map can be obtained by summing the results of these operations.

CKFO transforms a large number of matrix dot product into multiplications of matrix and single kernel element (As we know, in traditional convolution based on sliding window, there are a lot of matrix dot products). Meanwhile, we can observe from Fig. 2 that there is a sliding window similar to the traditional convolution in the calculation process of this algorithm. But the size of the sliding window of CKFO is larger. In fact, the size of the sliding window is the same as that

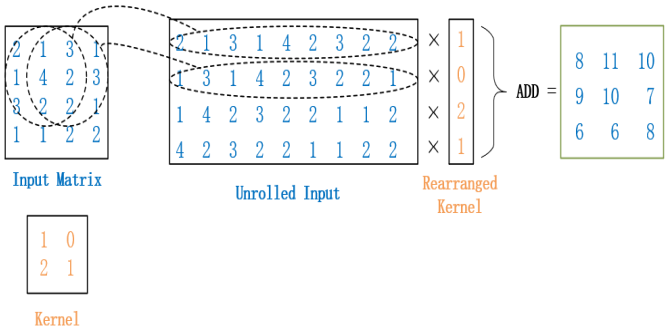

Fig. 2: The compute process of convolution in CKFO

of the output feature map and the sliding times of the sliding window depends only on the number of non-zero elements in the convolution kernel (not the size of the kernel, because the sliding times of the sliding window is equal to the number of non-zero elements in the kernel after the weights are pruned). From this point of view, when the size of the input feature map is larger than that of the kernel, the CKFO algorithm can greatly reduce the sliding times of the sliding window. Such as 512×512 input and 3×3 kernel, the sliding times of traditional convolution window is 510×510, while the sliding times of CKFO is just 3×3. It can be seen that for large size input, our algorithm greatly reduces the sliding times of the window. Furthermore, because of the characteristics of the CKFO algorithm, elements with a value of zero in the kernel can be skipped directly during the calculation. Therefore, this algorithm can reduce the calculation after the weights are pruned.

III. Memristor Convolution Circuit Based on CKFO

This section mainly describes the fundamental principle of a memristor convolution circuit based on CKFO. The circuit for compiling memristor and the circuit for formatting output are from the design in [7], [8], [29]–[31]. To make the circuit concise and intuitive, all programming circuits for the memristor are omitted in this paper. The concrete realization of this circuit is shown in Fig. 3.

Before introducing the circuit, it is necessary to introduce the GTO thyristor [32], because the gate-controlled switch composed of GTOs is the most critical part of the circuit. GTO is a special thyristor, which is a high-power semiconductor device. GTOs, as opposed to normal thyristors, are fully controllable switches which can be turned on and off by their third lead. The connection is accomplished by a positive pulse between the gate and cathode terminal. The disconnection is accomplished by a negative pulse between the gate and cathode terminal. Because of its fast response and simple control method, GTO is used to construct the gate-controlled switch on this circuit.

After introducing the characteristics of GTO, we will introduce the functions of each part of the circuit and the operation of the circuit. Part A is used to store the input feature map row-by-row. A positive pulse is applied to the row that is currently

