A Parallelization method of Inception architecture based on array processor

Xiaoyan Xie*, Zhuolin Du*, Chuanzhan Hu*, Kun Yang*, Anqi Wang*

*Xi'an University of Posts and Telecommunications, Xi'an, China

E-mail: dzl_w_emails@126.com Tel: +86-29-15829665742

***Abstract*— The Inception architecture proposed to GoogLeNet has the characteristics of few parameters, strong expression ability, and low degree of overfitting, which makes it possible for deployment of convolutional neural network (CNN) in mobile or embedded terminals with limited resources. In order to support the parallelling reconfigurable process of 28 × 28 and 32 × 32 image recognition with 1 × 1, 3 × 3, and 5 × 5 convolution kernels on a 4 × 4 processing elements (PE) array, this paper converts the input image into a dimensional arrays, reducing the frequency of hardware accessed during the convolution calculating. By analyzing the data dependency among convolution and pooling operations in the network, an overlapping window data reuse scheme is proposed, which reduces the number of pixels loaded by external memory by 30%. On the array processor (DPR-CODEC) platform developed by the project team, the proposed method was verified with Minist and Cifar-10 functional testing data sets. The experimental results show that, at the operating frequency of 123MHz, compared with the scheme without preprocessing, the preprocessing hardware access overhead is reduced to 45%, the data reuse rate of convolution calculation reaches 66.7%, and the operating power consumption is 6.395 W, the power per watt is 0.176, and the performance is significantly improved compared to the FPGA version.**

***Keywords*— Convolutional neural network,Inception module,Parallelization,Reconfigure**

I INTRODUCTION

With the widespread application of CNN in the field of artificial intelligence, deep convolutional neural networks (Deep Convolutional Neural Network, DCNN) have become the development trend with the deeper layers and higher accuracy [1]. However, with the deepening of the network layer, a large amount of intermediate data is generated during the training process, the probability of overfitting will also increase, and the calculation complexity will increase [2], making the CPU time cost and GPU power consumption cost become The bottleneck of its application, not to mention deployment on embedded devices with limited resources. The existing DCNN-based smart applications are all implemented in cloud computing. On the one hand, this design will greatly increase the load of data communication, on the other hand, it cannot provide effective protection for the user's data privacy. In recent years, academia has been trying to reduce the complexity of its network while ensuring the accuracy of DCNN calculations. The random sparse connections used in traditional methods to reduce the complexity of the network lead to a significant reduction in the calculation accuracy of CNN when the computer hardware and software are implemented. For this reason, Christian Szegedy proposed GoogLeNet [3], This network adds a large number of 1 × 1 convolution calculations before performing 3 × 3 and 5 × 5 convolution operations to reduce the dimension of the network and reduce the number of parameters to 6.8M and instead of using the average pool of fully connected network layer will increase the accuracy of 0.6%, bringing additional capabilities, but also makes it possible to realize the neural network on a hardware platform of limited resources.

Field Programmable Gate Array (FPGA) has a regular parallel structure and can be flexibly configured, which is very conducive to the research and development of CNN hardware acceleration. Due to the limited hardware resources of FPGA, it is usually difficult to map the entire CNN onto a single chip, so most FPGA-based implementations use a layer-by-layer mapping scheme based on serial execution between layers [1, 4-5], and by sharing the same processor cluster (Process Element Group, PEG) to reduce area overhead. And such a design requires an on-chip controller to switch between layers, and adjust the iteration time and the data address. Generally, the calculation method of the shared processor cluster cannot be reconfigured at runtime, otherwise the design difficulty and

hardware overhead will be greatly increased [1]. To solve this problem, [6] proposed a FPGA available resources into a plurality of processors a method, each processor for different size convolutional layer CNN customized, thereby improving resource utilization. But for the Inception architecture, this design lacks specific optimizations, such as data reuse between branches and layers. This method often requires more bandwidth, and because of the on-chip bandwidth, it cannot effectively improve performance. [7] proposed a hierarchical processing, but this approach relies on the server requires one hand, on the other hand require more suitable communication mechanisms and protocols, large development costs and causes a problem of heterogeneous data. Reference [8] merges the processing of multiple CNN layers and allows intermediate data to be cached between adjacent layers, thereby reducing off-chip data transmission. However, on the one hand, this mechanism may severely limit the performance of the chip and increase the burden on the chip buffer, on the other hand, it lacks the optimal use of computing resources.

