Chapter 8

Optimizing of Convolutional Neural Network Accelerator

Wenquan Du, Zixin Wang and Dihu Chen

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.75796

Abstract

In recent years, convolution neural network (CNN) had been widely used in many image-related machine learning algorithms since its high accuracy for image recognition. As CNN involves an enormous number of computations, it is necessary to accelerate the CNN computation by a hardware accelerator, such as FPGA, GPU and ASIC designs. However, CNN accelerator faces a critical problem: the large time and power consumption caused by the data access of off-chip memory. Here, we describe two methods of CNN accelerator to optimize CNN accelerator, reducing data precision and data-reusing, which can improve the performance of accelerator with the limited on-chip buffer. Three influence factors to data-reusing are proposed and analyzed, including loop execution order, reusing strategy and parallelism strategy. Based on the analysis, we enumerate all legal design possibilities and find out the optimal hardware design with low off-chip memory access and low buffer size. In this way, we can improve the performance and reduce the power consumption of accelerator effectively.

Keywords: convolution neural networks, CNN accelerator, memory access

1. Introduction

Convolution neural networks are developing rapidly in recent years. Due to the outstanding performance in image recognition, CNN are used widely in image classification [1–3]. Moreover, since its great success in image recognition, CNN are studied and applied to many other fields of artificial intelligence, such as speech recognition [4, 5], game play [6], etc.

Increasing the depth of CNN by increasing the amount of layers of CNN is a common and effective method to improve the accuracy of image recognition. For instance, in ILSVRC

2012, the champion work, one kind of CNN model namely AlexNet, achieved the top-5 accuracy of 84.7% in the image classification task, and the CNN model has 5 convolutional layers and 3 fully connected layers [2]. The ResNet, which won the first place in ILSVRC 2015 and achieved 96.43% accuracy exceeding human-level accuracy, consists of 152 layers [3]. Although making CNN model deeper can improve the performance, the computing process of CNN involves an enormous number of computation and data. It brings more pressure to the computing hardware. Traditional CPU became a limitation to CNN. Lacking of parallel computing, using CPU for CNN computing result in poor computing performance and high power consumption. It is necessary to find a better hardware to replace CPU for CNN computing. Therefore, more and more hardware are designed and used for CNN computing, such as FPGA designs [7, 8], GPU designs [9], and ASIC designs [10, 11]. These designs aim to accelerate the computing of CNN, improve the computing performance and reduce the energy consumption. Designing and optimizing a specific CNN hardware accelerator became one of popular topics.

In this chapter, we will propose a typical architecture of CNN accelerator and we will introduce two optimize methods of CNN accelerator, consisting of reducing data precision and data-reusing. Generally, reducing data precision will affect the accuracy of CNN model and data-reusing will bring extra on-chip buffer of CNN accelerator. In this paper, we will focus on these two optimization methods and make a deep analysis of the impact on CNN accelerator. We investigate the impact of reducing data precision to accuracy of CNN from several references [12–16], and found that reducing data precision appropriately almost have no impact on the accuracy of CNN model. It inspires us that reducing data precision is a feasible method to optimize CNN accelerator, which will bring many benefits to CNN accelerator, such as reducing the hardware resource, memory footprint and power consumption, etc. In addition, we will analyze three influence factors of data-reusing, including loops execution order, reusing strategy and parallelism strategy. Based on the analysis, we enumerate all legal design possibilities and find out the optimal hardware design with low off-chip memory access and low buffer size. In this way, we can optimize our CNN accelerator with high performance, small memory footprint and low power consumption.

This chapter is organized as follows:

- Review the history and development of convolution neural networks.
- Describe the architecture and the principle of CNN. A real-life CNN, namely VGG-16 will be presented.
- Present a typical architecture of a CNN accelerator implemented on an embedded FPGA platform and introduce the workflow of a CNN accelerator. Present a typical processing element of CNN accelerator.
- Propose and analyze two optimize methods of CNN accelerator including reducing data precision and data-reusing. Based on the analysis, find out the optimal design space for the CNN accelerator by enumerating all legal design possibilities.

http://dx.doi.org/10.5772/intechopen.75796

2. Background of convolution neural networks

2.1. CNN development overview

Convolutional neural network (CNN) is one type of Deep Neural Networks. The first CNN model namely LeNet-5 is proposed by LeCun in the paper [1] in 1998 and this model is used in handwriting digit recognition. However, due to the enumerated number of computing in training, CNN has been silence for some time. Until 2012, a breakthrough of CNN occurred. A group from University of Toronto used a deep neural network, namely AlexNet, won the first place in image classification in ILSVRC 2012, and its top-5 error rate achieved 15.3%, compared to the second place which achieved 26.2%, and also dropped the error rate by 10% approximately [2]. They improve the algorithm of CNN model in some aspects, such as deepening the model, using ReLu as the activation function, etc. And they train their CNN model with 2 GPU. Their effort resulted a great leap in Deep Neural Network. Since then, CNN developed rapidly. In 2015, the top-5 error rate of ImageNet champion work namely ResNet achieved 96.43% accuracy and exceeded human-level accuracy [3]. In their work, they continued to deepen the CNN model to 152 layers. In the latest ILSVRC 2017, the champion work top-5 error achieved 2.251% [17]. For the rapid development, CNN is making a great impact on many application areas, such as image and video recognition, speech recognition, game play, etc.

2.2. CNN basic

Convolution neural networks (CNNs) have an inference process for recognition and a back propagation process for training. Since training CNN spends a lot of time, many CNN application usually complete training off-line in advanced, and then use the trained CNN in terminals to execute tasks. Therefore, the inference process for recognition in terminals has more pressing demands on speed and power. In this work, we focus on the inference process of CNN and explore how to speed up the inference process with hardware accelerator.

Figure 1 is the simplified structure of a typical CNN. A typical CNN model usually consists of several parts, including convolutional layer (CONV), nonlinear activation function,

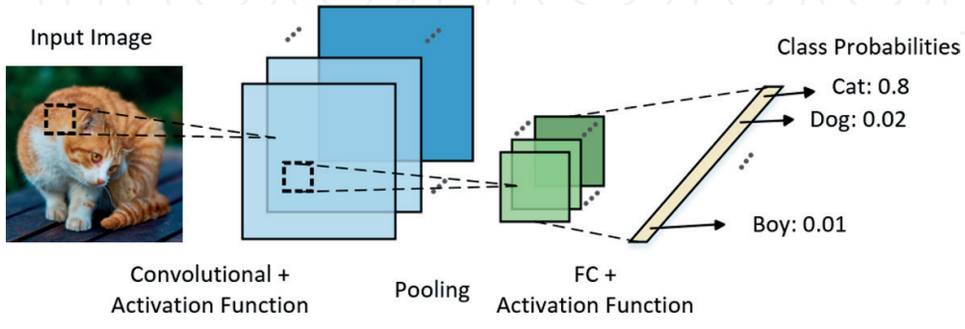

Figure 1. Structure of a typical convolution neural networks.

pooling layer and fully connected layer (FC). Layers are executed layer by layer. CNN reads an input image, and go through a series of CONV layers, nonlinear activation function, pooling layers and generate the output feature maps. These feature maps are turned into a feature vector in the FC layers. Finally, the feature vector is read by a classifier and the input image is classified to the most possible category and the probabilities of each classification are generated.