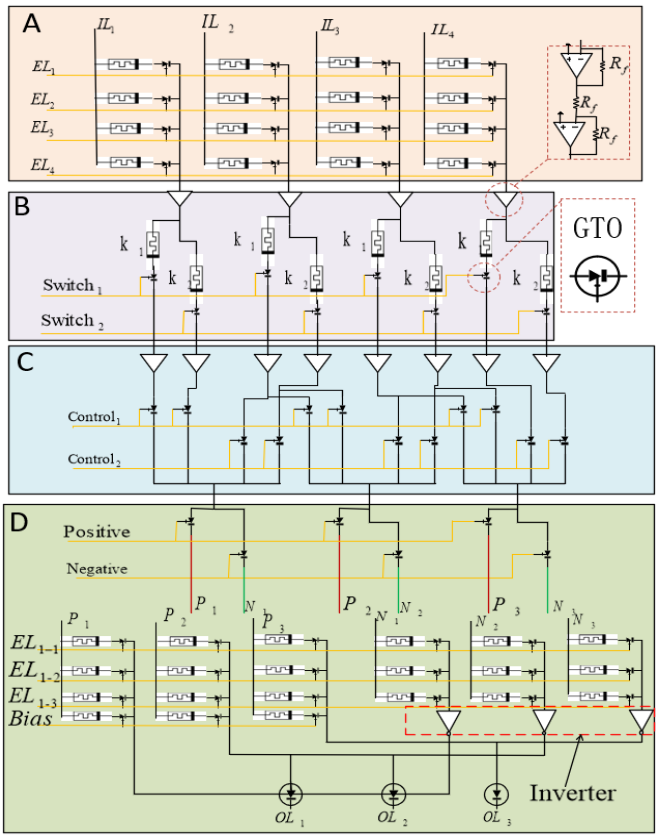

Fig. 3: Memristor convolution circuit based on CKFO. Part A: storing the input feature map; Part B: realizing the dot product of a row of elements and a kernel element; Part C: transferring the calculation result of the previous layer to the corresponding output column; Part D: storing the positive and negative values in the convolution process in two storage matrices respectively.

written, at the same time a negative pulse is applied to other rows. Then, a row of elements in the feature map are written into the memristor of the row to which the positive pulse is applied. Throughout this paper, we employ a voltage threshold memristor model [33]. In this way, the element values of each row are written into the storage matrix. Part B is used to realize the product of a row of elements and a kernel element. The value of the kernel element is stored in memristors. To eliminate the time consumed in storing kernel elements, two lines of memristors are used to store two different convolution kernel elements. When the memristor in one line is participating in the calculation, a new element can be written to another line. To implement multiplication operation, a positive pulse is applied to the EL_i and a negative pulse is applied to others row. Subsequently, a read voltage of 1V is inputted to all IL ports, then the row elements are output to Part B multiplied by the kernel element stored in memristors. Part C transfers the calculation result of the previous layer to the corresponding output column. As described in the previous sections, CKFO is different from the traditional convolution calculation that cannot produce an element of the output feature map by one such operation. However the value of kernel element may be

negative, so Part D stores the positive and negative values of the convolution process in two storage matrices, respectively. A row of memristors at the Bias control port is used to store bias values. When the element in the convolution kernel is negative, its absolute value will be written into the memristor of Part B. At the same time, a positive pulse is applied to the Negative port of Part D, and a negative pulse is applied to the Positive port. If the element value is positive, the opposite operation is performed. After the elements of the convolution kernel are calculated, the positive pulse will be applied to port Bias, and positive pulses are applied to EL_{1-1}, EL_{1-2}, EL_{1-3} in turn so that the final convolution results can be written into the storage matrix row-by-row.

As shown in (1), the ReLU function can be implemented with a diode at the output ports. In practice, a DC bias voltage must be added to the input to balance the diode and GTO leads voltage.

$$f(x) = \begin{cases} 0, & x < 0; \\ x, & x \geq 0. \end{cases} \tag{1}$$

Take the 4×4 input feature map and the 2×2 convolution kernel in Fig. 2 as an example, the control logic of the circuit is shown in Fig. 4, in which the process of writing the calculation results into the memristor is omitted. As the kernel elements in this example are not negative, the storage matrix that stores negative values in Part D is not used.