Because the data locality of different convolution kernels in the Inception architecture is very different, the hierarchical mapping usually leads to insufficient utilization of the DSP (Digital Signal Process) and memory access bandwidth problems in the FPGA. This paper uses the dynamic reconfigurable array processor (DPR-CODEC) provided by the project team to study the parallel model of the Inception architecture, which can effectively combine the flexibility of general-purpose processors and the efficiency of dedicated hardware. By analyzing the potential parallelism in the network and mining the data dependencies between operations such as convolution and pooling, a data reuse scheme with overlapping windows is proposed, thereby reducing the overhead of hardware resources. Based on the 4 × 4 PE array, convolution kernels with sizes of 1 × 1, 3 × 3, and 5 × 5 are realized respectively, and reconstruction reconstruction of images with sizes of 28 × 28 and 32 × 32 is supported. The results show that, on the Virtex-6 field programmable gate array connected to the BEE4 platform, the proposed reconfigurable scheme has greatly improved the resource utilization rate and algorithm operation efficiency.

II CHARACTERISTIC ANALYSIS OF INCEPTION MODULE ALGORITHM

2.1 INCEPTION MODULE

In multiple versions of the Inception Module, Incepetion V1 is the main module that makes up GoogLeNet, as shown in Figure 1. This structure draws on the ideas of Network In Network [9], uses Inception Module as a standardized basic network, and builds a large-scale network after repeated stacking. Inception V1 uses fewer parameters to achieve the effect of a deeper number and stronger expression in the traditional model. By using the global average pooling layer to replace the fully connected layer, the amount of intermediate calculation parameters is greatly reduced (in the traditional CNN network, the amount of fully connected layer parameters accounts for almost 90%), making model training faster and reducing overfitting. In addition, the Inception Module designed in Inception V1 improves the efficiency of parameter utilization. On the one hand, 1 × 1 convolution is used to achieve dimensionality reduction. On the other hand, convolution and aggregation can be performed simultaneously on multiple sizes. By constructing a dense block structure to approximate the optimal sparse structure, the purpose of improving performance without significantly increasing the amount of calculation is achieved.

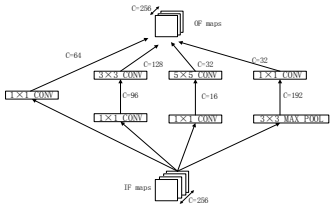


Fig 1 Inception Module

In the convolution layer, the convolution operation highly abstracts the input image to extract image features, that is, feature maps. The low-level feature map can obtain the detailed information of the input image (such as edges, colors, etc.), and the high-level feature map gradually obtains the overall characteristics of the input image (such as shape, contour, etc.). The sub-layered features used in convolution make it possible to achieve good recognition results. The convolutional layer receives n feature maps as input and generates m output feature maps. Each input image is convolved by multiple k × k shift windows, then each convolution result is added to the offset value, and the pixel value is limited to a reasonable range using a suitable nonlinear activation function. Commonly used activation functions include tanh, sigmoid and ReLU. Each element of the output image is calculated by Equation 1.

$$x_j^l = f(\sum_{i \in M_j} x_i^{l-1} * k_{ij}^l + b_j^l) \qquad (1)$$

Where f(x) is the activation function, x_j^l is the jth feature map of the lth layer, M_j represents all input feature maps, k_{ij}^l and b_j^l represent the convolution kernel and offset terms in the lth layer .

The pooling layer controls overfitting through the second extraction of features, so that the model has higher

fault tolerance, but does not change the number of feature maps. The pooling layer in CNN is generally implemented in the same convolution manner, that is, the filter slides on the input feature map, and each element of the output image is calculated by Equation 2. There are also different types of pooling operations, such as Mean-pooling and Max-pooling.

$$x_j^l = f(\beta_j^l down(x_j^{l-1}) + b_j^l) \quad (2)$$

Where f(x) is the activation function, *down* means pooling operation, x_j^{l-1} is the *j*th feature map of layer *l-1*, and β_j^l、b_j^l represent the bias item. By calculating the pixels of the $n \times n$ local area, a value of the output map is obtained, which is used to reduce the amount of data that needs to be stored in the memory, and only forward the features related to the classification.