2.2.1. Convolutional layer

Convolutional layer (CONV) is the most critical layer of CNN. Usually the operations of CONV layers constitute more than 90% of the total CNN operations [18]. For example, in AlexNet, the operations of CONV layers constitute 85–90% of total operations. Therefore, in this work, our discussion and analysis focus on the CONV layers.

CONV layer is like a feature extractor which can extract the features from the input image by the convolutional kernels. When input image read by CONV layer and convolves with weights kernel (generated by training previously), the feature information such as corners, lines are extracted into the output feature maps.

Figure 2 shows a typical CONV layer. The CONV layer takes a set of C feature maps as input and convolves with M sets of N weights kernel to obtain M output feature maps. C is the number of the input channels, M is the number of output channels, H and W are the input feature maps size, R and C are the kernel size, and E and F are the output feature maps size. Usually H is equal to W, R is equal to S and E is equal to F. It shows that the input feature maps convolved by a shifting window with a weight kernel iteratively to get the output feature maps. Actually, the weight kernel shifting with stride S and in most cases the stride S is 1.

Usually the computing process of CONV layer can be expressed with a pseudo code in **Figure 3**. The total computing process of CONV layer can be executed by a loop nest. Every output pixel of output feature maps is the sum of products of Input and Weight. After all loops are executed, all output pixels of output feature maps can be obtained.

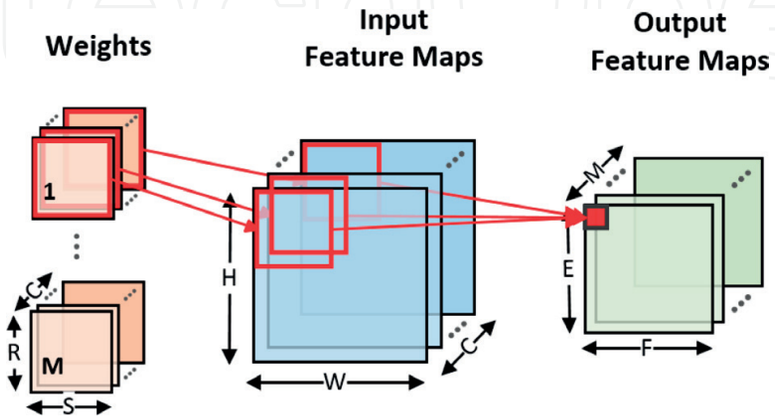

Figure 2. Diagram of convolutional layer.

2.2.2. Nonlinear activation function

CNN usually applies nonlinear activation function after each CONV layers or FC layers. The main function of nonlinear activation function is introducing nonlinearity into the CNN. Generally, nonlinear and differentiable are the two conditions that activation functions should to meet. Some conventional nonlinear activation functions in CNN such as sigmoid and tanh are shown in **Figure 4(a)**. However, these activation functions require long training time. In recent years the activation function Rectified Linear Unit (ReLU) [19] becomes more and more popular among CNN models. The expression of ReLU is $f(x) = \max(0, x)$, and it can be shown in **Figure 4(b)**. Compared to the conventional activation functions, the computation process of ReLU is more simple, and it makes training faster. In addition, since a number of output of ReLU are equal to 0, it makes the sparsity of CNN model.

2.2.3. Pooling layer

Pooling layer usually attach to the CONV layer. As **Figure 5** shown, usually the type of pooling is maximum or average. The pooling layers read input feature maps, and calculate the maximum or average value of every sub-area of input feature maps so that get low-dimension feature maps. Usually the stride of pooling is equal to the size of pooling.

```
for(x=0;x<F;x++){
 for(y=0;y<E;y++){
  for(k=0;k<C;k++){
    for(i=0;i<R;i++){
      for(j=0;j<S;j++){
        for(m=0;m<M;m++){
            Output[m][x][y] +=
                Input[k][x+i][y+j] ×
                Weight[m][k][i][j]
} } } } } }
```

Figure 3. Pseudo code of the computing process of convolutional layer.

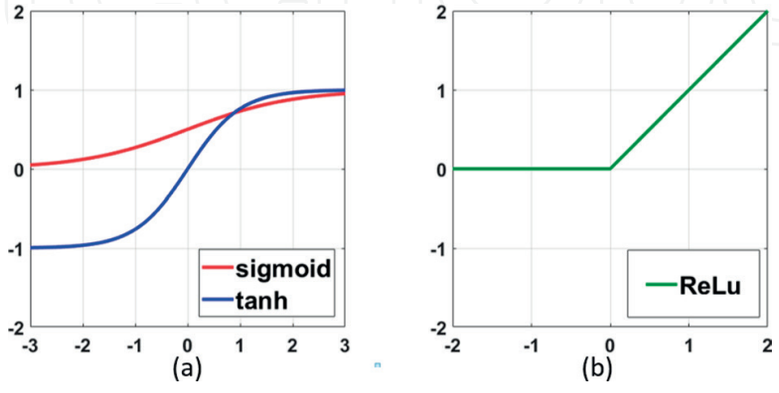

Figure 4. Nonlinear activation function: (a) sigmoid and Tanh; (b) Rectified Linear Unit.

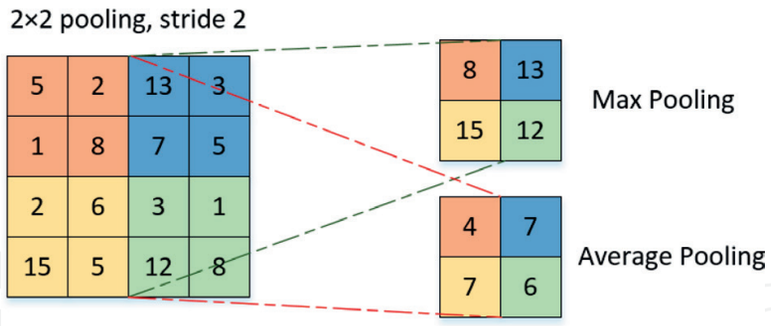

Figure 5. Two type of pooling layer: max pooling and average pooling.

Pooling layer has two major functions. First, pooling can aggregate information from the input feature maps and reduce the size and the dimension of feature maps. It brings a benefit that the size of feature maps and the amount of computation of CNN can be reduced greatly. The second function is that pooling layer can increase the invariance to small shifts.

2.2.4. Fully connected layer

Usually fully connected (FC) Layers are located posterior in the CNN model. The operation of FC layers can be expressed by Eq. (1). Input features (N_i) are fully connected to an output feature vector (N_o) by weights ($N_i \times N_o$). The N_i and N_o are the lengths of the input and output feature vector and the amount of weights is $N_i \times N_o$. $Bias_o$ is the corresponding bias term of the oth output feature vector.