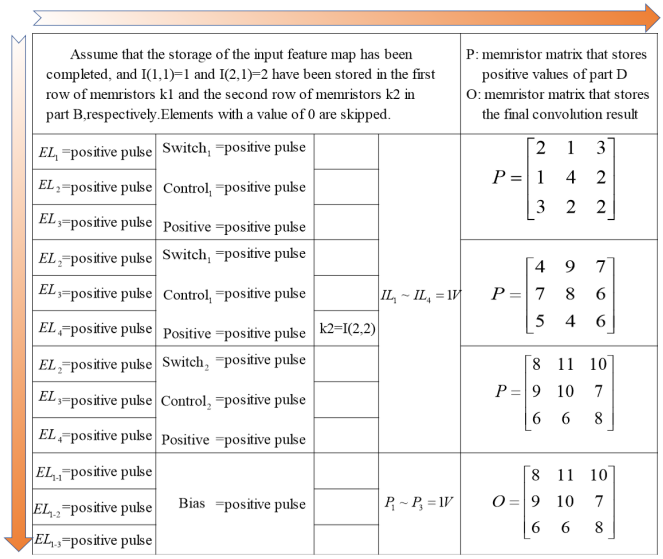

Fig. 4: The operate process of convolution circuit based on CKFO. In this figure, I denotes the kernel matrix, $k_2 = I(2,2)$ express writing $I(2,2)$ into memristors k_2 in Part B, $Y = 1V$ represents $1V$ read voltage is applied to port Y, $X = positive\ pulse$ stands for positive pulse be applied to port X.

IV. CNN Recognition System

This section describes how this memristor-based CNN recognition system operates in detail, in which the first and second convolution layers are designed based on the CKFO. The recognition algorithm is organized according to the flowchart in Fig. 5. The network structure is the same as LeNet except that the convolution order of the input feature graph is different in the second convolution layer.

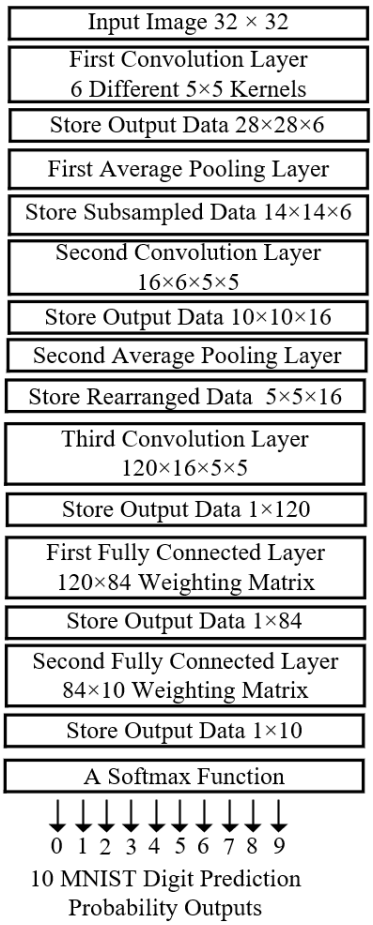

Fig. 5: Flowchart describing the CNN recognition system. The input image of 32×32 in the first layer is obtained by zero filling around the image of 28×28.

A. First Convolution Layer

In this layer, an input image will be convoluted with six different 5×5 kernels, and the final output is a data array which has a size of $28 \times 28 \times 6$. The memristor-based circuit for performing this operation is shown in Fig. 6 in which the circuit is used to produce one output feature map. To realize the first convolution layer, six such circuits are needed. x_1-x_{32} represents the row-by-row input of the feature map stored in the upper layer. The elements in each row of the feature map are multiplied by a kernel element stored in the memristor, and the calculation results are written into the positive or negative value storage matrix. After the calculation of all kernel elements are completed, the final convolution results can be obtained by reading the positive storage matrix and the negative storage matrix row-by-row.