In the fully connected layer, all nodes in the input layer and the output layer are completely connected to each other. The weight matrix reflects the strength of each node. Mathematically, the output vector in the fully connected layer is shown in Equations 3 and 4.

$$u^l = w^l x^{l-1} + b^l \quad (3)$$

$$x^l = f(u^l) \quad (4)$$

Wherein, w^l is the weight matrix of l th layer, u^l represents the fully connected operation of l th layer, $f(u^l)$ represents the activation of the fully connected operation of layer l, b^l represents the layer l bias item.

2.2 PARALLELISM OF CONVOLUTION CALCULATION

The basic calculation mode of the convolutional layer is the image-kernel convolution. According to Equation 1, the operation process of the convolution layer is shown in Algorithm 1, which contains four cycles. For each layer of the four-layer loop, it can be expanded in parallel according to its data correlation to improve the parallelism of its execution.

algorithm 1 Convolution algorithm

Input：Initial data：x^{l-1}；Convolution kernel：k_{ij}^l；Weight matrix：w^l；Bias item: b^l；Activation function: $f(u^l)$；Total number of input feature maps：n；Total number of output feature maps：m；counter：i。

Output：After the convolution calculation of network layer l：x^l

1：Perform the convolution calculation according to the procedure in section 2.1

2：**while** *i*<=*n* **do**

2：LOOP1：Traverse m output feature maps；

3： LOOP2：Traverse n input feature maps；

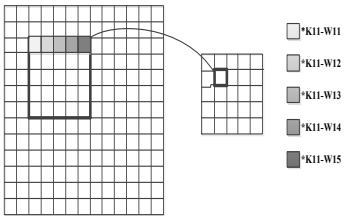
4： LOOP3：The convolution window slides in the input feature map of *XI* * *YI* size；

5： LOOP4：$k \times k$ multiply-accumulate operations in a convolution window；

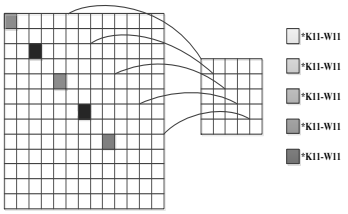
7：**end while**

10: **return** x^l

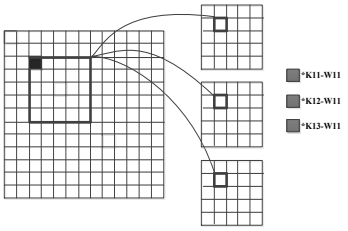
During the convolution operation of the upper layer feature map and the convolution kernel of size k × k connected to the neuron, since multiple convolution kernels without data correlation can perform convolution operations on the same feature map, Multiple convolutional output channels are generated, so the parallel calculation method of the array in the output channels can perform parallel processing on the k × k multiply-accumulate operations. Figure 2 (a) shows the expansion of the innermost loop LOOP4 to achieve parallelism within the convolution window. In the figure, *Knm-Wxy* is expressed as the weight value of the *x*th row and *y* column in the $n \times m$ th convolution kernel.



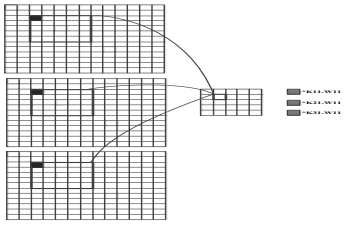
(a) Internal parallelization of the convolution window



(b) Input feature map internal parallel



(c) Parallel input feature map



(d) Parallel output feature map

Fig2 Computational parallel diagram

On the same input feature map, there is no data correlation between different neurons. According to this characteristic, it can be known that the convolution operation of the convolution window at each position can be performed in parallel in the array. Figure 2 (b) shows the expansion of LOOP3 to achieve the internal parallelization of the input feature map convolution window. In the figure, *Knm-Wxy* is expressed as the weight value of the *x*th row and *y*th column in the $n \times m$ th convolution kernel.

