Design of Convolutional Neural Network Based on FPGA

Sheping Zhai¹,², Cheng Qiu*, Yuanyuan Yang¹, Jing Li¹ and Yiming Cui¹

¹ School of Computer Science & Technology, Xi’an University of Posts & Telecommunications, Xi’an, Shaanxi, 710121, China
² Shaanxi Key Laboratory of Network Data Analysis and Intelligent Processing, Xi’an, Shaanxi, 710121, China

*Corresponding author’s e-mail: qc_0305@126.com

Abstract. Recently, FPGAs have been widely used in the implementation of hardware accelerators for CNN, especially on mobile and embedded devices. This paper proposes an FPGA-based CNN accelerator. The highly reusable accelerator function is designed to construct the optimized convolutional neural network and memory optimization with a lower hardware resource consumption. The results show the advantage of performance and power when compared with Core i5 CPU and GTX 960 GPU.

1. Introduction
Convolutional neural networks (CNNs) are a typical multi-layer neural network. With the continuous development of deep learning technology, they are widely used in the fields of machine vision and speech analysis[1-2]. The traditional convolutional neural networks execute calculations based on CPUs. Such calculations are slow and inefficient, and also difficult to meet real-time calculation requirements. Therefore, CPU-based convolutional neural networks are widely used. The literature [3] compared the CNN open source projects based on GPU and found that there are some problems of GPUs such as high power consumption and high cost.

Field-programmable gate array (FPGA) is a customizable programming hardware circuit structure, which is a mainstream digital circuit design mode. The computational mode of parallel computing provided by FPGA fits the computational characteristics of convolutional neural networks. At the same time, the reprogrammable features of FPGA are also suitable for the variable network structure of neural networks. Therefore, the design of CNN based on FPGA has received extensive attention. In [4-5], a deep convolution neural network accelerator based on FPGA is proposed. In [6], a deep pipeline FPGA cluster is designed to implement high efficiency CNN.

This paper first introduces the convolutional neural network, and according to the characteristics of CNN, a CNN system based on FPGA through appropriate parallel processing and pipeline structure is designed, which effectively improving its performance. Finally, compare it with the implementation of CNN on CPU and GPU is made.

2. Convolutional Neural Network

2.1 Model Structure
Convolutional neural network is one of the classic networks of deep learning. It consists of multiple convolutional layers, pooled layers and fully connected layers. Figure 1 shows the classical structure of convolutional neural networks. The convolutional neural network take the image which is going to be identified as the input directly, and obtain the recognition result through multiple convolution layers, pooling layers and fully connected layers.

2.1.1 Convolutional layer model. The convolution layer is convoluted by the input $f_j^{in}$ and the convolution kernel composed of the weights $w_{ij}$. Through offsetting the result to get the output, and the output $f_i^{out}$ is the convolved local sampled feature set[7]. The model of the convolutional layer is described as:

$$f_i^{out} = \sum_{j=1}^{n} f_j^{in} \ast w_{ij} + b, 1 \leq i \leq n_{out}$$

(1)

2.1.2 Pooling Layer Model. The pooling layer usually uses the maximum sampling or the mean sampling to perform the pooling operation to reduce the size of the input matrix. The operation is as shown in equation (2). The pooling operation can effectively reduce the data processing capacity of the next layer while avoiding the loss of characteristic information.

$$f_{i,j}^{out} = \max( f_{m,n}^{in}, f_{m+1,n}^{in}, \cdots, f_{m+s,p-1,n}^{in}, f_{m+1,s,n}^{in}, \cdots, f_{m+1,s,p-1,n}^{in}, \cdots, f_{w-1,n}^{in}, f_{w,s-1,n}^{in}, f_{w,s}^{in}, f_{w+1,s,n}^{in}, \cdots, f_{w+1,s,p-1,n}^{in} )$$

(2)

2.1.3 Full Connection Layer. The full connection layer converts the input into a linear space, thus the output is obtained.

$$f^{out} = \sum_{j=1}^{n} f_j^{in} w + b$$

(3)

2.1.4 Activation Function. The activation function implements a nonlinear transformation of the input excitation, and the output is usually processed after each layer. Common activation functions include shock response (Sigmod), nonlinear (Relu), trigonometric function (Tanh), and so on.