B. First Average-pooling Layer

Following the first convolution layer, an average-pooling operation is performed on the six generated maps, and the final output is a data array that has a size of $14 \times 14 \times 6$. This is just a simple convolution operation in which the convolution kernels

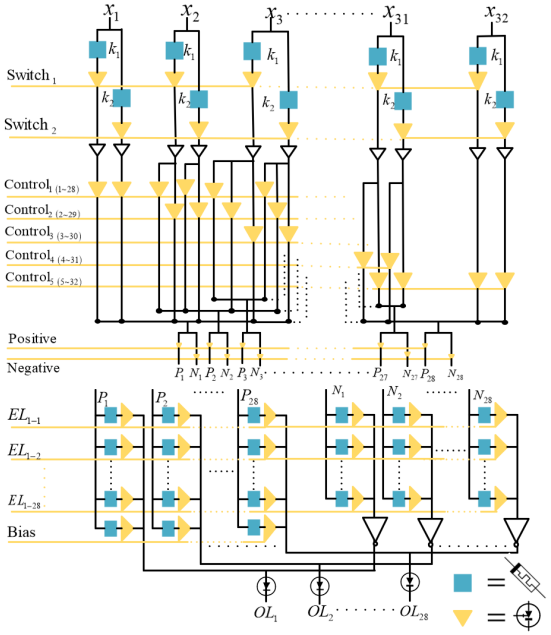

Fig. 6: Circuit used to perform the convolution operations for the first convolution layer.

applied to each feature map are shown in (2). Therefore, the average pooling operation is realized by convolution operation as shown in Fig. 7 in which the circuit is used to produce one output feature map. To realize the first average-pooling layer, six such circuits are needed. The positive pulse is applied to the control terminals of the two rows in turn. Meanwhile, a read voltage of 1V is applied to all columns. Average pooling can be accomplished with only fourteen such operations. In this way, the efficiency of pooling can be greatly improved.

$$K = \begin{bmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix}. \quad (2)$$

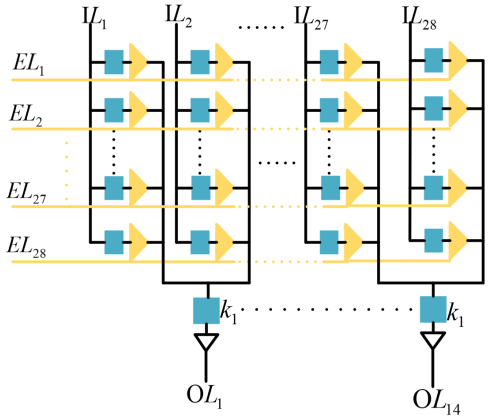

Fig. 7: Circuit used to perform the smoothing operation for the first average pooling layer.

C. Second Convolution Layer

Following the average pooling layer is another convolution layer that is similar to the first convolution layer. However, there are six input maps instead of one and sixteen output maps instead of six. Therefore, the size of the output data array generated by this layer is $10 \times 10 \times 16$. According to the principle of convolution calculation, after the six channels of the input feature map are convoluted with the 5×5 size kernel respectively, the results are summed to obtain the output. The circuit based on this method can refer to the first convolution layer. Two memristor-based storage matrices which store positive and negative values are added to each circuit that convolves one channel respectively, therefore the result of the convolution will be obtained by summing the stored values of these storage matrices. However this will consume a large amount of the memristors. To reduce the consumption of the memristors, as shown in Fig. 8, we have improved the circuit for performing convolution operation, which is used to produce an output feature map. To realize the second convolution layer, sixteen circuits are needed.

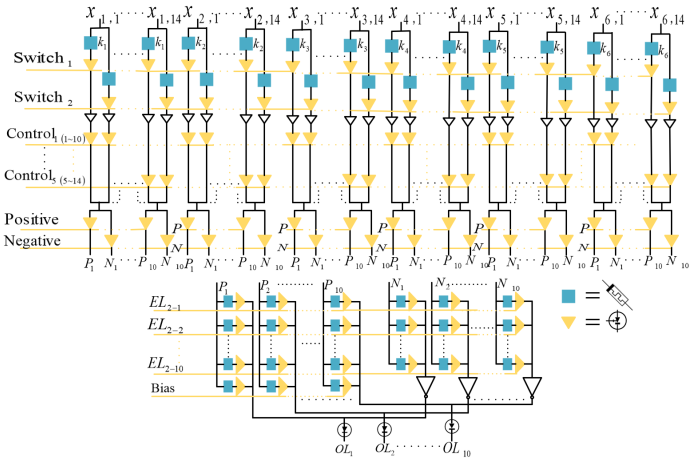

Fig. 8: Circuit used to perform the convolution operation for the second convolution layer. $x_{N,1}$-$x_{N,14}$, $N \in (0, 6]$, represent the row-by-row input of the N_{th} in the six feature maps stored in the upper layer.