For different input feature maps, there are *N* multiply-add convolution windows corresponding to each output neuron (*N* is the number of input feature maps), that is, each output value needs to be $N \times k \times k$ Multiply-add operations. For different

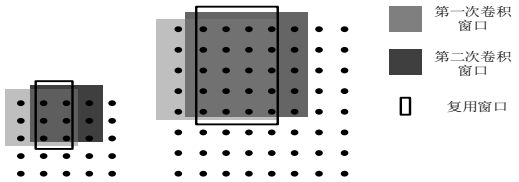
feature maps, the same convolution window can use the distribution mode in the array to perform parallel operations. Figure 2 (c) shows the expansion of LOOP2 to achieve parallel processing between input feature maps.

When processing multiple input feature maps, the same convolution kernel and multiple feature maps are used for convolution calculation, and the convolution kernel is shared on multiple feature maps. Therefore, for different output feature maps, the neurons at the same position connected to the convolution kernel of the previous layer are the same. To calculate the same position on the M output feature maps, $N \times M \times k \times k$ multiplication and addition operating. The same effect can be achieved by using multiple parallel techniques in the array to perform convolution operations on different input feature maps at the same scale. Figure 2 (d) shows the expansion of LOOP1 to achieve parallelism between output feature maps.

These four types of parallel computing can further utilize the reconstruction mechanism to improve resource utilization. According to the characteristics of convolutional parallel computing, because there is no data correlation between each feature map, during the calculation process, the convolution kernels of 3 × 3 and 5 × 5 sizes are initialized on different reconfigurable arrays. In the configuration storage, by calling different configurations in the same processor cluster (Process Element Group, PEG), the algorithm switches for convolution kernels of different sizes.

2.3 Data reuse of convolution kernel

In the calculation process of the convolution kernel, since the convolution kernel moves in the vertical or horizontal order, a region where the height of the convolution kernel and the pixel of the input image coincide will be called a pixel window. Taking the 3 × 3 convolution kernel as an example, during the horizontal movement of the pixel window with a step size of S = 1, the overlapping area marked by the black frame has a total of two columns. There is a 2/3 overlap area in the pixel window corresponding to the sub-convolution operation, which means that the data multiplexing rate in each convolution calculation process exceeds 65%, as shown in Figure 3(a). When the convolution kernel is 5 × 5 operation and the step size is S = 1, there are four columns in the overlap area, each column contains 5 pixels, which will cause 4/5 overlap in the pixel window corresponding to the two convolution operations The area, that is, the data multiplexing rate of two convolution operations is as high as 80%, as shown in Figure 3 (b). Due to the limitation of on-chip storage resources, the data of these overlapping parts needs to be repeatedly loaded from external storage, which will occupy a large amount of storage bandwidth, bring great access delay, and affect the calculation efficiency. Therefore, this paper proposes a convolution algorithm that can fully reuse the data, so that the convolution operation can be efficiently calculated while reducing the number of accesses to external storage. For details, see section 3.2.



（a）3x3 multiplexing window（b）5x5 multiplexing window

Fig3 Schematic diagram of data multiplexing

III Parallel design of Inception Module

3.1 Array processor structure

The video array processor used in this paper is a reconfigurable array processor independently developed by the project team. The processor logically divides the array into processing element groups (Process Elements Group, PEG), and each PEG is composed of 4 × 4 PE arrays. The data interaction between adjacent PEs is accomplished through a shared register using the adjacent interconnection mechanism, and the data interaction between non-adjacent PEs is achieved through access to 4 × 4 distributed random access memories that share a two-level switching structure. Figure 4 shows a schematic diagram of the structure of the PE array. The straight line with arrows indicates the data path for adjacent interconnection communication. Data exchange is performed through the shared registers between PEs. The reconfigurable function of the array and PE is realized by loading fixed instructions stored in the instruction RAM in advance or real-time instructions passed by the global controller and stored in the RAM. At runtime, the corresponding calculation method can be flexibly invoked according to the calculation needs of different layers so that the PE array can run in the data flow mode, so that the circuit has excellent performance such as flexibility, reusable hardware resources, parallel computing, and low power consumption.

