ACCDSE: A Design Space Exploration Framework for Convolutional Neural Network Accelerator

Zhisheng Li[1(✉)], Lei Wang[1], Qiang Dou[1], Yuxing Tang[1], Shasha Guo[1], Haifang Zhou[1], and Wenyuan Lu[2]

[1] National University of Defense Technology, Changsha, China
lizhsh_123@163.com
[2] Xi'an Satellite Monitoring and Control Center, Xi'an, China

Abstract. In deep learning, convolutional neural network (CNN) is quite representative. The convolutional operation of CNN is the focus of hardware acceleration research. Because of CNN's memory-intensive and compute-intensive features, increasing size of network poses a greater challenge on the design of the hardware accelerator. We need to determine the parameters of the accelerator at the early stages of the accelerator design.

This paper presents a design space exploration framework for CNN accelerator: ACCDSE, for determining the parameters of convolutional accelerator in FPGA. Simulation method and theoritical computation method are both used to find the optimal parameter. Experiment on LeNet shows that 16-bit fixed point is the most economical data precision for inference of LeNet. By theoritical analysis, the ACCDSE framework can obtain optimal matrix tiling parameters. Without decreasing the classification accuracy, the power consumption can be reduced by 33.57% and the storage can be reduced by 41.47% after weight pruning.

Keywords: CNN · Design space exploration · Data precision
Accelerator · Weight pruning

1 Introduction

In recent years, deep learning has subverted the algorithm design ideas in many areas, such as speech recognition and image classification. Since AlexNet wons the 2012 ImageNet large-scale image recognition competition (ILSVRC2012) with 83.6% top-5 accuracy, CNNs have become well-known. Nowadays, artificial intelligence is commonly applied to industrial manufacture. Because CNN's convolutional layers have a significant effect in extracting image features, CNN is widely used in practical applications. It makes the requirements for CNN's inference phase are increasing. However, CNN has the property of memory-intensive and compute-intensive. In addition, the scale of CNN continues to increase. These two facts make the hardware acceleration for CNN a urgent issue. Since

W. Xu et al. (Eds.): NCCET 2017, CCIS 600, pp. 22–34, 2018.
https://doi.org/10.1007/978-981-10-7844-6_3

the convolutional layers of CNN takes about 90% of the execution time [11], it is the focus of hardware acceleration. At present, there are a lot of accelerations for CNN. GPU, FPGA [10,13,16] and ASIC [3] are commonly used acceleration platform.

To meet various design target, accelerator design parameters need to be determined at early stages. Many aspects of CNN optimization and acceleration need to be considered, such as the design of the architecture [14], the parameter for matrix tiling [10,13,16], whether or not using low-precision operands [6].

However, previous research work focuses on a single aspect parameter setting. But these parameters are interactional and accelerator design is a system design process. In order to receive more systematic solution, we need a design space exploration framework to find the best parameters combination according to performance requirements.

This paper presents a design space exploration framework: ACCDSE. This framework is based on a in-house simulator that supports CNN's inference and three design space exploration modules. The three modules can find optimal configuration parameters that meet the performance requirements for data precision, matrix tiling, and weight pruning, respectively. The main contributions of this paper are as follows:

1. Proposed a design space exploration framework that supports CNN's inference engine's DSE;
2. Explored the impact of various data precision on the accuracy of prediction, selecting the most economical data precision;
3. Based on the roofline model and the matrix tiling strategy, the parallel parameters are selected under the given hardware constraint;
4. Explored the relationship among classification accuracy, power and weight pruning.

Through experiment with LeNet and Caffe's performance as benchmark, the design space exploration result shows that 16-bit fixed point operand data precision will not have an impact on performance. The matrix tiling module in the framework can calculate the optimal parameter configuration according to the convolutional layer parameters and the resource limit of the acceleration platform to achieve the maximum throughput. Without decreasing the classification performance priority, the storage can be reduced by 33.57% after weight pruning. When we concerned more about power, classification accuracy decreases by 4.0%, the storage can be reduced by 78.03% after weight pruning. If we concerned more about cost, the storage can be reduced by 49.89% after weight pruning without sacrificing classification accuracy.