FC layers usually turn the input feature maps into a feature vector. And finally the output feature vector read by a classifier which is a Softmax function and then the probability of each label are generated.

$$Out_o = \sum_{i=1}^{Ni} In_i \times W_{io} + Bias_o \tag{1}$$

2.3. A real-life CNN

Next, we will introduce a real-life CNN namely VGG-16 [20]. In ILSVRC 2014, VGG-16 won the second place in image classification task and its top-5 error achieved 7.4%. **Figure 6** [21] shows the architecture of VGG-16 model. VGG-16 is consisting of 13 CONV layers in 5 groups, 5 max pooling layers, and 3 FC layers. The network receives three 224 × 224 input images and the input images go through a series of layers. Finally, the network generates a output feature vector with the depth of 1000, which represent the likelihoods of 1000 categories. Compared to other CNN models, one of the characteristics of VGG-16 is that all kernel size of each CONV layer is 3 × 3.

Actually, each CONV layer of VGG-16 involves a large amount of operations and data. For a more visualized understanding, we make a detailed statistic for VGG-16. **Figure 7(a)** shows the amount of Input data, Weights and Output data for each CONV layer of VGG-16.

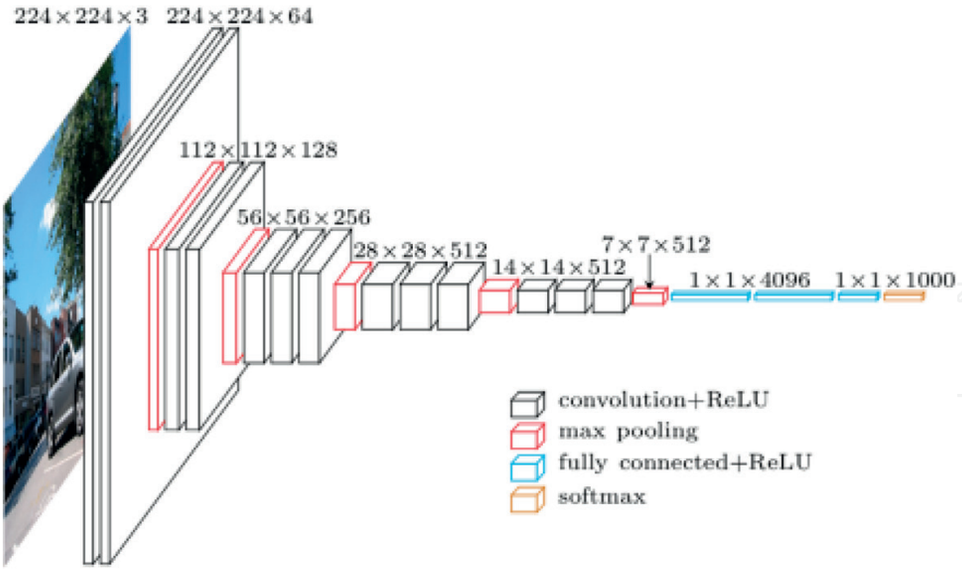

Figure 6. Architecture of VGG-16 model [21].

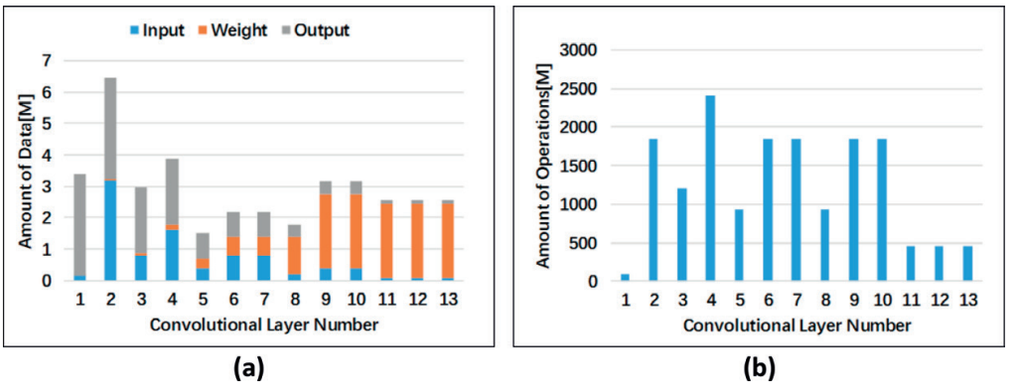

Figure 7. Statistic for each CONV layer of VGG-16: (a) amount of Input, Weight and Output for different CONV layers; (b) amount of operations for different CONV layers.

The 13 bars represent the CONV layer from layer 1 to 13. The height of each bar represents the total amount of data received and generated for each CONV layer, consisting of Input data, Weights and Output data. We can find that in different CONV layers, the amount of Input, Weight or Output vary greatly. For example, the CONV2 requires about 0.035 million weights, and the CONV9 requires 2.25 million weights, which is approximately 64 times to CONV2. **Figure 7(b)** shows the total amount of operation (multiply-add operation) for each CONV layer of VGG-16. The height of each bar represents the total amount of operation for each CONV layer. For example, there are about 87 million times multiply-add operations in the CONV1 and about 1764 million times in the CONV2. We can see that although the operation of CONV is similar, the amount of data involved and operation vary greatly between different CONV layers.

3. CNN accelerator

In this section, we will introduce the CNN accelerator and the challenges CNN accelerator face. As previous section shown, the operations of convolutional layers constitute a large proportion of whole CNN. Therefore, following in our work, the discussion and optimization of CNN accelerator focus on the CONV layers of CNN.

Due to deep convolution neural networks involve an enormous amount of data and operations as shown in Section 2, it is necessary to accelerate the CNN computation by hardware. Actually, many previous works developed the hardware platforms to accelerate the CNN and the implementations obtained good performance and high energy efficiency. Generally CNN can be accelerated by hardware platforms such as FPGA platforms [7, 8], GPU platforms [9], and ASIC platforms [10, 11]. These works have a same feature: they obtained high computational performance for CNN due to the parallelism of the hardware accelerator. In this work, we mainly discuss and analyze the CNN accelerator based on an embedded FPGA platform.

3.1. Overall architecture

Figure 8 is a typical architecture overview of CNN accelerator based on an embedded FPGA platform, and it is a CPU + FPGA architecture. The whole system is consisting of two parts including the Processing System (PS) and the Programmable Logic (PL).