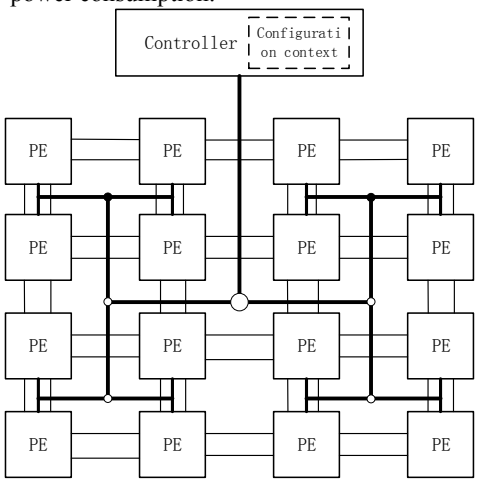


Fig4 Reconfigurable PE array structure diagram

3.2 Parallel reconfiguration design

3.2.1 PARALLEL RECONSTRUCTION DESIGN OF CONVOLUTION OPERATION

During the convolution operation, since the input image is a two-dimensional matrix, it is difficult to directly multiplex the data on the hardware. According to the analysis in Section 2.3, without data preprocessing, each convolution needs to access off-chip storage, which will cause a large access delay and affect the calculation efficiency. Therefore, it is necessary to preprocess the input image and the convolution kernel. First, the original image is divided into blocks according to the size of the convolution kernel and stored in a one-dimensional array by column, and the convolution kernel is also stored in a one-dimensional array by column. Taking a 3 × 3 convolution kernel and an input data size of 4 × 4 as an example, after preprocessing, the process of convolving the convolution kernel on the original input image is simplified as the process of multiplying and adding the corresponding positions of the two columns. At the same time, when the next convolution process only needs to slide the convolution kernel vector down by k elements, it avoids multiple accesses to the same input, thereby reducing the hardware access overhead.

Figure 5 (a) shows a parallelization scheme using a 4x4 PEG convolution operation. The calculation process in the algorithm is complicated, and the process of 3 × 3 convolution is used as an example for description. First, the preprocessed data is stored in DIM. PE00 loads the data from DIM and sends it out. Each computing PE performs parallel convolution calculation after receiving the data, and finally saves the result. The specific algorithm process is as follows.

algorithm 2 Convolutional parallel reconstruction algorithm

Input：Preprocessed data：$conv\_x^{l-1}$；Convolution kernel：k

Output：Convolution calculation result：$conv\_x^{l}$

1：PE00 loads preprocessed data from DIM: Save the original image block.

2：**while** *i*<=*n* **do**

3： **LOOP:**

4： Data distribution：After PE00 loads the data, it sends the data to PE01, PE10, PE11, and PE12 in blocks. After sending the data, PE00 sends handshake signals to each PE.

5： Multiplication / addition calculation process：When PE00 receives the handshake signal, each PE starts calculation at the same time. Multiply the two data through the left shift and add operation to calculate the final result.

6： Results saving process：When all the convolution operations are completed, PE02, PE20, PE21, PE22 transfer the calculation results to PE33, and finally PE33 writes all the results to the DOM.

7：Convolution calculation and reconstruction switching through transfer instructions。

8： **end LOOP**

9：**end while**

10: **return** Store convolution calculation results in DOM

When performing reconstruction calculations, flag bits are set for different convolution kernel sizes before performing convolution calculations. Use the default flag to enable PEG to perform 3 × 3 convolution calculation. When the 3 × 3 convolution calculation is completed, switch to 5 × 5 convolution calculation by calling the CALL command. The parallel calculation process at this time is the same as the 3 × 3 parallel process the same. After all convolution calculations are completed, the results are saved in the DOM, and the pooling operation takes out the intermediate results of the convolution calculations from the DOM through PE33 for calculation.