The rest of this paper is organized as follows: The Sect. 2 introduces the background and the related research work. The Sect. 3 analyzes the parallel characteristics of the convolutional operation. The Sect. 4 introduces the proposed ACCDSE in this paper. The Sect. 5, shows the experiments and performance analysis. The Sect. 6 is the work related to this research. The Sect. 7 is the conclusion.

Table 1. LeNet parameters

LeNet parameters				
Layer type	Input	Weights	Stride	Output
CONV1	1*28*28	20*1*5*5	1	20*24*24
Pooling1	20*24*24		2	20*12*12
CONV2	20*12*12	50*20*5*5	1	50*8*8
Pooling2	50*8*8		2	50*4*4
FC1	800*1	800*500		500*1
ReLU	500*1			500*1
FC2	500*1	500*10		10*1
Soft-Max	10*1			10*1

2 Background

There are many algorithms for convolutional neural networks, but the main components are the same. In this paper, we use LeNet as a research neural network.

2.1 LeNet

LeNet is the most basic network of CNN, mainly used in handwritten number recognition. As is shown in Fig. 1, it is divided into eight layers, including tow convolutional layers, two pooling layers, two fully-connected layers, one ReLU layer and one SoftMax layer [6]. The Table 1 shows the parameter for the LeNet.

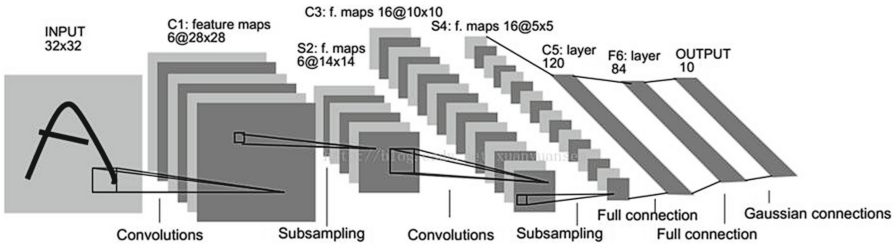

Fig. 1. LeNet structure

2.2 Convolutional Operation

As is shown in Fig. 2, the convolutional operation is the point product of a three-dimensional matrix and a four-dimensional matrix. In the process of operation, the convolutional kernel performs the point product in the order of the corresponding data of the input matrix, the result is summed to obtain a point of the

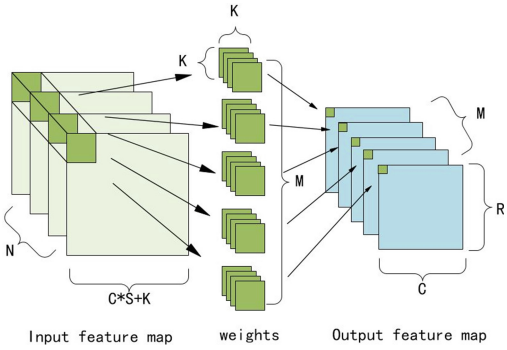

Fig. 2. Convolutional layer diagram

output matrix finally. Different groups of convolutional kernel repeat the same operation. When all the kernels and the input matrix complete the operation after the corresponding output matrix, a convolutional operation is completed. Figure 4 shows the pseudo-code of the convolutional operation.

2.3 Roofline Model

Using the FPGA platform to accelerate the CNN, two major problems need to be solved. First, improve the utilization of resources. Second, meet the FPGA bandwidth requirements [16]. In order to solve these two problems, this paper takes the Roofline Model [9].

Figure 3 shows the Roofline Model. As is shown in the figure, the slash represents the bandwidth of the platform. The line paralleled to the X axis represents the resource limit of the platform. It can be seen that the area among the slash, the line and the X axis is achievable. For point in the slash above, the limiting factor is the bandwidth. For point in the straight line above, the limiting factor is the resources. Two algorithms are shown in the figure. Obviously, in Fig. 3 the throughput of algorithm 1 is smaller than that of algorithm 2. When the throughput is the same, the computation to communication ratio (CTC) high point's performance is better [16].