D. Second Average-pooling Layer

Following the second convolution layer, another average pooling layer is applied (that will further reduce the size of data array). The circuit of this layer is similar to the first average-pooling layer. The difference is that the input feature size is $10 \times 10 \times 16$ instead of $28 \times 28 \times 6$, and the output data array size is $5 \times 5 \times 16$. The concrete circuit is shown in Fig. 9 in which the circuit is used to produce one output feature map. To realize the second average-pooling layer, sixteen such circuits are needed.

E. Third Convolution Layer

Following the second average pooling layer, the third convolution layer is applied. As mentioned in the previous section, CKFO is suitable for the situation where the size of the input feature map is much larger than that of the kernel. But in this layer, they are equal in size, so the implementation of this layer circuit is based on the memristor crossbar in [7]. To facilitate

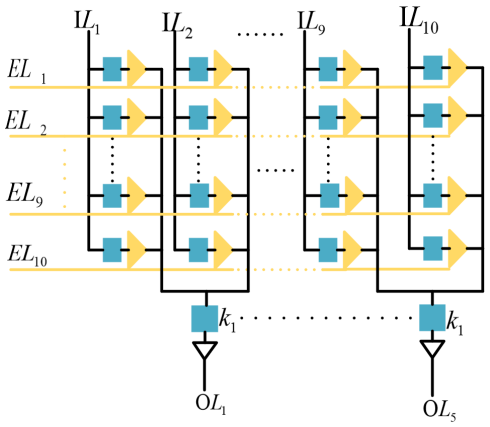

Fig. 9: Circuit used to perform the smoothing operation for the second average pooling layer.

the design of the circuit of this layer, the $5 \times 5 \times 16$ data array outputted by the previous layer is rearranged into a column vector of 400×1. At the same time, the convolution kernel of this layer $120 \times 16 \times 5 \times 5$ is rearranged to 120×400. Therefore, the convolution operation is converted into simple matrix multiplication and addition, and the size of the output data array is 1×120. Since the bias only participates in the addition operation, just one-row memristor is used to store the absolute offset value, and the voltage orientation of the input x_β indicates whether the offset value is positive or negative. The concrete circuit is shown in Fig. 10.

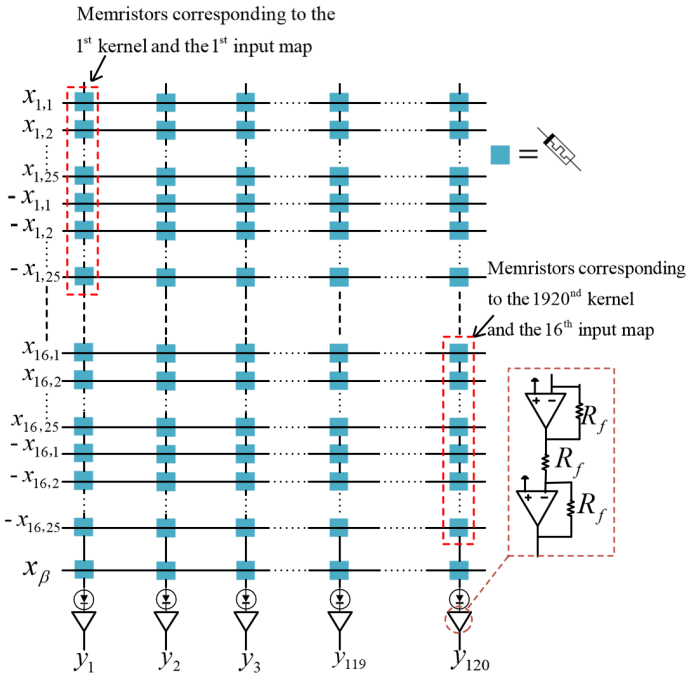

Fig. 10: Circuit used to perform the convolution operations for the third convolution layer.