3.2.2 PARALLEL DESIGN OF POOLING OPERATION

After the convolution operation, the main function of the pooling operation is to perform dimensionality reduction calculation on the feature map to reduce the parameters and calculation amount in the network, while retaining important information in the feature map to prevent information loss in subsequent calculations. In the Inception-V1 module, the size of the pooling operation is 3 × 3, and the stride is 2. In the process of mapping, when the previous layer of convolution operation is completed, the controller controls the computing unit PE33 to load the data in the DOM and sends the pooling operation instruction to each computing unit through the instruction issuing module to complete the pooling calculation , As shown in Figure 5 (b). In order to maintain the accuracy of the feature map as much as possible, we use a 2 × 2 pooling kernel to perform a pooling operation with a step size of 2 on the feature map. This result can not only effectively maintain the accuracy of the feature map, but also reduce the data size, making the structure more conducive to hardware implementation. First, the convolution result will be taken from the DOM and delivered. After the delivery is completed, each PE starts to perform the maximum pooling calculation. The result is saved in the DOM through PE33 so that the next layer of convolution calculation uses the reconstruction scheme. The specific algorithm process is as follows:

algorithm 3 Pooling parallel algorithm

Input：Preprocessed data：$conv\_x^{l}$；pooling kernel：$k = 2 \times 2$；Total number of blocks：n

Output：Pooling calculation results：$pool\_x^{l}$

1：PE33 loads the convolution result data from the DOM: and blocks the result.

2：**while** *i*<=*n* **do**

3： **LOOP:**

4： Data distribution：After PE33 loads the data, it sends the data to PE11, PE12, PE13, and PE31 in blocks. After sending the data, PE33 sends handshake signals to each PE.

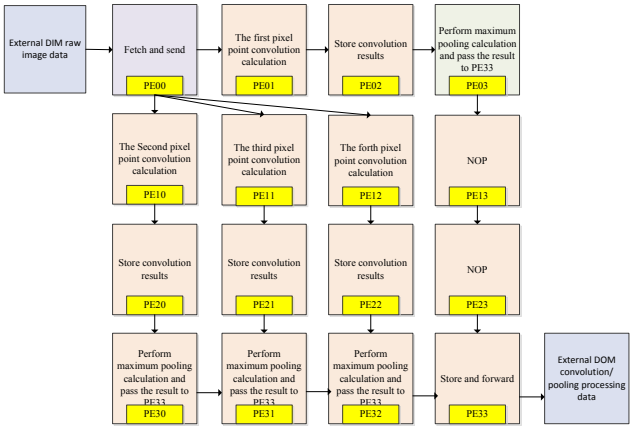
5： Pooling calculation process：PE13 loads addresses 0, 1, 27, and 28 to achieve the maximum pooling calculation of data through a comparison operation, and stores the final result in address 13 of PE13. PE31, PE11, PE12 operate in the same way.

6： Results saving process：When all pooling operations are completed, the calculation results of PE21, PE22, PE23, and PE32 are re-introduced into PE33, and finally PE33 writes all results into the DOM.