3 Convolutional Operation Parallelism Analysis

As is shown in Fig. 4(a), the convolutional operation is a multiple nested loop. The same loop internal data is independent of each other and can be executed in parallel. The research of this paper is mainly for the following three parallel characteristics.

As is shown in Fig. 4(a), the 5th and 6th loop represent the convolutional kernel. The data inside the convolutional kernel is independent of each other and can be executed in parallel. We denote Tk as the number of data, which within the same convolutional kernel, executed at the same time ($Tk <= K * K$).

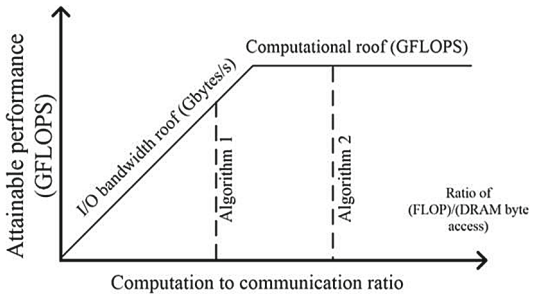

Fig. 3. Roofline model

```
for (r=0;r<R;r++){
 for (c=0;c<C;c++){
  for (m=0;m<M;m++){
   for (n=0;n<N;n++){
    for (i=0;i<K;i++){
     for (j=0;j<K;j++){
       output[m][r][c]+=
       weights[m][n][i][j]*input[n][S*r+i][S*c+j];
}}}}}}
```

```
for (r=0;r<R;r++){
 for (c=0;c<C;c++){
  for (m=0;m<min(M,m+Tm);m+=Tm){
  #Tile Tm times
   for (n=0;n<min(N,n+Tn);n+=Tn){
   #Tile Tn times
    for (i=0;i<min(K*K,i+Tk);i+=Tk){
    #Tile Tk times
       x=i/K;y=i%K;
       output[m][r][c]+=
       weights[m][n][x][y]*input[n][S*r+x][S*c+y];
}}}}}
```

(a) Pseudo code of a convolutional layer (b) Pseudo code of a tiled convolutional layer

Fig. 4. Convolutional pseudocode

The 4th loop represents the data between the convolutional kernels. The data between the convolutional kernels are independent of each other and can be executed in parallel. We denote Tn as the number of convolutional kernel executed at the same time ($Tn <= N$).

Similarly, the 3th loop is among the different outputs. The data are also independent of each other, which can be executed in parallel. We denote Tm as the number of output executed at the same time ($Tm <= M$).

The convolutional code for adding the parallel parameters is shown in Fig. 4(b).

The matrix tiling calculation model is based on the method of work [10,16]. Figure 2 shows the meaning of the relevant parameters. The Eq. (1) shows the number of cycles for the convolutional layer:

$$number\quad of\quad execution\quad cycle = \left\lceil \frac{M}{Tm} \right\rceil \times \left\lceil \frac{N}{Tn} \right\rceil \times \left\lceil \frac{RC}{TrTc} \right\rceil \times \left\lceil \frac{K}{Ti} \right\rceil \times \left\lceil \frac{K}{Tj} \right\rceil \times (TrTc \times \left\lceil \frac{TiTj}{Tk} \right\rceil + P) \tag{1}$$

Where P is the number of pipelines, Tr, Tc, Ti, and Tj are the dimensions of the data to be prepared in one operation. The total number of operations for calculating a convolutional layer is shown in Eq. (2):

$$total\quad number\quad of\quad operations = 2 \times R \times C \times M \times N \times K \times K \tag{2}$$

Equation (3) represents the upper limit of the calculation under parallel parameters:

$$computational \quad roof = \frac{2 \times M \times N \times K \times K}{\lceil \frac{M}{Tm} \rceil \times \lceil \frac{N}{Tn} \rceil \times \lceil \frac{K}{Ti} \rceil \times \left\lceil \frac{K}{Tj} \right\rceil \times \left\lceil \frac{TiTj}{Tk} \right\rceil} \tag{3}$$

The purpose of optimizing acceleration is to maximize the computational roof.