F. Fully Connected Layer

The implementation of the fully connected layer is similar to that of the third convolution layer, where the calculation consists of matrix multiplication and addition. So the structure of the circuit is similarly. The input of the first fully connected layer is a 1×120 column vector and the output is a 1×84 row vector. The input to the second fully connected layer is a row vector of size 1×84, and the output is a row vector of 1×10, which will be the input of the last layer.

V. Simulation Results

At the first step, CNN is trained completely in software to generate the weights and kernel values that will be programmed into the circuit system designed in the previous section. Subsequently, the trained model is pruned by using the method in [15] and then re-trained. The training data-set comes from the MNIST handwritten digit database. The change curve of accuracy and loss are shown in Fig. 11.

The next step in the simulation process is to test the accuracy of the CNN recognition system described in the previous section. After the parameters of the unpruned network being written into the CNN recognition system, a testing accuracy of 98.43% was achieved. While the parameters of the pruned network were written into the CNN recognition system, a testing accuracy of 98.37% was achieved.

In the process of pruning and retraining the network, the change curve of the pruning ratio is shown in Fig. 12. In Table I, we also count the pruning ratio of each layer of network and the total pruning ratio of the network. The parameters in the table represent the kernel elements in the convolution layer or the weights in the full connection layer. The calculation method of the pruning ratio is shown in (3). A is the number of parameters after being pruned and B is the number of parameters before being pruned. As can be seen from the table, when the overall parameters were pruned by 75.24%, the accuracy was just reduced by 0.06%.

During the experiment, when adjusting the circuit according to the result of pruning and removing the memristors at the corresponding position where the weights were pruned to zero, we were surprised to find that the circuit of the first convolution layer and the second convolution layer designed based on CKFO need not make any modification. Because the kernel elements in the circuit were input one by one and calculated one by one, the kernel elements with a value of zero can be skipped directly. Meanwhile, in the case of channel pruning, circuit adjustment is also simple by Just deleting the convolution circuit of the corresponding channel. But in the third convolution layer and full connection layer based on the memristor crossbar, the adjustment of the circuit is very complicated, because the distribution of the pruned weights is irregular. Moreover, neither the traditional convolution based on sliding window nor the convolution in Caffe can reduce the calculation by pruning the weights, because the kernel elements which were pruned to zero still needed to participate in the calculation. While in CKFO, the kernel elements with the values of zero can be skipped directly, so the calculation can be reduced.

$$Ratio = 1 - \frac{A}{B} \tag{3}$$

TABLE I: Comparison of the Number of Parameters in Unpruned Network and Pruned Network.

	Number of Parameters Before Prune	Number of Parameters After Prune	Prune Ratio	Total Prune Ratio
First Convolution Layer	150	112	25.33%	75.24%
Second Convolution Layer	2400	1799	25.04%	
Third Convolution Layer	48000	12421	74.12%	
First Fully Connected Layer	10080	816	91.90%	
Second Fully Connected Layer	840	70	91.67%	