Processing System (PS) mainly consists of CPU and off-chip memory. Due to the enormous amount of input data and weights, it is impossible to store data and weights in on-chip memory. Therefore, usually data and weights are stored in the off-chip memory like DDR3 at the beginning. CPU can configure the control module of accelerator, so that adjust the accelerator to accommodate different scale of CONV layers. In addition, CPU can realize

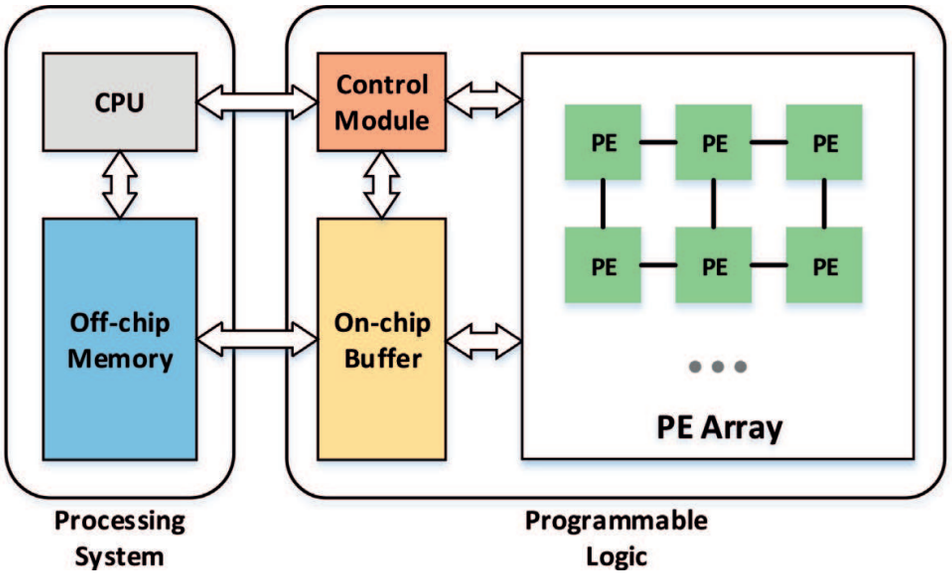

Figure 8. Overview of a typical architecture of CNN accelerator based on an embedded FPGA platform.

some simple operation such as the Softmax function of CNN model. We know that the operation of CONV layers usually constitute more than 90% of the total CNN operations. Accelerating the operations except the CONV layers have little performance improvement. Therefore, we can use CPU to handle the operations except the operation CONV layers such as Softmax function.

Programmable Logic (PL) actually is a FPGA chip and we can program the PL to meet our requirement. PL consists of several parts including processing element (PE) array, control module and on-chip buffer. PE array is consisting of a certain number of PE. PEs are the computation unit for convolution and usually the number of PEs decide the computational performance of CNN accelerator. Data can be interchanged between PEs so that data can be reused without accessing buffer. On-chip buffer is used to cache data and weights for PEs and store the results. Since data and weights of CONV layers are reused repeatedly, buffering and reusing data can reduce the off-chip memory access and it will be introduced in Section 4. Control module receives configuration information from PS, and control the computational process and dataflow of PE array.

The whole working process of CNN accelerator of an implementation can be divided into three steps. First, before system working, all data and weights are stored in the off-chip memory. Next, CPU starts to configure the control module of accelerator and then control module control the on-chip buffer to fetch data from off-chip memory. So that PE array can read data and weights from on-chip buffer and start computation. Finally, PE array finishes all computation and returns the results to off-chip memory so that CPU can read the results.

3.2. PE architecture

Processing element (PE) is the computation unit for CNN accelerator. **Figure 9** shows a typical architecture of PE. It is consisting of multipliers, adder trees, ReLU module, pooling module and registers for Input, Weights and Psum. The multipliers and adder trees are used to complete the convolutional operation. The ReLU module applies a ReLU function to the psum

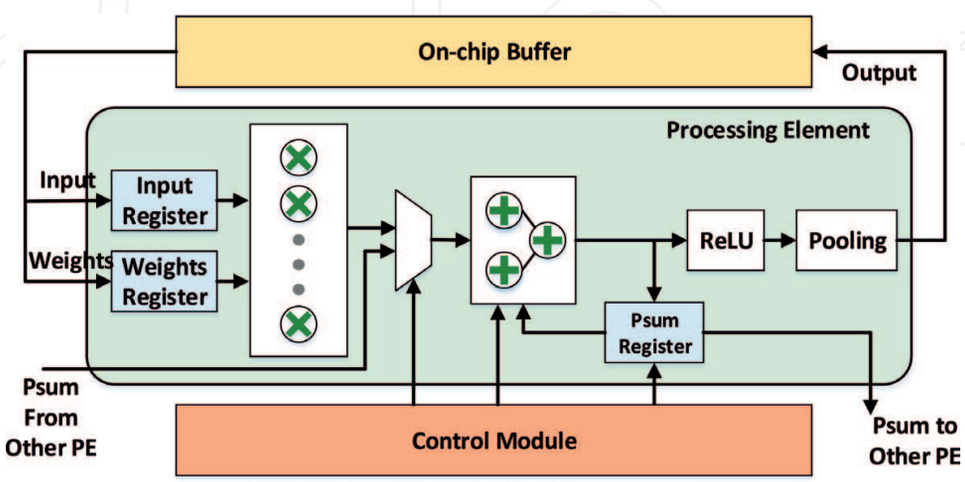

Figure 9. Typical architecture of processing element.

from adder trees. Pooling module realizes the max or average pooling operation and generates the result to on-chip buffer. Register is used to store the Input, Weights and Psum so that PE can reuse the data. PE can interchange data with other PEs. It improves the convenience of data accessing for PE, and it reduces the amount of on-chip buffer access.

Before PE working, control module receives the configuration information and configures the PE. After finishing configuration, PE starts to operation. Input and Weights register read and store data. Multipliers and adder trees read data from register or other PEs, and psum are generated and store in psum register. After finishing convolutional operation, psum are sent to ReLU module and Pooling module and Output are generated and sent to on-chip buffer. PE operation is done.

3.3. Challenges

Although the CONV layer algorithm is simple, due to the enormous amount of data and computation, hardware accelerators face some grave challenges. One of the challenges is the limitation of off-chip memory bandwidth. Generally, CNN accelerator computes with high parallelism by increasing the processing elements (PEs), which can improve the computational performance of accelerator. However, it is accompanied by the pressure of the bandwidth caused by the large amount of data access. Another challenge is that the large amount of off-chip memory access consumes a lot of energy.

Figure 10 is a diagram from reference [22]. It shows the normalized energy cost of each level memory hierarchy relative to the computation of one multiply-accumulate (MAC) operation and the data are extracted from a commercial 65-nm process. We can see that the energy cost of DRAM access is much higher than the energy cost of on-chip buffer access and MAC operation. Therefore, large amount of DRAM access will cause the high power consumption. Besides, large memory footprint caused by enormous amount of data is also an inevitable challenge.

To solve these problems, previous works proposed some optimization methods. One of the optimization methods is reducing data precision. Several researches show that reducing the data precision appropriately of CNN model almost has no impact on image recognition

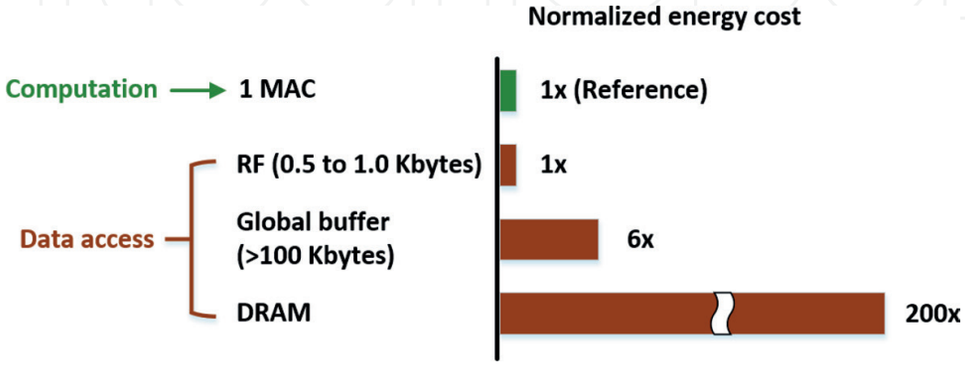

Figure 10. Normalized energy cost of each level memory hierarchy [22].

accuracy. It inspires us that reducing data precision is an efficient and significant method to reduce memory footprint and computational hardware resource. In addition, data-reusing [7, 8, 23] is also an optimization direction for CNN accelerator. Due to most of data in CNN are used repeatedly, data-reusing is also an efficient method to reduce memory access, consequently reducing the memory bandwidth and power consumption. In the next section, we will introduce these two optimization methods in detail.