All operations are required for data support. The amount of data set must be less than the bandwidth of the platform.

The data involved in the operation is divided into three parts: input, weights and output. Hence, different buffer are required to hold the necessary data. The data required for the three-part data are shown in (4) (5) (6):

$$\beta_{in} = Tn \times (S \times Tr + Ti - S)(S \times Tc + Tj - S) \times 2\,Bytes \tag{4}$$

$$\beta_{weights} = Tm \times Tn \times Ti \times Tj \times 2\,Bytes \tag{5}$$

$$\beta_{out} = Tm \times Tr \times Tc \times 2\,Bytes \tag{6}$$

Similarly, the number of loads and stores can be calculated using (7) (8) (9):

$$\alpha_{in} = \frac{M}{Tm} \times \frac{N}{Tn} \times \frac{R}{Tr} \times \frac{C}{Tc} \times \frac{K}{Ti} \times \frac{K}{Tj} \tag{7}$$

$$\alpha_{weights} = \frac{M}{Tm} \times \frac{N}{Tn} \times \frac{R}{Tr} \times \frac{C}{Tc} \times \frac{K}{Ti} \times \frac{K}{Tj} \tag{8}$$

$$\alpha_{out} = 2 \times M \times R \times C \tag{9}$$

The amount of memory and the buffer size can be calculated by calculating the communication ratio. Calculating the communication ratio is a measure of the degree of reuse of the on-chip storage data.

$$\begin{aligned} CTC &= \frac{Total \quad required \quad computation}{Total \quad requried \quad communication} \\ &= \frac{2 \times M \times N \times R \times C \times K \times K}{\alpha_{in} \times \beta_{in} + \alpha_{weights} \times \beta_{weights} \alpha_{out} \times \beta_{out}} \end{aligned} \tag{10}$$

CTC as the horizontal axis, the throughput of the vertical axis, all the circumstances listed. According to rooflin model theory, we select the highest throughput point as the design point.

4 ACCDSE Framework

The objective of this paper is to determine the parameters of the design space exploration based on the metric such as performance, power and energy efficiency, so that the accelerator can meet the design requirements. Figure 5 shows the framework for design space exploration. The design parameters that can be selected in this framework, including the data precision, the parameters of matrix tiling and whether or not performing the pruning operation.

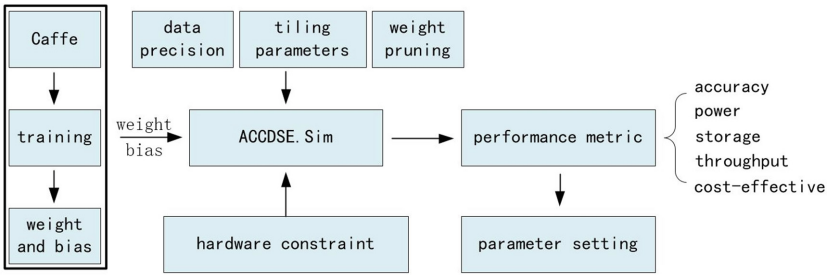

Fig. 5. Design space exploration framework

4.1 Data Precision Selection

High data precision can lead to stable performance. Low data precision will save hardware costs. Appropriate data precision is a trade-off between performance and hardware costs.

Weights of trained Caffe Model are extracted and fed into the simulator. We observed that the absolute value of the extracted weights is between 0 and 1. Through the method of enlarging the data, we convert float point data to fixed point data. The data after processing is involved in the operation. The result of the calculation needs to be reduced to the original size. In the process of operation, inappropriate data precision may lead to data overflow. The data overflow will cause the result of inference to be invalid. There are three data precisions in the framework: float point, 32-bit fixed point and 16-bit fixed point. Through the experimental analysis, we select the most suitable data precision.

