Proceedings of the 42nd Chinese Control Conference
July 24-26, 2023, Tianjin, China

A Scalable ARM+FPGA-Based CNN Accelerator with Limited Hardware Resources

Jinlin Ye[1,2], Wei Zhang[1,2*]

1. Hebei University of Technology, Tianjin 300401, China
2.Control Engineering Technology Innovation Center of Hebei Province, Hebei University of Technology, Tianjin 300401, P. R. China
E-mail: 2020050@hebut.edu.cn

Abstract: General processors cannot meet the requirements of low power consumption and high performance in mobile application scenarios, the hardware research platform of CNNs has begun to turn to high-performance computing platforms such as GPUs and FPGAs. The GPU contains a large number of stream processors to execute operations in parallel, which greatly reduces the operation time of the model. However, because of its large power consumption, mobile applications can hardly afford the GPU's power requirement. Compared with GPUs, FPGAs also have parallel architecture, but the power consumption is much lower, which is suitable for CNN mobile application scenarios. In this work, concerning that FPGAs with large resource can meet the deployment requirements of CNNs but they are too expensive. How to deploy CNNs with little-resource has become an urgent problem to be solved. Based on this, we propose a scalable ARM+FPGA-based CNN accelerator with limited hardware resources, data flow is controlled through ARM, network is computed through FPGA. LeNet is adopted as the acceleration target, experiments were carried out on MNIST datasets and under the little-resource FPGA xc7z020clg400-2. The result show that at the frequency of 100 MHz, the accuracy of the accelerator prediction of 10000 images is 98.57% and the processing time of each image is 16.09ms.

Key Words: Scalable CNN Accelerator; Pipeline Structure; Ping-pong Buffer Mechanism; ARM+FPGA Architecture

1 Introduction

Convolutional neural networks (CNNs) have been widely used in various image-manipulation fields [1]. With the development of 5G, Internet of Things, embedded system and other technologies, the application of CNN class network are growing continually in mobile application scenarios [2]. The number of layers and parameters of deep neural network models continues to increase, which sets higher requirements for hardware computing power, memory bandwidth, and data storage [3]. Traditional computers use serial computing so they cannot adapt to the network structure with complex parallelism [4]. Under the background that general-purpose processors cannot meet the requirements of low power consumption and high performance in mobile scenarios, the hardware research platform of neural networks has begun to turn to high-performance computing platforms such as GPUs and FPGAs [5, 6].

The GPU contains a large number of stream processors, which can execute operations in parallel, which greatly reduces the operation time of the model [7]. However, GPU increase speed by increasing the number of cores and increasing the clock frequency, which also causes the power consumption to be too high [8, 9]. Edge devices have high power consumption requirement due to the inability of industrial environments to provide stable power supply and the need for heat dissipation to prevent them from breaking through their power ceilings. Therefore, GPU is not suitable for mobile application scenarios.

FPGA has high performance, low power consumption, high parallelism, strong flexibility, repeatable configuration and other characteristics, is also suitable for embedded devices, FPGA-based convolutional neural network hardware acceleration is favored by the majority of researchers, more and more used as a deep learning accelerator hardware platform [10].

Artificial intelligence inference algorithms are deployed on FPGAs, data optimization methods are used to compress and optimize neural network data at the software level, and combines with parallel computing architecture, pipeline design, data storage mode and other architectures at the FPGA hardware level to further improve memory access efficiency and throughput, reduce computing and memory access energy consumption, and realize efficient accelerated computing of CNN models. Zhang at al. [11] designed a reconfigurable CNN accelerator with AXI bus in ARM+FPGA architecture, combining convolutional layer and pooling layer to reduce the number of data movements and off-chip memory accesses of the convolutional layer and pooling layer, converting floating-point numbers to 16 into dynamic fixed-point format to improve computing performance, and the peak performance at 300MHz clock frequency is 189GOP/s, the power is 11.8w; Alhussain at al. [12] proposed a CNN-based Soc-FPGA accelerator, which uses cyclic tiling transformation data flow modeling to convert the convolutional layer and the fully connected layer into a vector multiplication between the input and output feature maps to form an on-chip computing unit, and uses cyclic block transformation to construct the IP core of accelerator, the peak performance at 200MHz clock frequency is 230GOP/s, the power is 4.7w; Thang [13] designed a 2D-convolutional hardware accelerator for FPGA with VHDL, which improved the execution performance of CNN by 6.2 times compared to the software implementation; Shi at al. [14] proposed a general parallel convolution acceleration unit, the results of weights and