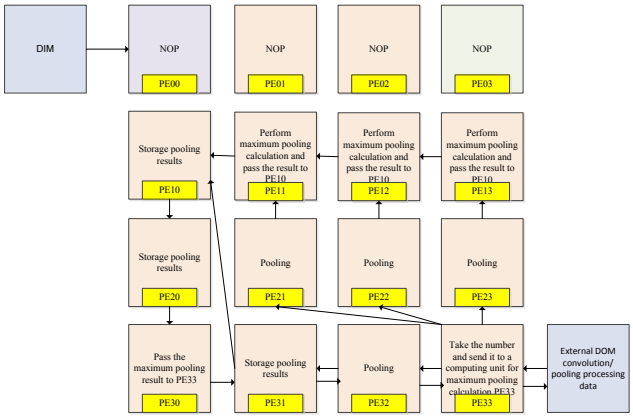
8： **end LOOP**

9：**end while**

10: **return** Store pooled calculation results in DOM



(a) Convolutional parallel operation structure

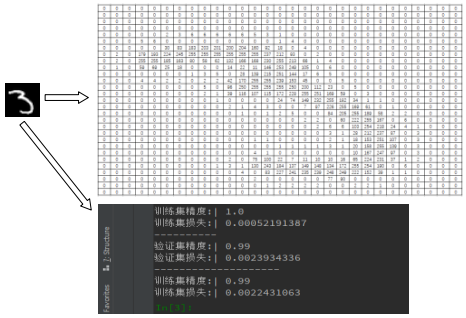


(b) Max Pooling parallel computing structure

Fig5 Parallel operation structure diagram

IV Simulation Experiment and Result Analysis

This paper verifies the feasibility of the parallel scheme on the array processor. First convert the test image into a binary sequence that can be recognized by the array, then store it in the off-chip storage DIM, secondly initialize the instructions of the parallel scheme to the corresponding PE instruction storage, and finally perform simulation verification on QuestaSim, FPGA use Vertex-6 series XC6VLX760 devices, the operating frequency can reach 123MHz. Based on the 4 × 4 PEG, the experiment realized the parallel mapping of 5 × 5 and 3 × 3 convolution operations of the Inception module in the GoogLenet network. The function evaluation data set is selected from Minist (28 × 28 pixel grayscale handwritten digital picture) and Cifar-10 (32 × 32 color image classic data set). The data in the Minist dataset has been pre-processed and turned into a grayscale image with a size of 29 × 29, and Cifar-10 will be processed according to the original data.



（a）PC-side pixel conversion and verification results

（b） FPGA-side verification results

Fig6 System verification results

Figure 6 (a) is the result of pixel conversion of the test image on the PC side using Matlab software. It can be seen that the accuracy of the image recognition of the Minist data set by the network can reach 99%.Figure 6 (b) is the result of FPGA implementation in this method. It can be seen that the classification results are correct.

A 5 × 5 convolution operation takes 425 clock cycles, Inception V1 generates 5 × 5 × 32 convolution results, which requires a total of 340,000 clock cycles; a 3 × 3 convolution is 248 clock cycles, and Inception V1 generates. The 3 × 3 × 128 convolution results require a total of 285696 clock cycles. Table 1 shows the calculation time of this paper and literature [10] and [11]. This article and literature [10, 11] are based on FPGA. Inception V1 of this paper inputs 3 feature maps, 3 × 3 convolution inputs 96 feature maps, and outputs 128 feature maps; 5 × 5 convolution inputs 16 Feature map, output 32 feature maps. In literature [10], 7 feature maps are input on the first layer, and 64 feature maps are output, and 8 feature maps are input on the second layer, and 19 feature maps are output.

Table1 Convolutional layer calculation time

Convolutional layer	[10] (cycles)	[11] (cycles)	This work (cycles)
sum	2006000	62668800	47449088

As shown in Table 1, the processing speed of [10] is much faster than the method in this paper, but the amount of data processed is only 60% of this paper. Reference [11] is the same as the network model in this paper, but the processing speed of this paper is 0.76 times that of it.

Table 2 shows the results of the identification of the two data sets using the design of this paper. It can be seen that the design of this article has reached the functional requirements of the Inception network, and the recognition rate has reached about 99%.

Table 2 Data Set

DataSet	Size	Number of training sets	Number of test sets	Accuracy
Minist	28×28	60000	10000	99.32%
Cifar-10	32×32	50000	10000	98.94%

Table3 FPGA logic resource utilization statistics table

Logic device	Resource utilization		
	[11]	[12]	This work
Slice Registers	144k（16.65%）	-	11.7k（1%）
Slice LUTs	199k（45.98%）	178k（41%）	33.6k（9%）
LUT-FF pairs	-	181k（21%）	106k（30%）

Table 3 shows the FPGA chip resource usage in this article. It can be seen that compared with literature [11, 12], the method proposed in this paper uses fewer resources. The comprehensive results show that the operating power consumption of this method is only 6.395W. Because different methods use different parallel strategies and FPGA platforms, considering the differences in hardware and portability, literature [13, 14] pointed out that the use of energy efficiency ratio (Efficiency) and other indicators can provide an effective comparison, which is derived from Equation 5.

$$\text{Performance per Watt} = \frac{\text{Operations}}{\text{Time}\times\text{Power}} \qquad (5)$$

Wherein, Performance per Watt refers to the power per watt performance, Operations is the number of operations, Time and Power represent time and power consumption, respectively.

Table 4 compares the energy efficiency ratio and the number of operations per second. [1] is the result of optimizing CNN code through CPU. It can be seen that compared with the CPU implementation of [1], the energy efficiency ratio is 4.75 times. [13, 14] are based on FPGA implementation. Compared with the design in this paper, literature [13] has higher operations per second and power consumption, but the energy efficiency ratio is 2.2 times that of this paper. The number of operations per second is 1.5 times that of [14], and the energy efficiency ratio is 1.3 times..