4.2 Matrix Tiling Parameters

In Sect. 3, three parallel parameters (Tk, Tn, Tm) have been proposed. In the process of hardware acceleration, the utilization of hardware resources and bandwidth must be considered. Inappropriate parallel parameters, always leading to reduction in resource utilization, and failing to achieve good performance.
The choice of parallel parameters must follow three principles:

(1) the design of the buffers size must not exceed the on-chip memory,
(2) the design of the bandwidth must not exceed the maximum bandwidth,
(3) the computing unit must not exceed the on-chip resources.

In this calculation module, entering the resource constraint and convolutional layer parameters, the output is matrix tiling parameters.

As the work in [16], there are 3 steps to select parameters.

(1) According to resource constraints, the theoretical maximum throughput is obtained.
(2) According to the convolutional layer parameters and Roofline Model calculated computational roof and CTC.

(3) After listing all possible parameter configurations, we select the best point as the design point.

Assuming our implementation is built on the VC707 board which has a Xilinx FPGA chip Virtex7 485t, we optimize CONV2 which in Table 1. The data precision is 16-bit fixed point. By calculation, the theoretical maximum throughput is 140 GFLOPS. By calculating the model, CTC and computational roof are obtained. With the bandwidth limit, the maximum throughput point is the design point. At this time the parameters (Tk, Tm, Tn) are (25, 5, 5). At this point, the throughput is 125 GOPS.

4.3 Weight Pruning

In the work [12], the contribution of different weights to the final prediction results is different. A small part of the weight can determine the final performance. In the process of optimizing the CNN, the weight that makes a great contribution to the result, is preserved. The weight that has smaller contribution is selectively removed. Through the pruning operation, power and storage can be reduced at the same time, and meanwhile having no significant impact on performance.

The strategy in this framework is to set a value appropriate threshold. The weight which does not exceed the threshold, will be set to zero. Threshold size of the selection, can only affect little to final results, but also as much as possible to reduce the amount of computing, power and storage. It can be found that a suitable threshold is critical to optimization.

5 Experimental Evaluation

5.1 Experimental Environment

Caffe is the classical framework of convolutional neural networks. In this paper we use Caffe as the training framework and inference reference model. Caffe trained the LeNet network with the MNIST training set. The weight of the simulator comes from the trained Caffe Model. Caffe's output is used as the baseline. The ACCDSE framework is running on ubuntu 16.04 system. The processor is Intel Core i5-6600.

5.2 Implementation

The ACCDSE framework proposed in this paper is implemented in C. The key component of the ACCDSE framework is a CNN inference engine simulator. Figure 6 presents the structure of the ACCDSE framework implementation. Caffe uses the training set in MNIST to train and get the weights. The simulator that obtained the weight is evaluated with the MNIST test set. The results are compared with Caffe's test performance and analysis. The simulator can perform data precision, matrix tiling and weight pruning selection. As mentioned in the

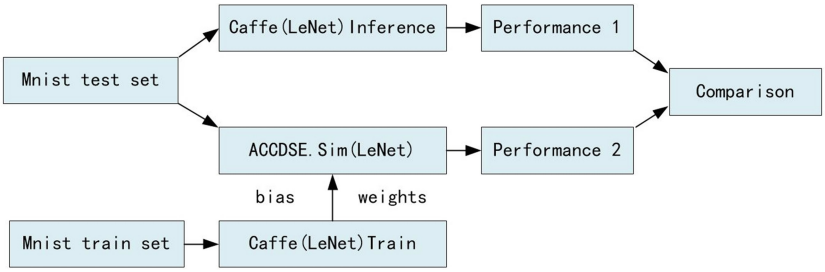

Fig. 6. Diagram of the experimental process

Sect. 4, the options for selecting data precision are float point, 16-bit fixed point and 32-bit fixed point. Matrix tiling with (Tk, Tm, Tn) three parameters. Weight pruning has a choice of thresholds.