4. Accelerator optimization

4.1. Data precision

Generally, to guarantee high recognition accuracy, 32-bit floating point data and weights are used to train CNN model. However, such high data precision brings more pressure to hardware because high data precision usually requires more computational resources and larger memory footprint. More and more studies indicated that reducing data precision appropriately almost has no impact on the accuracy of CNN model [12–14]. In Ref. [15], authors used MNIST dataset to test the impact of data precision on accuracy. The test result showed that using 16-bit fixed point data instead of 32-bit floating point in the inference process and using 32-bit floating point in the training process, the accuracy almost had no reduction. If using 16-bit fixed point data in the inference process while using 32-bit fixed point in the training process, the error rate reduced from 99.18 to 99.09%. In the reference [16], authors explored the precision requirements of CNN using AlexNet and VGG models. 8-bit fixed point for the convolution weights are used to test in the process of inference. Compared to full precision weights, the result showed accuracy reduced less than 1%.

These studies show that in many cases there is not necessary for CNN model to use high data precision such as 32-bit floating point. It inspires us reducing data precision is a feasible method to optimize CNN accelerator. Reducing data precision brings a lot of benefits to CNN accelerator. Firstly, for the reason that computation of floating point requires more computational resource than the computation of fixed point, using fixed point data in CNN accelerator will save a large number of computational resource. In addition, using lower precision data will make CNN accelerator more energy-efficiency. For instance, the power consumption of an 8-bit fixed point adder is 30× less than a 32-bit floating point adder and the power consumption of an 8-bit fixed point multiplier is 18.5× less than a 32-bit floating point multiplier [24]. Finally, reducing data precision of data or weights reduces the memory footprint and the bandwidth requirement directly. For instance, if using 16-bit fixed point data instead of 32-bit fixed point, the memory footprint shrink by half and the bandwidth requirement reduce by half.

Due to these advantages, many works [16, 25, 26] optimize their accelerator by reducing data precision of CNN accelerator under the premise of meeting the requirement of recognition accuracy. Ref. [16] used 8–16 bit fixed point data precision for the both AlexNet and VGG models. The accuracy reduction caused by the fixed point operations in the FPGA implementation is less than 2% for top-1 accuracy and less than 1% for top-5 accuracy.

4.2. Data-reusing

Data-reusing is an important optimization method of CNN accelerator to reduce the memory accesses. The main idea of data-reusing is use on-chip buffer to buffer the data which will be used repeatedly. When next time we use these data again, they can be accessed in the on-chip buffer, so that we do not need to access off-chip memory.

Figure 11 shows the computational process of convolution. In this process, a 3 × 3 weight kernel shift and convolve with 9 pixels of input feature map in sequence. Between two times convolving as the red frame and green frame shown in **Figure 11**, the value of weight kernel does not change, and the red area of input feature map will be used repeatedly. So that the weight kernel and the part of input feature map can be reused.

Table 1 shows the amount of data, of total data accesses and the average access times per data of Input, Weights and Output for CONV1 of VGG-16 model. The amount of data represents the amount of actual Input, Weights or Output. The amount of total data access represents the amount of data access of Input, Weights or Output. The average amount of using times per data is calculated by the amount of total data access dividing the amount of data and it represents the using times of each Input, Weights or Output. From **Table 1**, we can find that in the computing process of convolutional layers, although a large amount of Input, Weights and Output are involved, actually these data are always used repeatedly. Some data even can

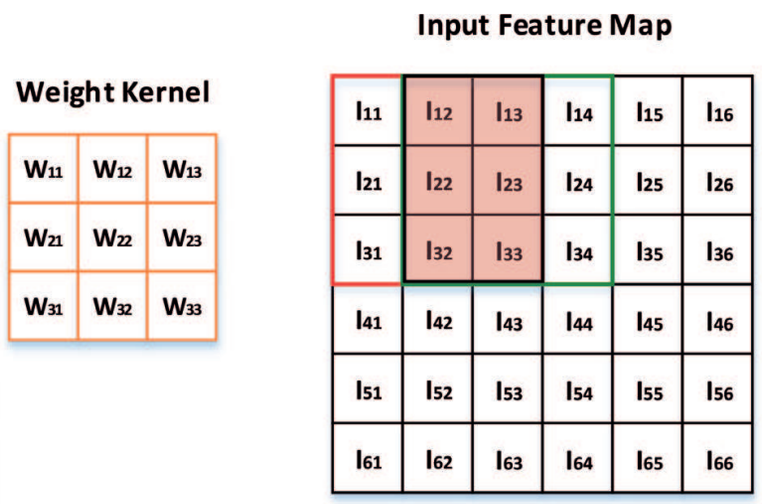

Figure 11. Data-reusing in the process of convolution.

	# of data(K)	# of total data access(K)	Average # of using times per data
Input	150	86,704	576
Weights	1.728	86,704	50,176
Output	3211	86,704	27

Table 1. Statistics of CONV1 of VGG-16 model.

be used over thousands of times. Therefore, data-reusing is an available optimized method to reduce the data accesses of off-chip memory.

Data-reusing greatly reduce off-chip memory accesses, and the benefits are obvious. First, it can relieve the pressure of the off-chip memory bandwidth. Second, it can shorten the operation time of accelerator since the time of accessing off-chip memory is more longer than accessing on-chip buffer. Finally, as **Figure 10** shown, since the energy cost of off-chip memory access is much higher than the energy cost of on-chip buffer access, data-reusing will make accelerator more energy-efficiency.

Although data-reusing is an available optimized method to reduce the data accesses of off-chip memory, it needs extra on-chip memory as a price to buffer data. Usually the benefits and the cost brought by data-reusing vary greatly from different data and CONV layers. From **Table 1** we can easily infer that reusing Weights reduce more data accesses of off-chip memory than reusing Input or Output, for the reason that the average amount of using times per Weights is far larger than the Input and Output. In addition, the size of buffer for reusing Weights in CONV1 is larger than CONV13, for the reason that the amount of Weights in CONV1 is far larger than CONV13 as **Figure 7(a)** shown. From the above discussion, although data-reusing is an available method, due to the different benefits and cost caused by reusing different data and reusing in different CONV layers, data-reusing is a complicated optimization method involving many factors. Therefore, it is necessary to explore the strategy of data-reusing.