2.2 CNN Characteristics
Convolutional neural networks have the properties of sparse connections and parameter sharing. Sparse connections are achieved by reducing the size of the convolution kernel such that the convolution kernel size is smaller than the image size. Figure 2 shows the sparse connection mode. Each output neuron is only connected to its neighboring neurons, $x_j$ is the input neuron of this layer, and $s_i$ is the output neuron. The convolution kernel has a width of 3, this mean only three outputs are simultaneously affected by the same input $x_j$. By constructing sparse connections, one can
effectively describe complex connections or complex interactions in the network.

Parameter sharing means that when calculating the same output feature map, each output pixel is calculated using the same parameter value, that is, the same convolution kernel is used to traverse the entire input image. As shown in Figure 3, the dark connecting lines represent the shared parameters. Parameter sharing means that you do not have to learn a specific convolution kernel at each individual location, but you can use the same convolution kernel at all locations. The convolutional layer which complete the parameter sharing greatly reduces the number of parameters in the convolutional layer, and can effectively reduce the storage space of the convolutional neural network system and improve the computational efficiency.

3. CNN Accelerator Design

3.1 System Framework
The scale of convolutional neural networks is relatively large. This paper divides the functional modules of convolutional neural networks and designs a convolutional neural network system architecture based on FPGA, as shown in Figure 5. The system can be divided into a PS part and a PL part, and the two parts are connected through the AXI bus. The PS part includes ARM and DDR memory, ARM is responsible for the overall flow control of the convolutional neural network algorithm and sends commands to the PL part. DDR stores the characteristic data of the input and output and the parameters of the kernel. The PL part is a convolution acceleration module that includes a DataMover for carrying data from the DDR to the peripheral, a control module, an on-chip input, an output buffer, and a convolution calculation unit. The PL part sets different clocks of the DataMover and the Controller separately, to realize the parallel of computing and data acquisition. The crossing of the control signals between the two clock domains is achieved by the synchronization module, and the crossing of the data signals is achieved by the asynchronous FIFO.
3.2 Convolution Unit

In CNN, the convolution operations performed the most frequently, accounting for more than 90% of the entire network. After each convolution kernel slide by one step, a multiply-and-accumulate operation is performed. In the hardware design, the convolution operation between different convolution kernels or different convolutional layers can be realized. In the operation, the data will be highly reusable and irrelevant: The input feature graph data correspond to multiple convolution cores, and the operation is independent of each other. The convolution sub-region in the input feature graph data shares the same convolution kernel. The different convolution sub-regions in the input feature image data have a lot of overlaps in space. Therefore, in order to improve computing performance, it is necessary to increase data utilization and the parallelism of convolution operation. This paper designs a pipeline parallel multiply-accumulate structure to complete all multiply-accumulate operations in a convolution kernel of one clock cycle. Figure 5 is a convolution operation module with a convolution kernel size of $3 \times 3$, which can perform 9 multiplication operations per clock cycle, and then output the final output feature pixels through the adder. For a convolutional neural network that requires a filled operation, the edge position of the input feature maps needs to be padded with zeros. Therefore, a blocker is added to the convolver to set the edge position of the output to zero. This design makes the pipeline not be interrupted by data padding or feature map switching, and guarantees the parallelism between convolution calculations from the circuit level.
3.3 Data Cache
The data cache in the FPGA is implemented using on-chip BRAM memory. Therefore, different data inputs and outputs are instantiated as on-chip BRAM blocks, and the same block shares the same input and output interface. Due to limitations on the number of I/O interfaces, the accelerator's data throughput rate is affected by the cached data throughput rate. Therefore, dividing the cached array of different blocks of BRAM blocks and increasing the number of interfaces will help to increase the speed between data throughput and the parallelism of accelerator data.

In a convolutional neural network, when calculating a convolution operation on a point, it is necessary to obtain $K^2$ data of the input pixel $X_{in}$ and the $K^2 - 1$ neighborhood around it, and then multiply the corresponding $K^2$ weights in the convolution kernel and accumulate. Since the data is fed in the form of a data stream, this paper designs a cache structure to enable simultaneous acquisition of $K^2$ data in a convolution window, as shown in Figure 6. In registers reg1 to reg9, one can get the image data of a 3*3 convolution window. The convolution window can be moved to the right or down each clock, enabling each clock to get all the data for a convolution window.

![Shift Register](image)

Figure 6. Data cache structure.