5.3 Experimental Result

5.3.1 Data Precision

Our simulator can select the data precision. In Sect. 4, we have proposed a method of data type selection. Caffe's classification accuracy as the baseline. 10,000 pictures of Mnist test set as benchmark. Float point, 32-bit fixed point and 16-bit fixed point three data precision experiment for performance comparison respectively. By using the Sect. 4.1's method, we transform the weight into fixed point. The middle value is saved with double bit width. When the median value participates in the next operation, the value is scaled and rounding. When the number of significant digits is 1 and the error rate is high, there is no representation in the figure. The overflow situation is also not shown in the figure.

First, the simulator analyzes the weights (absolute value), and the weight distribution is shown in Fig. 7. The results show that the weights are mostly between 0.01 and 0.1. So, when the weights are converted to integers and magnified by a factor of 1,000, the vast majority of the weights are covered.

The experimental results are shown in Fig. 8. The horizontal axis is the data precision, and the number of significant digits of the weight is marked in parentheses. The red dotted line indicates the performance of Caffe.

Through the experimental results, high data precision will lead to stable performance (compare with baseline) and not easy to overflow. When the data precision is 16-bit fixed point and 32-bit fixed point, the more the effective bit of the weight, the better the performance. According to the comparison among all of the performances of data under precision, int and 16-bit fixed point have the best performance, which is better than baseline. At this time, the weight of the amplification parameter of 1000, confirms the conclusions of Fig. 7. The principle of selecting data precision: performance priority, hardware costs secondly. According to the principle, we select 16-bit fixed point as the data precision.

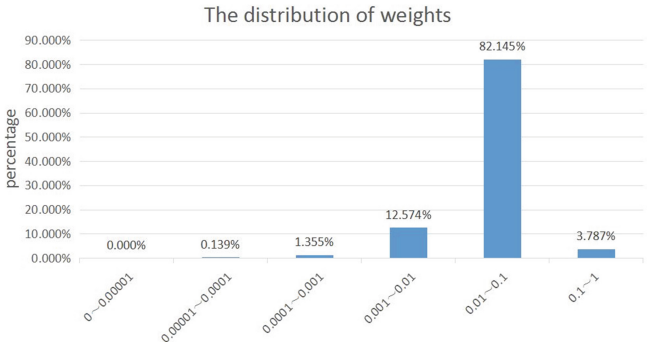

Fig. 7. The distribution of weights (absolute value)

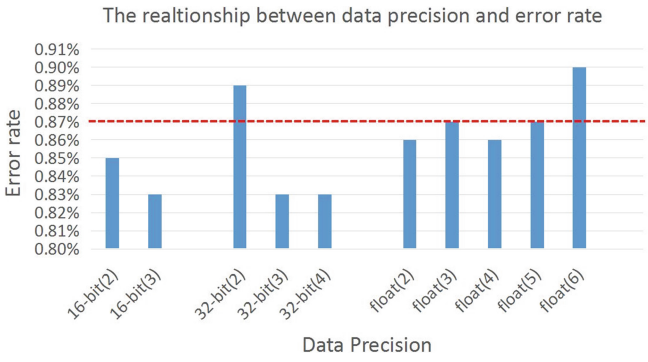

Fig. 8. Performance under three data precision (Color figure online)

5.3.2 Weight Pruning

The simulator can research the relationship among threshold, calculation and accuracy. The steps are as follows. When the simulator reads the weights, the total number of weights and the number of pruning are counted. When the parameters of the convolutional layer are determined, the total amount can be calculated according to the parameters. Similarly, according to the number of pruning weights, we can also get the amount of weight after the calculation. Here, we calculate the amount as a criterion for power consumption. The experiment was conducted with the MNIST 10000 test set as a benchmark. The experiment uses 16-bit fixed point as the data precision.