Actually, some previous works explored the optimization of memory access by reusing data. Work [7] explored the influence of loop tiling and transformation to data-reusing, which has an impact on computing throughput and memory bandwidth. Work [8] explored the influence of multiple design variables to the accelerator design. These works took account of the influence factors to data-reusing in a certain extent, such as the loop tiling and loop interchange; however, they did not take account of the influence of parallel computation to data-reusing. Work [27] proposed a methodology on how to determine the parallelism strategy. However, this work did not take account of the influence of loop tiling and loop interchange.

Following in this chapter, we will discuss the factors impacting on data-reusing in deep, including loops execution order, reusing strategy and parallelism strategy.

4.2.1. Loop execution order

The computational procedure of convolutional layer can be expressed by a convolution loop nest as **Figure 3**. The similar iteration loop dimensions can be merged to a new loop dimension, and the loops execution order can be transformed, as **Figure 12(a)** and **(b)** shown. **Figure 12(b)** is transformed by **Figure 12(a)**. The loops execution order of two loop nests is different but the result is same.

For the reason that the weights will be accessed repeatedly in the loop ***a***, we can buffer the weights which will be accessed repeatedly in loop ***a*** and reuse them. Assuming reusing weights in loop ***a***, we need corresponding on-chip buffer to store the weights. All weight which will be accessed in loop ***a*** should be buffered. For example, in the loops execution order

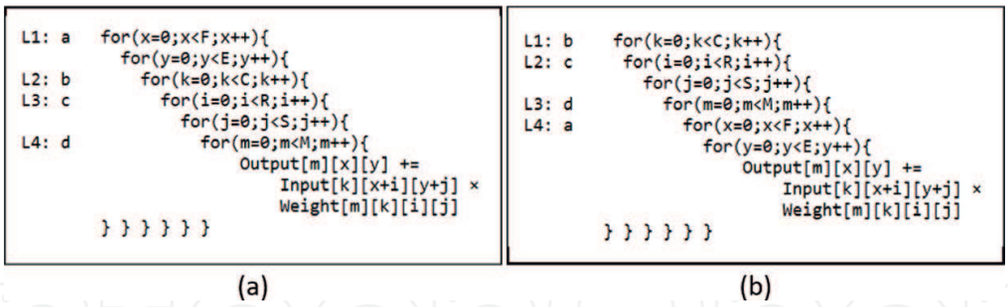

Figure 12. Two different loops execution order: (a) order a; (b) order b.

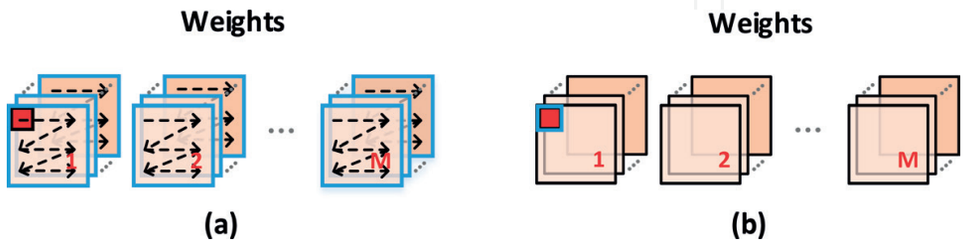

Figure 13. Reusing weight in two situations with different loops execution order: (a) situation a; (b) situation b.

of **Figure 12(a)**, when executing loop ***a***, the weights in the inner loops such as loop ***b***, ***c*** and ***d*** will be accessed. Thus, if we want to reuse the weights in loop ***a***, we need to buffer the weights which accessed not only in loop ***a***, but also in inner loop ***b***, ***c*** and ***d***. Therefore, the on-chip buffer size depends on the amount of the weights which will be accessed when executing loop ***a***. And the amount of the weights accessed depends on the relative position of loop ***a*** in the whole loops nest.

Compare the loop execution order of two cases of **Figure 12(a)** and **(b)**. In **Figure 12(a)**, if we want to reuse the weight which is represented by the red pixel in **Figure 13(a)** in loop ***a***, between two times for accessing the red pixel weight, the other weights as the blue frame shown in **Figure 13(a)** will be accessed. Thus, we need to buffer these weights which are shown as the blue frames in **Figure 13(a)** and the amount of these weight is $C \times R \times S \times M$. Assuming the bit width of weights is 2 bytes, the buffer size for reusing weights is $2 \times C \times R \times S \times M$ bytes.

In **Figure 12(b)**, due to the loop ***a*** is the innermost loop of the loop nest, if we want to reuse the weight which is shown as the red pixel in **Figure 13(b)**, between two times for accessing the red pixel weight, there is no other weights need to access. So that we just need to buffer the weight which is shown as the blue frame in **Figure 13(b)** and the amount of weight is only 1. Thus the buffer size for reusing weights is 2 bytes. By comparing these two cases, we can find that the loops execution order has a critical influence on the on-chip buffer size for reusing data.

Similarly, the Input and Output can be reused either and the size of the buffer for reusing Input and Output can be calculated like above. When consider the influence on the buffer size for reusing data, Input, Weight and Output all should be taken into account, and find out the optimal loops execution order.

http://dx.doi.org/10.5772/intechopen.75796

4.2.2. Reusing strategy

In the above, we introduce the influence of the loop execution order on the buffer size. Actually reuse strategy is also an important influence factor to the data reuse. Reuse strategy is a concrete scheme indicating which data will be reused in which loop dimension. For instance, in **Figure 12(a)**, we can reuse or not reuse the Weight in loop ***a***. Similarly, we can choose in loop ***a*** or loop ***d*** that we reuse Input. Thus, there are a lot of reusing strategies combination we can choose. Given a loops execution order, different reuse strategy will make buffer size and memory access different. When we consider the data-reusing, reusing strategy should be taken into account and find out the corresponding optimal reusing strategy.

4.2.3. Parallelism strategy

Parallel computing is an effective method to improve computational performance of CNN model. Actually parallelism is also a method reusing data indirectly. In some cases, parallel computing involves the same data, thus when parallel computing we just need to access the data once, compared to serial computing, which we need to access data repeatedly. However, parallelism also brings some problems such as higher hardware overhead, higher bandwidth requirement, extra buffer size, etc. Generally, a legal parallelism strategy of CONV layer for **Figure 3** must meet the following constraints:

$$\begin{cases} 0 < P_a \times P_b \times P_c \times P_d \le (\#\text{ of PEs}) \\ 0 < P_a < E \times F \\ 0 < P_b < C \\ 0 < P_c < R \times S \\ 0 < P_d < M \end{cases} \tag{2}$$

In Eq. (2) P_x represent the parallelism degree in loop dimension ***x***. Eq. (2) indicates the number of PEs is the limitation of total parallelism. And the parallelism degree in each loop dimension cannot exceed the loop iterations.