Table 4 Performance comparison table

device	OP/s [GOP/s]	Power [W]	Perf.per Watt [GOP/s/W]
[1]（CPU）	3.54	95	0.037
[13]（FPGA）	14.11	17.67	0.80
[14]（FPGA）	0.75	5.54	0.135
This Work(FPGA)	1.13	6.395	0.176

V SUMMARY

In the process of convolution operation, this paper proposes a reconfigurable array implementation scheme that takes into account the high parallelism of computing and the overhead of on-chip resources for the memory access and power consumption problems faced by convolutional neural networks in hardware acceleration. The calculation of different convolution kernel sizes and pooling operations can be completed on the same array through reconstruction, which improves the utilization of on-chip resources. By converting the input image into a one-dimensional array for storage, the frequency of external storage access in the convolution calculation process is reduced, and a data organization scheme with overlapping windows is proposed, which reduces the number of pixels loaded in external storage by 30%. . The program was tested in the Minist and Cifar-10 test sets, and the results showed that under the operating frequency of 123MHz, the operating power of the FPGA was 6.395W, and the performance was 4.75 times that of the CPU version; compared to other FPGA platform implementations, the proposed Reconfigurable implementation has obvious advantages. This scheme can also be used to implement neural network calculations of similar computing architectures.

ACKNOWLEDGMENT

This paper is supported the National Natural Science Foundation of China under Grant 61834005,61772417,61802304,61602377,61634004, the Key R & D programs in Shaanxi 2017GY-060, and Shaanxi International Science and Technology Cooperation Program No.2018KW-006.

REFERENCES

[1] Chang J, Kang S. Optimizing FPGA-based convolutional neural networks accelerator for image super-resolution[C]. asia and south pacific design automation conference, 2018: 343-348.

[2] Liu Z, Dou Y, Jiang J, et al. Throughput-Optimized FPGA Accelerator for Deep Convolutional Neural Networks[J]. ACM Transactions on Reconfigurable Technology and Systems, 2017, 10(3).

[3] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[C].computer vision and pattern recognition, 2015: 1-9.

[4] Zhang C, Fang Z, Zhou P, et al. Caffeine: towards uniformed representation and acceleration for deep convolutional neural networks[C]. international conference on computer aided design, 2016.

[5] Rahman A, Lee J, Choi K, et al. Efficient FPGA acceleration of Convolutional Neural Networks using logical-3D compute array[C]. design, automation, and test in europe, 2016: 1393-1398.

[6] Shen Y, Ferdman M, Milder P, et al. Maximizing CNN Accelerator Efficiency Through Resource Partitioning[C]. international symposium on computer architecture, 2017, 45(2): 535-547.

[7] CAO Qu-cheng, CHEN Qing-kui. Deep Neural Network Layering Strategy for Embedded Devices [J] Journal of Chinese Computer Systems 2019,40(07):1455-1461.

[8] M. Alwani, H. Chen, M. Ferdman and P. Milder, "Fused-layer CNN accelerators," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, 2016, pp. 1-12.

[9] Lin M, Chen Q, Yan S, et al. Network In Network[C]. international conference on learning representations, 2014.

[10] Shen Y, Ferdman M, Milder P, et al. Overcoming resource underutilization in spatial CNN accelerators[C]. field programmable logic and applications, 2016: 1-4.

[11] Korol G, Moraes F G. A FPGA parameterizable multi-layer architecture for CNNs[C]//2019 32nd Symposium on Integrated Circuits and Systems Design (SBCCI). IEEE, 2019: 1-6.

[12] Shen Y, Ferdman M, Milder P. Escher: A CNN accelerator with flexible buffering to minimize off-chip transfer[C]//2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, 2017: 93-100.

[13] Bettoni M, Urgese G, Kobayashi Y, et al. A Convolutional Neural Network Fully Implemented on FPGA for Embedded Platforms[C]// 2017 New Generation of CAS (NGCAS). IEEE, 2017.