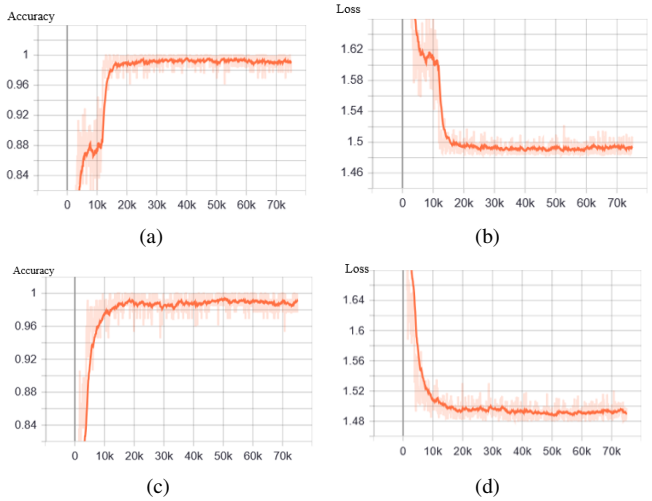

Fig. 11: The training accuracy and loss curves when training the CNN network on the software. The abscissa is the number of training steps, and the ordinate is the accuracy and loss of training. Unpruned pruning: (a) (b); Pruned network: (c) (d)

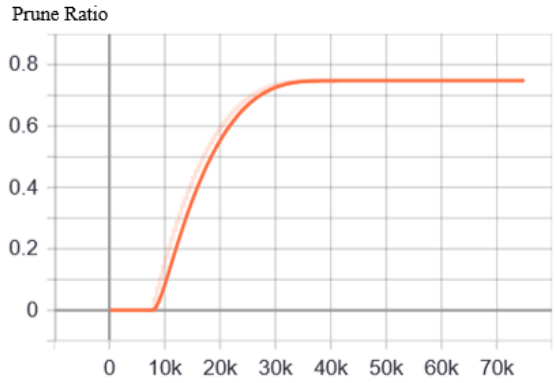

Fig. 12: Circuit used to perform the convolution operations for the third convolution layer.

VI. Conclusion

This paper presents a new convolution algorithm. Compared with the traditional convolution based on sliding window, the size of the window is larger and the sliding times of the window are smaller. Moreover, this algorithm solves the problem that even if the parameters are pruned but the calculation still can not be reduced in the existing convolution calculation method.

Subsequently, according to this algorithm, this paper designs a complete hardware circuit of the CNN recognition system based on memristor and verifies the effectiveness of the circuit by simulation. In the simulation process, we also prune the weight of the network in software and modify the hardware circuit according to the results of pruning. We are surprised to find that when the pruning ratio of the network parameters reaches 75.24%, the accuracy is just reduced by 0.06%. At the same time, the results of pruned weight can be applied to the hardware circuits of the first convolution layer and that of the second convolution layer without any modification to the circuits.

References

[1] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, and G. Wang, "Recent advances in convolutional neural networks," *Computer Science*, 2015.
[2] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.
[3] Schmidhuber and Jrgen, "Deep learning in neural networks: An overview," *Neural Netw*, vol. 61, pp. 85–117, 2015.
[4] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *The handbook of brain theory and neural networks*. MIT Press, 1998, pp. 255–258.
[5] A. Giusti, D. C. Cireşan, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Fast image scanning with deep max-pooling convolutional neural networks," in *2013 IEEE International Conference on Image Processing*. IEEE, 2013, pp. 4034–4038.
[6] Y. Lecun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *IEEE International Symposium on Circuits Systems*, 2011.
[7] M. Z. Alom, "Memristor crossbar deep network implementation based on a convolutional neural network," in *International Joint Conference on Neural Networks*, 2016.
[8] C. Yakopcic, M. Z. Alom, and T. M. Taha, "Extremely parallel memristor crossbar architecture for convolutional neural network implementation," in *International Joint Conference on Neural Networks*, 2017.
[9] S. Wen, H. Wei, Z. Zeng, and T. Huang, "Memristive fully convolutional network: An accurate hardware image-segmentor in deep learning," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 5, pp. 324–334, 2018.
[10] A. Vasudevan, A. Anderson, and D. Gregg, "Parallel multi channel convolution using general matrix multiplication," in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2017, pp. 19–24.
[11] J. D. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips, "Gpu computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.
[12] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *Fiber*, vol. 56, no. 4, pp. 3–7, 2015.
[13] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.
[14] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 2074–2082.
[15] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.
[16] H. Mao, H. Song, J. Pool, W. Li, X. Liu, W. Yu, and W. J. Dally, "Exploring the granularity of sparsity in convolutional neural networks," in *Computer Vision Pattern Recognition Workshops*, 2017.
[17] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *International Conference on Neural Information Processing Systems*, 2012.