Under the constraints as Eq. (2), there are many combinations of the parallelism strategy. For the reason that the computational performance of CNN accelerator is generally proportional to the total parallelism degree, if the total parallelism degree is similar, the computational performances of taking different parallelism strategies are similar. However, the memory access of taking different parallelism strategies differs greatly. Assuming two cases, the first case is that take a parallel computing in the loop dimension ***b*** and the parallelism degree is Pm. The diagram of the computing process is shown as **Figure 14(a)**. The red frame represents the data involved in a parallel computing. The computing diagram shows that when parallel computing in the loop dimension ***b***, there are Pm Input and Pm Weight, totally 2Pm data are involved in parallel computing per time. Another case is that take a parallel computing in the loop dimension ***d*** and the parallelism degree is also Pm, as **Figure 14(b)**

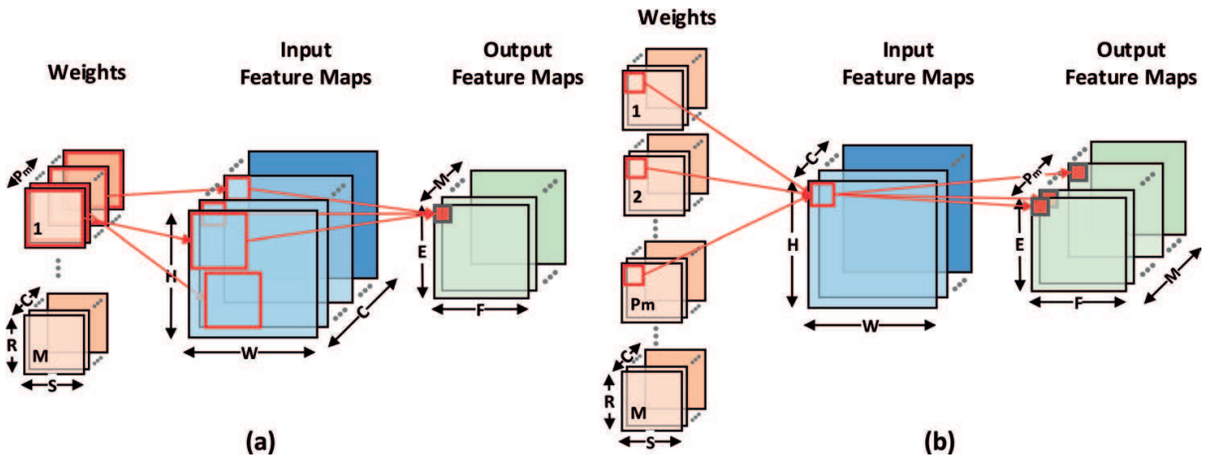

Figure 14. Computing process of two parallelism strategies: (a) parallelism in loop dimension ***b***; (b) parallelism in loop dimension ***d***.

shown. Similarly, we can easily get that there are 1 Input and Pm Weight, totally Pm + 1 data are involved in parallel computing per time. Obviously, the amount of data accessed by the second parallelism strategy is less approximately by half than the first parallel strategy. By comparison between these two cases, we can find that different parallelism strategies have different influence on the amount of data accessed. Therefore, when we design the CNN accelerator and take the parallel computing for the accelerator, a good parallelism strategy can reduce the amount of data accessed, and reduce the amount of memory accesses accordingly.

Usually parallel computing will bring extra on-chip buffer. When calculating the on-chip buffer size, we can regard parallel computing as the combination of a number of (it depends on the parallelism degree) independent computing. In that circumstance, the total buffer size can be calculated by the buffer size of each independent computing. Thus we can calculate the buffer size of each independent computing and get the total buffer size for parallel computing.

4.3. Design space evaluation

In the above, we introduce the influence of data precision and data-reusing to CNN accelerator. In this section we combine all the design factors of accelerator and make a comprehensive analysis. Here we use the buffer size and off-chip memory access as the evaluation parameters of CNN accelerator. Enumerate all design possibilities which are the combinations of different loops execution order, reusing strategy and parallelism strategy. And calculate the buffer size and the off-chip memory access of all design possibilities with the analysis in the previous section and generate a series of groups consist of design space, buffer size and off-chip memory access, and depict all these groups on a graph.

Based on the CONV1 of the VGG-16 model and using 16-bit fixed point data precision, under the constraint that the number of PEs less than 500 which means the total parallelism degree less than 500, we enumerate all design possibilities and obtain a series groups of all design

space. To simplified analysis, we remove the overlapping and obviously poor design groups and remain a series of the optimal design groups. **Figure 15** depicts these optimal design groups and it shows the relationship between the on-chip buffer size and off-chip memory access of all design groups. The "x" axis denotes the buffer size of CNN accelerator and the "y" axis denotes the off-chip memory access. Each point in the graph represents a kind of design space. The blue curve depicts the distribution trend of the optimal design space. As the blue curve shown, the optimal design space points distribute approximately on a hyperbola. It indicates that when we design a CNN accelerator, the relationship between buffer size of accelerator and off-chip memory access is an inverse relationship as a whole. It means if we want to reduce the off-chip memory access, corresponding on-chip buffer are required for reuse data. Reduce the on-chip buffer size, usually accompanied by the increment of off-chip memory access.

Based on **Figure 15**, we can choose the optimal design space according to the requirement of hardware. For example, if our FPGA platform only provides 1 KB on-chip buffer, we can choose the design space of point A in **Figure 15** to design our accelerator. If we want to reduce the memory bandwidth and the power consumption of accelerator and we do not concern about the buffer size, we can choose the design space of point C. Moreover, if we want to make a balance between on-chip buffer size and off-chip memory access, we can choose the design space of point B. In a word, we can obtain the appropriate and optimal design space of CNN accelerator referring to **Figure 15**.

4.4. Power consumption evaluation

In the previous section, we mentioned that the power consumption between different memory accesses differs greatly. According to **Figure 10**, under the commercial 65-nm process, the power consumption of DRAM access is 33 times more than the global buffer access. Therefore,

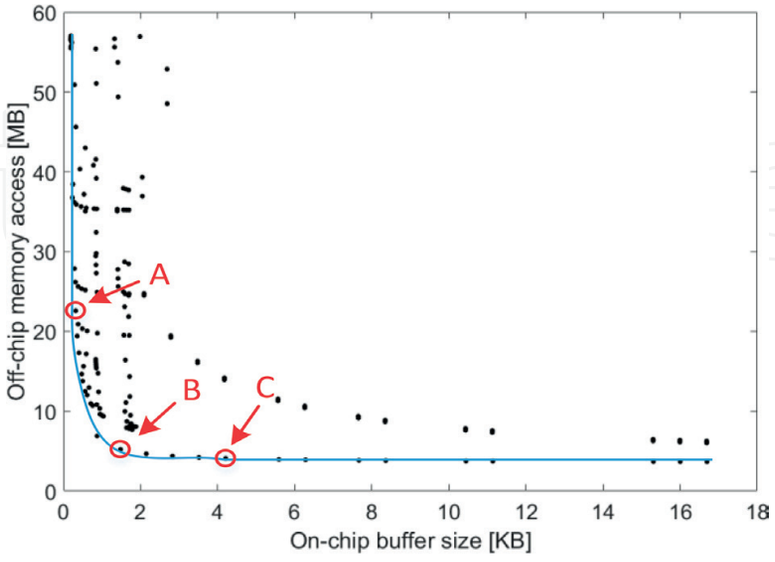

Figure 15. The relationship between the on-chip buffer size and off-chip memory access of all design groups for CONV1 of VGG-16.

we can consider that the power consumption of DRAM access is the main part of power consumption of memory access. Storing and reusing data with on-chip buffer can reduce the DRAM access, so that it can reduce the power consumption of the computing process of CNN accelerator.