*This work was supported in part by the Natural Science Foundation of Hebei (No.F2021202043), Hebei University of technology Basic Scientific Research Business Expenses Special Funds(No.24/424132022) and S&T Program of Hebei (21567698H) (Corresponding author: Wei Zhang)

Wei Zhang is now working in school of Artificial Intelligent and Data Science, Hebei University of Technology, Tianjin 300401, China (email: 2020050@hebut.edu.cn)

feature maps are stored using semi-floating points, and the convolution process is performed using fixed-point calculations, a Lenet-5 convolutional neural network was deployed on the SOC development platform of ZCU102, at the frequency of 150MHz, the computing performance of FPGA reached 28.8 GOPS, and the recognition rate reached 99.11% on the MNIST data level. So far, the traditional methods mostly focus on the issue that FPGAs can successfully deploy CNNs with large computation resources, but neglect that with little resources consumption traditional methods can hardly implement. So, to solve the problem that little-resources FPGAs cannot deploy CNNs, we focus on: (a) How to implement convolution operations through hardware. (b) How to make efficient use of FPGA hardware resources.

In this paper, we propose a scalable ARM+FPGA-based CNN accelerator with limited resources. The major innovation of this work lies on these points:

(1) We explore the hardware implementation of convolution: Multiple Processing Elements (PEs) are constructed to perform convolution operations of each channel in parallel.

(2) We propose a method to realize CNN under the condition of limited hardware resources: The convolution operation data is stored in off-chip DRAM, the FPGA's on-chip resources are only performed for convolution operations. Data flow is controlled through ARM.

2 Methodology

2.1 CNN Algorithm

Convolutional neural network (CNN) is one of the artificial intelligence algorithms that are mainly used for various task in image and video analysis. CNN consists of convolutional layer, pooling layer and fully connected layer.

(1) Convolutional Layer

The function of the convolutional layer is to extract the features of a local region. Convolution operation are actually a cross-correlation operation. In the process of convolution, it is necessary to flip the convolution kernel, by the flipping of the convolution kernel has nothing to do with its feature extraction ability. Therefore, in the specific implementation, in order to reduce some unnecessary operations, cross-correlation is usually used instead of convolution. The convolution operations in this paper are replaced by cross-correlation.

The convolution calculation process is shown in Fig. 1. The number of channels of the input feature map is "CH_{in}", the size of the input feature map is "$C_{in} \times R_{in}$", the number of channels of the output feature map is "CH_{out}", the size of the output feature map is "$C_{out} \times R_{out}$", the size of the convolution kernel is "$K_c \times K_r$", the number of convolution kernel is "$CH_{in} \times CH_{out}$", each output channel corresponds to a set of convolution kernels, each convolution kernel in each group of convolution kernels corresponds to an input channel, and the movement step is "$stride$". The input feature data is stored on the input feature map, the weight data is stored on the convolution kernel, and the output feature data is stored on the output feature map. Convolution calculation process: the data of the corresponding position of the input feature map is multiplied by the weight parameter of the corresponding position on the convolution kernel, and then the multiplication result on all convolution kernels in each group is accumulated, and the accumulation result is mapped to the corresponding position of the output feature map, and then the slide is carried out with stride as the step to extract the features of the next position.

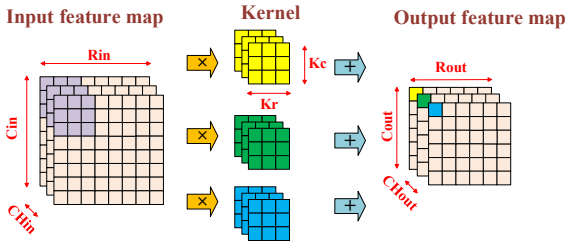

Fig. 1 Convolution calculation process

The process implemented by the convolution is equivalent to a six-level nested loop. The algorithm is:

$$Out_{cho,ro,co} = \sum_{chi=0}^{CH_{in}-1} \sum_{kr=0}^{K_r-1} \sum_{kc=0}^{K_c-1} \left(W_{cho,chi,kr,kc} \times In_{chi,s\times ro+kr,s\times co+kc} \right)$$
$$cho \in [0, CH_{out}-1]$$
$$chi \in [0, CH_{in}-1]$$
$$kr \in [0, K_r-1]$$
$$kc \in [0, K_c-1] \quad (1)$$
$$co \in [0, C_{out}-1]$$
$$ro \in [0, R_{out}-1]$$

where Out is the output feature parameter, In is the input feature parameter, W is the weight, CH_{in} is the number of input channels, CH_{out} is the number of output channels, K_c is the convolution kernel length, K_r is the convolution kernel width, C_{out} is the output feature length, R_{out} is the output feature width, s is the movement step.