3.4 Calculation Accuracy
Convolutional neural network algorithms usually use floating-point arithmetic, but implementing floating-point operations on an FPGA consumes a lot of resources, so the choices of the appropriate fixed-point number is needed. Through multiple experiments and comparisons, it is found that the input parameters adopt 16-bit fixed-point numbers and the weight data are 18-bit fixed points, which can obtain results closer to floating-point numbers.

4. Experimental

4.1 Program
The experiment uses Xilinx Zedboard embedded FPGA experimental platform, which is equipped with an xc7z020 chip, which is composed of ARM A9 processor and reconfigurable logic. The number of resources such as BRAM, DSP, FF and LUT can be reconstructed by the logic part. They are 560, 220, 106400 and 53200 respectively. In the comparison experiment, the CPU uses Intel Core i5 2500K processor, the clock speed is 3.3GHz, and the GPU uses NVIDIA GeForce GTX 960. The hardware implementation environment is Xilinx Vivado 2017.

The experiment used a Lenet-5 network consisting of two convolution layers, two pool layers and one full connection layer, as shown in Table 1. The data set uses the MNIST handwritten digital data set. All pictures are 28×28 size grayscale images, and the convolutional neural network is trained and executed using CAFFE version 1.0.

| Layer | Input | Kernel | Output |
|-------|-------|--------|--------|
| Conv-1 | 28×28×3 | 5×5 | 28×28×6 |
| Pool-1 | 28×28×6 | 2×2 | 14×14×6 |
| Conv-2 | 14×14×6 | 5×5 | 10×10×16 |
| Pool-2 | 10×10×16 | 2×2 | 5×5×16 |
| Conv-3 | 5×5×16 | 5×5 | 10 |

Table 1. Lenet-5 parameters.
4.2 Result
The FPGA resource consumption is shown in Table 2. Because the network is mostly convolutional operation, the parallel operation of the convolution operation consume a large amount of DSP resources, and the DSP consumption occupancy rate is 85.5%, and other resources are relatively less used.

Table 2. Resource consumption.

| Module | Number | Occupancy rate |
|--------|--------|----------------|
| BRAM   | 257    | 45.8%          |
| LUT    | 25436  | 47.8%          |
| DSP    | 188    | 85.5%          |

Compare the performance of CNN in FPGA with Intel Core i5 CPU and NVIDIA GeForce GTX 960 GPU, resulting in Table 3. In terms of processing images, the convolutional neural network based on FPGA optimization design processes a single picture much less time than the CPU, which is equivalent to the GPU speed. In terms of power consumption, the FPGA has the lowest power consumption due to its efficient computational efficiency. The power consumption is about one-third of the CPU power consumption, which has a very high energy consumption advantage.

Table 3. Compare results.

| Platform | CPU     | GPU     | FPGA |
|----------|---------|---------|------|
| Times    | 18.594  | 6.643   | 6.8  |
| Powers   | 130     | 300     | 4.35 |

5. Conclusions
The accuracy of Lenet-5 model implemented in this paper is affected by the accuracy of floating point operation in hardware implementation. After sufficient training, the network recognition accuracy rate is 97%, which is basically the same as CPU and GPU platform. The network realized in this design has no effect on the precision of the network.

References
[1] Jia, Y.Q., Shelhamer. (2014) Caffe: Convolutional Architecture for Fast Feature Embedding. In: Poc of the 22nd ACM Int Conf on Multimedia. New York. pp. 675-678.
[2] LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R.E., Hubbard, W. (2014) Backpropagation applied to handwritten zip code recognition. Neural Computation,1(4):541-551
[3] Li, X., Zhang, G., Huang, H.H. (2016) Performance Analysis of GPU-Based Convolutional Neural Networks. In: International Conference on Parallel Processing. Philadelphia. pp. 67-76.
[4] Zhang, C., Li, P., Sun, G. (2015) Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In: Acm/sigma International Symposium on Field-Programmable Gate Arrays. Monterey. pp. 161-170.
[5] Qiu, J., Wang, J., Yao, S. (2016) Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In: Acm/sigma International Symposium on Field-Programmable Gate Arrays. Monterey. pp. 26-35.
[6] Zhang, C., Wu, D., Sun, J. (2016) Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster. In: International Symposium on Low Power Electronics and Design. San Francisco, pp. 326-331.
[7] Lecun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998) Gradient-based learning applied to document recognition. Proceedings of the IEEE,pp.2278-2324