As **Figure 15** shown, the off-chip memory access of point B is 4.5 times more than point A, while the on-chip buffer size of point B is only 2 KB approximately more the point A. It means we can increase a small on-chip buffer (2 KB) to obtain the 4.5 times reduction of the power consumption of memory access of CONV1.

Although it is theoretical estimate of power consumption, it shows that we can make a balance between on-chip memory size and power consumption. Our design space evaluation method can help us to choose the optimal design space of CNN accelerator which can reduce the power consumption by increasing small on-chip buffer size. Facing the tendency that miniaturization and low power consumption of IOT, our evaluation method is an effective design strategy and match the concept of green electronics.

5. Summary

Convolution neural network (CNN) has been developing rapidly and used widely for many computer science domains in the past few years, such as image recognition, speech recognition, game play, etc. In the image recognition filed, the recognition accuracy of ResNet exceeded human accuracy in 2015. The outstanding performance makes CNN more and more popular in the artificial intelligence applications. Many researches increase the depth of CNN model to improve the accuracy; in the meanwhile, it brings great pressure to the hardware. Therefore, many specific CNN accelerators are designed and used for CNN computing, including FPGA designs, GPU designs, and ASIC designs, which aim to improve the computing performance and reduce the energy consumption of CNN computing.

In this chapter, we reviewed the history of CNN and introduce the basic and principle of CNN. Following we presented a real-life CNN model, namely VGG-16. We illustrated several CNN accelerators and then we introduced and analyzed two optimization methods of CNN accelerators, including reducing data precision and data-reusing. Based on the analysis, we enumerated all legal design possibilities of CNN accelerator and the optimal design space for CNN accelerator can be obtained. By depicting the design space groups on a graph, we obtained the appropriate design space of CNN accelerator according to our design requirements.

Acknowledgements

This work was supported by the Project Science and Technology of Guangdong Province of China (2015B090912001, 2016B010123005, and 2017B090909005).

Author details

Wenquan Du, Zixin Wang and Dihu Chen*

*Address all correspondence to: stscdh@mail.sysu.edu.cn

Sun Yat-Sen University, Guangzhou, China

References

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

[2] Krizhevsky A, Sutskever I, Hinton GE, Editors. ImageNet classification with deep convolutional neural networks. International Conference on Neural Information Processing Systems; 2012

[3] He K, Zhang X, Ren S, Sun J, editors. Deep Residual Learning for Image Recognition. Computer Vision and Pattern Recognition; 2016

[4] Sainath TN, Mohamed AR, Kingsbury B, Ramabhadran B, editors. Deep convolutional neural networks for LVCSR. In: IEEE International Conference on Acoustics, Speech and Signal Processing; 2013

[5] Hinton G, Deng L, Yu D, Dahl GE, Mohamed AR, Jaitly N, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine. 2012;**29**(6):82-97

[6] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Driessche GVD, et al. Mastering the game of go with deep neural networks and tree search. Nature. 2016;**529**(7587):484

[7] Zhang C, Li P, Sun G, Guan Y, Xiao B, Cong J, editors. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In: Acm/Sigda International Symposium on Field-Programmable Gate Arrays; 2015

[8] Ma Y, Cao Y, Vrudhula S, Seo JS, editors. Optimizing Loop Operation and dataflow in FPGA acceleration of deep convolutional neural networks. In: Acm/Sigda International Symposium on Field-Programmable Gate Arrays; 2017

[9] Chetlur S, Woolley C, Vandermersch P, Cohen J, Tran J, Catanzaro B, et al. cuDNN: Efficient primitives for deep learning. Computer Science. 2014;arXiv:1410.0759

[10] Chen YH, Krishna T, Emer JS, Sze V. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits. 2016;**PP**(99):1-12

[11] Luo T, Liu S, Li L, Wang Y, Zhang S, Chen T, et al. DaDianNao: A neural network supercomputer. IEEE Transactions on Computers. 2017;**66**(1):73-88

[12] Holi JL, Hwang JN. Finite precision error analysis of neural network hardware implementations. Computers IEEE Transactions on. 1993;**42**(3):281-290

[13] Draghici S. On the capabilities of neural networks using limited precision weights. Neural Networks. 2002;**15**(3):395

[14] Chen T, Du Z, Sun N, Wang J, Wu C, Chen Y, et al. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. Acm Sigplan Notices. 2014; **49**(4):269-284

[15] Luo T, Luo T, Liu S, He L, He L, Wang J, et al., editors. DaDianNao: A machine-learning supercomputer. In: Ieee/Acm International Symposium on Microarchitecture. 2014

[16] Suda N, Chandra V, Dasika G, Mohanty A, Ma Y, Vrudhula S, et al. Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks. In: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Monterey, California, USA. 21 – 23 February, 2016. pp. 16-25

[17] Hu J, Shen L, Sun G. Squeeze-and-excitation networks. Computer Science. 2017;arXiv: 1709.01507

[18] Desoli G, Chawla N, Boesch T, Singh SP, Guidetti E, Ambroggi FD, et al., editors. 14.1 A 2.9TOPS/W deep convolutional neural network SoC in FD-SOI 28nm for intelligent embedded systems. In: Solid-State Circuits Conference; 2017

[19] Nair V, Hinton GE, editors. Rectified linear units improve restricted boltzmann machines. In: International Conference on International Conference on Machine Learning; 2010

[20] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. Computer Science. 2014;arXiv:1409.1556

[21] Deep CNN and Weak Supervision Learning for Visual Recognition [Internet]. 2016. Available from: https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/

[22] Chen YH, Emer J, Sze V. Using dataflow to optimize energy efficiency of deep neural network accelerators. IEEE Micro. 2017;**37**(3):12-21

[23] Alwani M, Chen H, Ferdman M, Milder P, editors. Fused-layer CNN accelerators. In: Ieee/acm International Symposium on Microarchitecture; 2016

[24] Sze V, Chen YH, Yang TJ, Emer JS. Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE. 2017;**105**(12):2295-2329

[25] Gupta S, Agrawal A, Gopalakrishnan K, Narayanan P. Deep learning with limited numerical precision. Computer Science. 2015;arXiv:1502.02551

[26] Qiu J, Wang J, Yao S, Guo K, Li B, Zhou E, et al., editors. Going deeper with embedded FPGA platform for convolutional neural network. In: Acm/Sigda International Symposium on Field-Programmable Gate Arrays; 2016

[27] Li H, Fan X, Jiao L, Cao W, Zhou X, Wang L, editors. A high performance FPGA-based accelerator for large-scale convolutional neural networks. In: International Conference on Field Programmable Logic and Applications; 2016