(2) Pooling Layer

The function of the pooling layer is to reduce the number of features. The pooling calculation process is shown in Fig. 2. Average or maximize the pixels in the specified area of the input feature map, and then map to the output feature map.

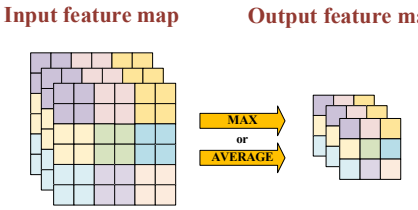

Fig. 2 Pooling calculation process

(3) Fully Connected Layer

The fully connected layer is mainly used for classification. The feature space mapping sample label space calculated by the previous layer (convolutional layer, pooling layer, etc.) is also used to transform the two-dimensional feature map of the convolutional output into a one-dimensional vector, which functions as a combination feature and classifier in the whole convolutional neural network.

LeNet [15] is designed by Yann LeCun for handwritten digit recognition. As shown in Fig. 3, the model has two convolutional layers, two max-pooling layers and three fully connected layers.

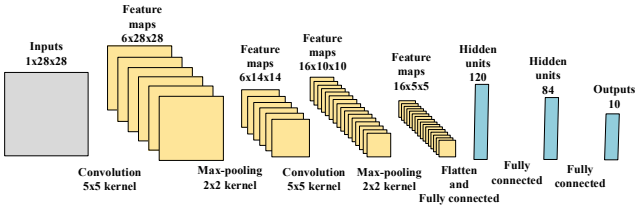

Fig. 3 LeNet structure

2.2 Optimization Algorithm

The following is the optimization algorithm used in this article:

(1) Pipeline Structure

The pipeline structure of hardware is a method of high-speed processing through a large number of calculations [16]. The core idea of pipeline architecture is to use multiple hardware circuits to implement computing tasks in parallel.

Fig.4 is high-speed processing principle of pipeline structure. Fig.4(a) is non-pipeline structure. The unit operation time is L, the total operation amount is N, the time to complete N operations is T. Fig.4(b) is pipeline structure, each operation corresponds to a piece of hardware circuit, the hardware circuit is divided into n equal stages, the unit stage time is $\frac{L}{n}$, these stages are parallel module, the start of the nest operation does not rely on the finish of the previous one.

Fig.4(a) is non-pipeline structure, all operations are executed sequentially on the same hardware circuit. The unit operation time is L, the total operation amount is N. The time required for the non-pipeline structure to complete N operations:

$$T(N) = L \times N \tag{2}$$

Fig.4(b) is pipeline structure, each operation corresponds to a piece of hardware circuit, the hardware circuit is divided into n equal stages, these stages are parallel module, the start of the nest operation does not rely on the finish of the previous one. The unit operation time is L, the total operation amount is N, and the pipeline is divided into n stages. For n-level pipeline structures, the time required to complete operation 1 is L, and the next operation is completed after $\frac{L}{n}$ time after operation 1 is completed. And so on, the subsequent $N-1$ operations are completed every $\frac{L}{n}$ unit time. Therefore, the total operation time of the pipeline structure:

$$T_{pipe}(N) = L + (N-1) \times \frac{L}{n} = \frac{(n+N-1)L}{n} \tag{3}$$

Formula (2) divided by formula (3) gives the lift rate of the speed at which the pipeline structure is used:

$$S_{pipe}(N) = \frac{T(N)}{T_{pipe}(N)} = \frac{nN}{n+N-1} = \frac{n}{1+\frac{n-1}{N}} \tag{4}$$

When $n \ll N, S_{pipe}(N) \cong n$. Therefore, the speed increase obtained by the pipeline structure and the non-pipeline structure is proportional to the number of stages, which is about n times, that is, the throughput index can be increased by up to n times.