[18] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, H. Miao, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Acm/ieee International Symposium on Computer Architecture*, 2016.

[19] P. Chi, S. Li, C. Xu, T. Zhang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Acm/ieee International Symposium on Computer Architecture*, 2016, pp. 27–39.

[20] I. Vourkas and G. C. Sirakoulis, "A novel design and modeling paradigm for memristor-based crossbar circuits," *IEEE Transactions on Nanotechnology*, vol. 11, no. 6, pp. 1151–1159, 2012.

[21] I. Ebong and P. Mazumder, "Self-controlled writing and erasing in a memristor crossbar memory," *IEEE Transactions on Nanotechnology*, vol. 10, no. 6, pp. 1454–1463, 2011.

[22] S. Wang, Y. Cao, T. Huang, and S. Wen, "Passivity and passification of memristive neural networks with leakage term and time-varying delays," *Applied Mathematics and Computation*, vol. 361, pp. 294–310, 2019.

[23] Y. Cao, Y. Cao, S. Wen, Z. Zeng, and T. Huang, "Passivity analysis of reaction-diffusion memristor-based neural networks with and without time-varying delays," *Neural Networks*, vol. 109, pp. 159–167, 2019.

[24] X. Xie, S. Wen, Z. Zeng, and T. Huang, "Memristor-based circuit implementation of pulse-coupled neural network with dynamical threshold generator," *Neurocomputing*, vol. 284, pp. 10–16, 2018.

[25] S. Wen, X. Xie, Z. Yan, T. Huang, and Z. Zeng, "General memristor with applications in multilayer neural networks," *Neural Networks*, vol. 103, pp. 142–148, 2018.

[26] X. Xie, L. Zou, S. Wen, T. Huang, and Z. Zeng, "A flux-controlled logarithmic memristor model and equivalent circuit," *Circuits, Systems, and Signal Processing*, vol. 38, pp. 1452–1465, 2019.

[27] R. Hasan, C. Yakopcic, and T. M. Taha, "Ex-situ training of dense memristor crossbar for neuromorphic applications," in *IEEE/ACM International Symposium on Nanoscale Architectures*, 2015.

[28] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[29] S. Wen, S. Xiao, Y. Yang, Z. Yan, Z. Zeng, and T. Huang, "Adjusting the learning rate of memristor-based multilayer neural networks via fuzzy method," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 38, no. 6, pp. 1084–1094, 2019.

[30] S. Wen, R. Hu, Y. Yang, Z. Zeng, T. Huang, and Y.-D. Song, "Memristor-based echo state network with online least mean square," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 99, no. 0, pp. 1–10, 2018.

[31] S. Wen, H. Wei, Y. Yang, Z. Guo, Z. Zeng, T. Huang, and Y. Chen, "Memristive lstm networks for sentiment analysis," *IEEE Transactions on Systems, Man and Cybernetics: Systems*, vol. 99, no. 0, pp. 1–11, 2019.

[32] P. B. Shah, B. R. Geil, M. E. Ervin, T. E. Griffin, S. Bayne, K. A. Jones, and T. R. Oldham, "Advanced operational techniques and pn-pn-pn structures for high-power silicon carbide gate turn-off thyristors," *Power Electronics IEEE Transactions on*, vol. 17, no. 6, pp. 1073–1079, 2002.

[33] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano Letters*, vol. 10, no. 4, pp. 1297–1301, 2010.