We denote P as the number of correct samples, P′ as the total number of samples, M as the storage after the pruning of the weights, M′ as the storage before pruning, Q as the calculated amount after the pruning of the weights, Q′ as the amount of computation before pruning, E as the power efficiency, and C as the cost-effective.

The relationship shown as follows: $accuracy = P/P'$, $storage = M/M'$, $power = Q/Q'$, $E = P/Q$, $C = E/(1 - accuracy)$.

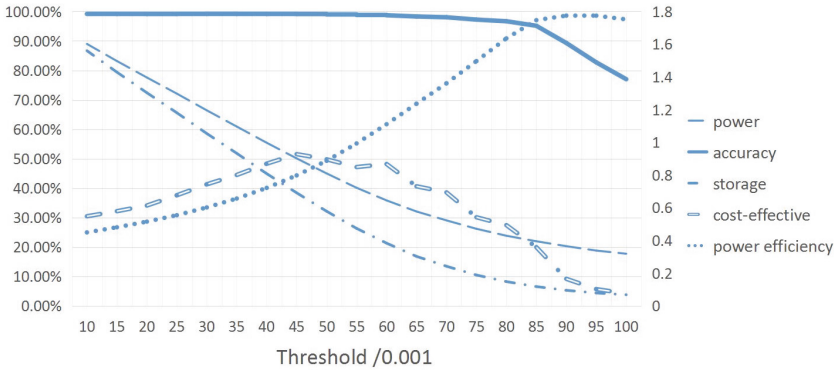

Fig. 9. The relationship between threshold and performance index

The experimental results are shown in Fig. 9. The abscissa is the threshold. In the figure, the power efficiency and cost-effective are the trend lines.

As the threshold increases, power and performance are reduced. However, in a particular area, the performance degradation can be ignored, but the power will be drastically reduced. When the threshold increases to a certain value, the performance will be significantly reduced. If practical applications is concerned about the performance, we can select a threshold of 0.03. The error rate is only 0.81%, the power reduced to 66.43%, and the storage reduced to 58.57%. If the performance requirements of the application are very strict, we do not have to take pruning.

If the actual application is more concerned about power, we select the highest power efficiency point. When the threshold is 0.09, the power efficiency is the highest. But at this time, the error rate is 10%. The performance is unbearable obviously. The final selection threshold is 0.085 as the design point. At this time, the error rate is 4.86%, the power is only 21.97%, the storage reduced to 6.58%.

If the actual application is more concerned about the cost-effective, we can select threshold of 0.045, which is the best cost-effective point obviously. In this time, power and performance achieve the best match. At this point, the error rate is 0.84%, the storage reduced to 38.42%, and the power reduced to 50.11%.

6 Related Work

CNN accelerator design work has been carried out for several years. ASIC, FPGA and GPU are the main acceleration platforms. Among them the most famous ASIC is DianNao series of accelerators. Work [3] introduced the first accelerator. The acceleration strategy is to separate the operations in the network, then calculate each neuron, finally get the results. The nearest accelerator is ShiDianDao [4]. Obtaining the data in the terminal directly, this could avoid some of the memory consumption. In addition, IBM developed the TureNorth chip [1]. The data is transmitted using spiking. The advantages of FPGAs are

flexibility and low power consumption. The acceleration work is mainly applied to the inference. GPU is generally used for training.

In the chip design, the key problem is solving resources and bandwidth, so design space exploration is very important. In work [10,16], the Roofline Model is used to solve the problem of resource and bandwidth in FPGA. In view of the problem of resource utilization is not well, the work [13], put forward the idea of hardware resource block, making the use of resources more fine-grained. There is still a lot of work designed for the computing unit. For example, the work [5,14]. In addition, work [12] is referred to the weight pruning operation. This is also the future direction of CNN accelerator design. The corresponding work [7] proposed an effective compression engine. In work [6], we study the problem of finite numerical precision, and also provide a very bright direction for the acceleration work. Work [15] also design and implement Caffeine, a hardware/software co-designed library to efficiently accelerate the entire CNN on FPGAs.