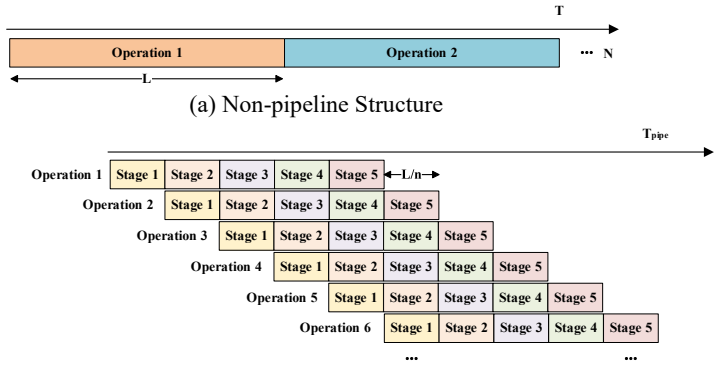

Fig.4 High-speed processing principle of Pipeline Structure

(2) Ping-pong Buffer Mechanism

The Ping-pong Buffer is used to speed up a device that can overlap the I/O operation with the data processing operation [17]. In Fig. 5, one buffer is used to hold a block of data so that a reader device will see a complete version of the data, while in the other buffer a writer device is creating a new version of data. When the new block of data is complete, the reader and the writer device will alternate the two buffers. As a result, the usage of double buffer increases the overall throughput of a device and helps to prevent eventual bottlenecks.

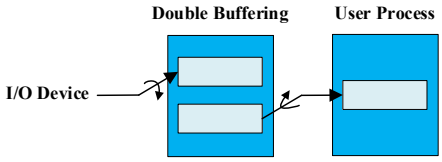

Fig. 5 Ping-pong Buffer Mechanism

3 Accelerator Design

In the traditional structure of accelerators, the resources on the FPGA are not efficiently utilized, and the scalability of CNN is not fully exerted, which become the reason that the little-resource FPGA proposed in the introduction cannot deploy CNN. In order to fully utilize the FPGA resources, the mathematical model of FPGA's on chip resources utilization are written as:

$$\begin{cases} X_{BRAM} \le BRAM_{max} \\ X_{DSP} \le DSP_{max} \\ X_{FF} \le FF_{max} \\ X_{LUT} \le LUT_{max} \end{cases} \quad (5)$$

$$\min Latency$$

where X is the number of various resources used, Latency is the time required for network prediction, BRAM is Block RAM, DSP is Digital Signal Processing Element, FF is Flip Flop, and LUT is Look Up Table.

To solve (5), we propose a scalable ARM+FPGA CNN accelerator (AFCA). AFCA design method is shown in Fig. 6. In the first step, we make LeNet lightweight. We modify the LeNet model and we call the modified LeNet i-LeNet, and then we train i-LeNet. In the second step, we design the convolution and pooling operations as custom hardware IP core. we build multiple PEs, parallelize the operation process of convolution, and pipeline the PE at the same time. And then we build a data access and storage mode and pipeline data access and storage and convolution operations through Ping-pong Buffer. In the third step, we design

ARM+FPGA system. Data flow is controlled through ARM, network is computed through FPGA. In the fourth step, we test accelerator on the MNIST datasets.

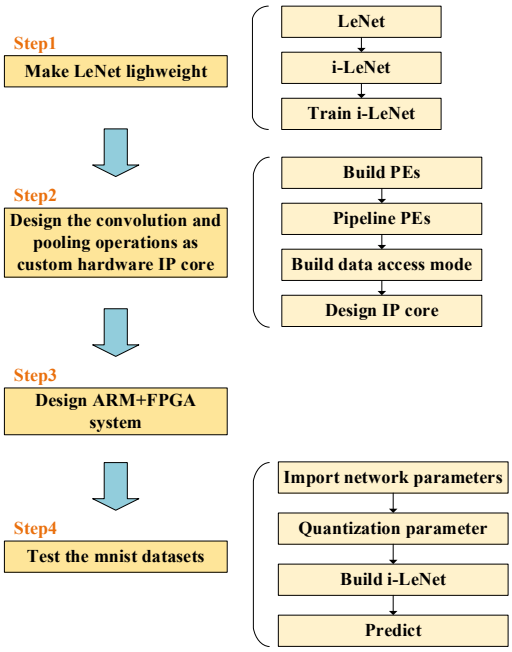

Fig. 6 AFCA design method

3.1 I-LeNet

In order to solve the problem that the number of LeNet channels at each layer is too small to give full play to the parallel advantage of FPGA, a novel i-LeNet model is proposed where three major modifications are made on traditional LeNet algorithm. I-LeNet's network structure is shown in Table 1. We remove two fully connected layers to reduce network size. We add two convolutional layers to improve feature extraction capability. To accommodate the parallelism of the CNN accelerator, we set the number of channels to 32, set the convolution kernel size to 5×5, set the max-pooling kernel size to 2×2, set the convolution stride to 1, set the max-pooling stride to 2 and set the padding to 2. We use relu as the activation function of the convolutional layer.

Table 1: I-LeNet's Network Structure

Layer	Input channels	Output channels	Kernel size	stride	padding
Conv1+relu	1	32	(5,5)	(1,1)	(2,2)
Conv2+relu	32	32	(5,5)	(1,1)	(2,2)
Max-pool1	32	32	(2,2)	(2,2)	/
Conv3+relu	32	32	(5,5)	(1,1)	(2,2)
Conv4+relu	32	32	(5,5)	(1,1)	(2,2)
Max-pool2	32	32	(2,2)	(2,2)	/
Fc1	1568	10	/	/	/

When training and testing i-LeNet, in order to ensure the accuracy, the network parameters use floating-point number. However, floating-point arithmetic consumes large memory resources in FPGA, in order to improve the efficiency of convolution operation, we use fixed-point strategy. We transform the weight, bias and feature value from floating-point number to 16-bit fixed-point.

3.2 AFCA

Due to the operation process of convolution and pooling, there is no data dependency between the input feature map and the output feature map of different layers, the two dimensions of input layer and output layer are processed in parallel. The hardware implementation of convolution is shown in Fig. 7. We decompose the matrix of the input feature map, output feature map and weight by degree of parallelism. The operation of processing one layer of convolution is regarded as a Processing Element (PE), and multiple PEs are used to process the convolution operation of different layers respectively, and the PEs are pipelined to improve hardware resource utilization and reduce latency.

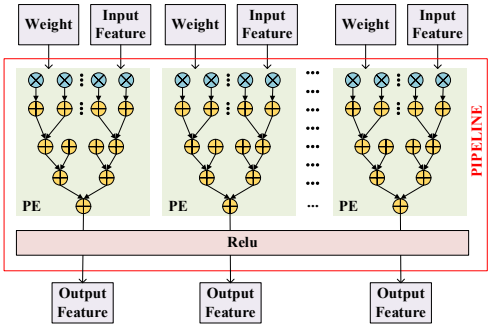

Fig. 7 The hardware implementation of convolution

The input characteristics and output characteristics of each layer of the network in the neural network are not fixed, if the number of input or output layers increases, and the convolution calculation is still carried out according to the parallel pipeline design method, it is bound to consume more computing and storage resources, and when the upper limit of the FPGA on-chip resources is reached, the layer of convolution cannot be calculated.

The large-scale convolution is divided into blocks according to the two dimensions of the number of input feature layers and the number of output feature layers. Each piece of convolution is required to have a consistent amount of computation and require less hardware resources than the on-chip resources of the FPGA. The entire convolution is done by calling this block accelerator in a loop. Compared with the literature [18] method of storing weights in off-chip DRAM and input feature data and output feature data in on-chip BRAM, we store all data of convolution operations in off-chip DRAM, which are input feature data, weight data and output feature data, while on-chip resources are all used for convolution operations. Considering that the convolutional layer is adjacent to the pooling layer, the method described in [11] is adopted to carry out the pooling operation after completing the convolutional calculation, and then the output features are stored in the off-chip DRAM to reduce the delay caused by data flow. The accelerator and the off-chip DRAM are transferred via the AXI bus. The architecture of the convolutional accelerator is shown in Fig. 8. The operating mode of the accelerator is shown in Fig. 9.