The method proposed in work [2] uses the data correlation among the multi-layer convolutional layers to improve the data reuse and reduce the pressure of the bandwidth. In work [8], the neural network compilation method is proposed, which can match the network and hardware.

7 Conclusion

This paper presents ACCDSE, a design space exploration framework for convolutional neural network accelerator. Taking LeNet as an example, the FPGA is used to accelerate the platform. The framework has been used for data precision, matrix tiling and weight pruning selection.

Experiment result shows that ACCDSE can achieve the most economical precision selection for convolutional neural network to achieve maximum throughput. It can make full use of the acceleration platform hardware resources. The weight pruning enables a combination of power and performance. Through this framework optimization, the accelerator can achieve trade-offs among throughput, performance, and power. Under a variety of performance requirements, the accelerator can achieve optimal parameter configuration.

Acknowledgment. This project was supported by NSFC 61402501. I was also grateful to my teachers and students who had helped me in this project.

References

1. Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R.: Truenorth: design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. **34**(10), 1537–1557 (2015)
2. Alwani, M., Chen, H., Ferdman, M., Milder, P.: Fused-layer CNN accelerators. In: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 1–12, October 2016

3. Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., Temam, O.: Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. In: International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 269–284 (2014)
4. Du, Z., Fasthuber, R., Chen, T., Ienne, P., Li, L., Luo, T., Feng, X., Chen, Y., Temam, O.: Shidiannao: shifting vision processing closer to the sensor. In: Proceedings of the International Symposium on Computer Architecture, ISCA 2015, pp. 92–104 (2015)
5. Farabet, C., Poulet, C., Han, J.Y., Lecun, Y.: CNP: an FPGA-based processor for convolutional networks. In: International Conference on Field Programmable Logic and Applications, pp. 32–37 (2009)
6. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. Computer Science (2015)
7. Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M.A., Dally, W.J.: EIE: efficient inference engine on compressed deep neural network. In: International Symposium on Computer Architecture, pp. 243–254 (2016)
8. Ji, Y., Zhang, Y.H., Li, S.C., Chi, P., Jiang, C.H., Qu, P., Xie, Y., Chen, W.G.: Neutrams: neural network transformation and co-design under neuromorphic hardware constraints. In: The IEEE/ACM International Symposium on Microarchitecture, pp. 1–13 (2016)
9. Meloni, P., Deriu, G., Conti, F., Loi, I., Raffo, L., Benini, L.: Curbing the roofline: a scalable and flexible architecture for CNNS on FPGA. In: The ACM International Conference, pp. 376–383 (2016)
10. Motamedi, M., Gysel, P., Akella, V., Ghiasi, S.: Design space exploration of FPGA-based deep convolutional neural networks. In: Asia and South Pacific Design Automation Conference, pp. 575–580 (2016)
11. Peemen, M., Setio, A., Mesman, B., Corporaal, H.: Memory-centric accelerator design for Convolutional Neural Networks (2013)
12. Reagen, B., Whatmough, P., Adolf, R., Rama, S., Lee, H., Lee, S.K., Jos Ndez-Lobato, M., Wei, G.Y., Brooks, D.: Minerva: enabling low-power, highly-accurate deep neural network accelerators. In: ACM/IEEE International Symposium on Computer Architecture, pp. 267–278 (2016)
13. Shen, Y., Ferdman, M., Milder, P.: Overcoming resource underutilization in spatial CNN accelerators (2016)
14. Wang, C., Gong, L., Yu, Q., Li, X.: DLAU: a scalable deep learning accelerator unit on FPGA (2016)
15. Zhang, C., Fang, Z., Zhou, P., Pan, P., Cong, J.: Caffeine: towards uniformed representation and acceleration for deep convolutional neural networks. In: International Conference on Computer Aided Design (2016)
16. Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., Cong, J.: Optimizing FPGA-based accelerator design for deep convolutional neural networks. In: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 161–170 (2015)