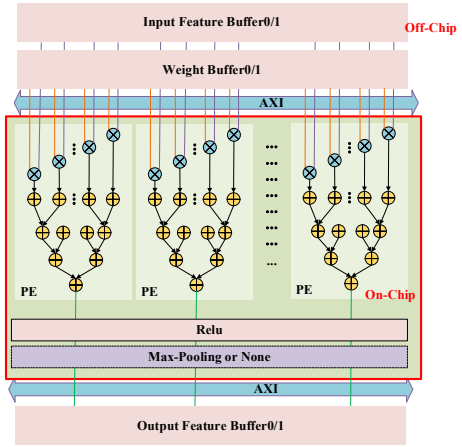

Fig. 8 The architecture of the convolutional accelerator

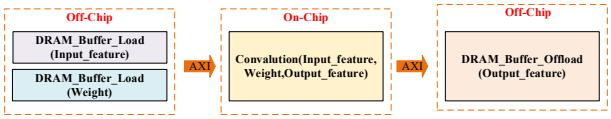

Fig. 9 The operating mode of the accelerator

Data transmission and convolution operations are implemented in different hardware modules, and it is necessary to access the input data in the off-chip DRAM, perform convolution operations, and then store the results of the operations in off-chip DRAM. To further improve the efficiency of the accelerator, both processes are optimized by Ping-pong Buffer.

We split the on-chip Buffer into two groups of the same size (Buffer0 and Buffer1). When the accelerator performs a convolution operation on the data in Buffer0, Buffer1 loads the data required for the next convolution operation in off-chip DRAM. When the accelerator finishes processing the data in Buffer0, Buffer1 stores the calculation results. When the next clock cycle comes, the accelerator directly processes the data in Buffer1, and Buffer0 loads the input data of the next convolution operation in off-chip DRAM, and so on, streamlining the transmission of data and the operation of convolution. The sequence diagram of Ping-pong Buffer is shown in Fig. 10.

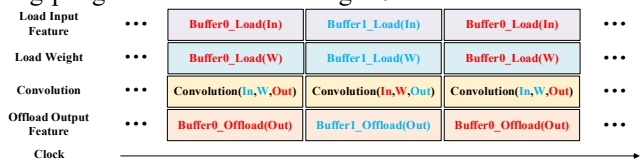

Fig. 10 The sequence diagram of Ping-pong Buffer

The scalability of AFCA is that it can achieve convolution of any size with limited resources. We can explore the upper limit of the scale of convolution that can be achieved with a fixed resource. And the accelerator of this scale of convolution is the unit block, and the method of time-sharing multiplexing is used to achieve large-scale convolutional deployment.

3.3 ARM+FPGA System

We encapsulate AFCA into IP core of AXI interface and store network parameters and input data on an SD card. We use DDR as off-chip DRAM. The system we built is shown in Fig. 11. ARM reads data from SD card and stores it in the DDR, and then calls AFCA-IP to build i-LeNet. ARM controls the data flow, and FPGA accelerates the network.

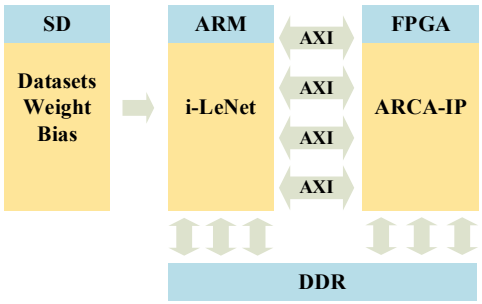

Fig. 11 ARM+FPGA system

4 Evaluation

We conducted experiments on xc7z020clg400-2 to verify the accelerator proposed in this paper. We use pytorch to build and train i-LeNet, use Vivado HLS to design AFCA IP core, use Vivado to build ARM+FPGA system. We use SDK to test accelerator. We use MNIST to test the performance of the accelerator. The result show that the accuracy of the accelerator prediction of 10000 images is 98.57% and the processing time of each image is 16.09ms. The comparison between the processed design and previous works is presented in Table 2. Zynq-7000 has far less on-chip resources than ZCU102, the performance of the acceleration is inferior to reference [14]. However, compared with the method proposed in reference [14], the resource utilization rate is higher. Therefore, we solve the problem that little-resource FPGAs cannot deploy CNNs.

Table 2: Compare with previous work

		[14]	proposed
Platform		ZCU102	Zynq-7000
Frequency (MHz)		150	100
Accuracy (%)		99.11	98.57
BRAM	Using resource	54.4	33
	Logical resource	912	280
	Utilization (%)	5.98	11
DSP	Using resource	204	38
	Logical resource	2520	220
	Utilization (%)	8.1	17
FF	Using resource	66569	11345
	Logical resource	584160	106400
	Utilization (%)	11.4	10
LUT	Using resource	25276	30201
	Logical resource	274080	53200
	Utilization (%)	9.22	56
Latency (ms)		16.2	16.09

5 Conclusion

In this work, we design a scalable ARM+FPGA-based CNN accelerator with limited hardware resources, which solves the problem that little-resource FPGAs cannot deploy large-scale convolution. We lighten the target network and transform the weight, bias and feature value from floating-point number to 16-bit fixed-point. We store the network parameters to the off-chip DDR, and on-chip resources of FPGA are used for calculation. We accelerate the convolutional layer through FPGA, build the network and control the data through ARM. LeNet is adopted as the acceleration target, experiments were carried out on MNIST datasets and under the little-resource FPGA xc7z020clg400-2. The result show that at the frequency of 100 MHz, the accuracy of the accelerator prediction of 10000 images is 98.57% and the processing time of each image is 16.09ms. In future research, we will further improve the performance of the accelerator, build an image acquisition and display system, and realize edge AI computing.

References

[1] R. Chauhan, K. K. Ghanshala, and R. Joshi, Convolutional neural network (CNN) for image detection and recognition, *in 2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, 2018: 278-282.

[2] H. A. Imran, U. Mujahid, S. Wazir, U. Latif, and K. Mehmood, Embedded development boards for edge-AI: A Comprehensive Report, *arXiv preprint*, 2020.

[3] Z. Li, Y. Zhang, J. Wang, and J. J. J. o. S. Lai, A survey of FPGA design for AI era, *Journal of Semiconductors*, 41(2), 2020.

[4] B. Li, J. Gu, and W. Jiang, Artificial intelligence (AI) chip technology review, *in 2019 International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI)*, 2019: 114-117.

[5] A. Reuther, J. Kepner, M. Jones, P. Michaleas, S. Samsi, and V. Gadepally, Survey of machine learning accelerators, *in 2020 IEEE high performance extreme computing conference (HPEC)*, 2020: 1-12.

[6] Y. Deng, Deep learning on mobile devices: a review, *in Mobile Multimedia/Image Processing, Security, and Applications 2019*, 2019: 52-66.

[7] T. Baji, GPU: the biggest key processor for AI and parallel processing, *in Photomask Japan 2017: XXIV Symposium on Photomask and Next-Generation Lithography Mask Technology*, 2017: 24-29.

[8] S. Hong and H. Kim, An integrated GPU power and performance model, *in Proceedings of the 37th annual international symposium on Computer architecture*, 2010: 280-289.

[9] T. Baji, Evolution of the GPU device widely used in AI and massive parallel processing, *in 2018 IEEE 2nd Electron Devices Technology and Manufacturing Conference (EDTM)*, 2018: 7-9.

[10] X. Peng, J. Yu, B. Yao, L. Liu., and Y. Peng. A review of FPGA - based custom computing architecture for convolutional neural network inference, *Chinese Journal of Electronics*, 30(1), 1-17, 2021.

[11] S. Zhang, J. Cao, Q. Zhang, Q. Zhang, and Y. Wang, An FPGA-based reconfigurable CNN accelerator for YOLO, *in 2020 IEEE 3rd International Conference on Electronics Technology (ICET)*, 2020: 74-78.

[12] A. Alhussain and M. Lin, Hardware-efficient template-based deep CNNs accelerator design, *in 2022 IEEE International Conference on Networking, Architecture and Storage (NAS)*, 2022: 1-4.

[13] T. V. Huynh, FPGA-based Acceleration for Convolutional Neural Networks on PYNQ-Z2, *International Journal of Computing and Digital Systems*, 11(1), 441-450, 2022.

[14] Y. Shi, T. Gan, S. Jiang. Design of parallel acceleration method of convolutional neural network based on FPGA, *in 2020 IEEE 5th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*, 2020: 133-137.

[15] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, Gradient-based learning applied to document recognition, *Proceedings of the IEEE*, 86(11): 2278-2324: 1998.

[16] Y. Li, L. Ling, and J. Wu, The application of pipeline technology: An overview, *in 2011 6th International Conference on Computer Science & Education (ICCSE)*, 2011: 47-51.

[17] Y. M. Joo and N. McKeown, Doubling memory bandwidth for network buffers, *In Proceedings. IEEE INFOCOM'98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century*, 1998: 808-815.

[18] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, A high performance FPGA-based accelerator for large-scale convolutional neural networks, *in 2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016: 1-9.