A FPGA-based end-to-end acceleration framework for fast deployment of Convolutional Neural Networks

Lin Zhang\(^{1,*}\), Bing Li\(^{1}\) and Binfeng Liu\(^{1}\)

\(^{1}\)School of Electronic Science & Engineering, Southeast University, Nanjing, Jiangsu, China

*E-mail: edison2301@qq.com

Abstract. Nowadays, CNNs has delivered the state-of-the-art performance in the field of computer vision, image classification, etc. As CNNs going deeper, it becomes more difficult to implement CNNs applications based on general-purpose computing platforms. Recently, many FPGA-based CNNs accelerators have been proposed, these accelerators achieved high performance on specific CNNs models, however they are somewhat lack of reconfigurability to fit different applications. To deal with this problem, an end-to-end acceleration framework was proposed in this paper, which consists of a parameterized hardware accelerator and a fully automatic software framework. Parallel computation and pipeline optimization are deployed in the hardware design to achieve high performance. Simultaneously, runtime reconfigurability is implemented by using a global register list. By encapsulating the underlying driver, a three-layer software framework is provided for users to deploy their pre-trained models. A typical CNNs model used for handwritten digital recognition was selected to test and verify the accelerator. The experimental result shows that the accelerator can reach a recognition speed of 22.65FPS under the clock frequency of 100MHz, comparing with ARM Cortex-A9 working at 650MHz, it can achieve 25.9 times of acceleration effect, with only 1.59W power consumption.

1. Introduction

By reconstructing optic nerve's behavior, Convolutional Neural Networks(CNNs) have achieved great success in object detection\(^{1}\), image classification\(^{2}\)[3], video surveillance\(^{4}\), etc. As CNNs models going deeper\(^{5}\)[6], their computation-intensive and storage-intensive features become increasingly prominent, which makes it difficult to deploy CNNs applications on general-purpose computing platforms. Central Processing Unit(CPU) is designed for general-purpose computing and its sequential execution feature prevents it from implementing complex CNNs algorithms. Graphics Processing Unit(GPU) achieved good results in accelerating training and inference of CNNs\(^{3}\)[5][7]. However, the power consumption of GPU is too high. Field Programmable Gate Array(FPGA) seems to be a good choice for accelerating CNNs applications due to its excellent parallel computing ability, runtime reconfigurability, short development round and high power efficiency. Recently, a great deal of FPGA-based CNNs accelerators have been proposed\(^{8}\)[9][10][11].

CNNs acceleration mainly focuses on following aspects. Chen Zhang proposed a Roof-line model to fully explore the design space and finally achieved a balance between computation throughput and communication bandwidth\(^{12}\). By using a 11-bit fixed-point data format for MAC calculation and caching all weight parameters and intermediate results in on-chip buffers, Zhou Yongmei implemented a complete network of Lenet-5 on FPGA using Vivado HLS\(^{13}\). Y Huang proposed to
accelerate CNNs on FPGA using the Winograd algorithm, which reduces the number of multiplications but increases the number of additions[14]. The above solutions paid more attention to the improvement of computing performance or communication bandwidth, while neglects the runtime reconfigurability for different applications. In order to rapidly deploy CNNs applications on FPGAs, this paper proposes an end-to-end framework for CNNs acceleration which shows high computing performance and runtime reconfigurability. The main contributions of our work are as follows.

(1) A hardware accelerator for CNNs inference was proposed, which is parameterized and highly reconfigurable. Parallel computation and pipeline optimization have been deployed to achieve high performance. Simultaneously, runtime reconfigurability is implemented to fit varies applications.

(2) An automatic software framework was designed to rapidly mapping the pre-trained CNNs models into FPGA hardware accelerator.

(3) A model for handwritten digit recognition was selected to validate the framework on Pynq-Z1, which achieved a recognition speed of 22.65FPS on MNIST dataset, the average power is only 1.59W.

2. Background
As shown in figure 1, a typical CNNs model consists of three types of layers, including convolutional layers, pooling layers, and full-connected layers. All layers are connected to each other to form a whole neuron network. Figure 2(a) shows the structure of convolutional layer. Convolution layers are usually used for feature extraction by performing 3-dimensional multiply and accumulate operations between the convolution kernel and the input feature maps. Pooling layers are usually located behind convolutional layers to perform down-sampling of the extracted features, which improves the tolerance of CNNs to the distortion of input images. As shown in figure 2(b), the pooling layers first divides the input feature map into multiple non overlapping rectangular regions, then maximum or an average operation is performed on the pixels in each sub-region. Pooling layer is used to effectively reduces the size of a CNNs model. Full connected layers using a 2-dimensional matrix multiplication to perform full connect mapping from $D_i$ input neurons to $D_o$ output neurons.

![Figure 1. Structure of CNNs.](image)

(a) Convolutional layer structure  (b) Pooling layer structure

![Figure 2. Structure of different layers.](image)

3. Design methods
3.1. Implementation overview
Figure 3 shows an overview of the end to end acceleration framework in this paper. The system can be divided into two parts: Programmable Logic(PL) and Processing System(PS). The PL side consists of a parameterized hardware accelerator designed with HLS and a Direct Memory Access(DMA) controller responsible for transferring data between external memory and on chip buffers. The hardware accelerator consists of Convolutional Engine, Pooling Engine, Full Connect Engine and Control Logic, it is named as Compute Engine. The PS side is mainly composed of an ARM cortex-A9, running an Ubuntu operating system. A shared SDRAM is used to data communication between
PS and PL. Besides, a three-layer software framework is designed to interact with the underlying hardware logic and also provide a convenient user interface for fast deployment of CNNs applications.

3.2. Compute engine
The computation process of convolutional layers has been described in section 2, there are four types of parallel computing that can be utilized to accelerate the operation, which are parallelism of MAC operations inside the filter, parallelism across all input channels, parallelism inside the output feature map and parallelism across all output channels. In order to reduce the size of on-chip memory, we choose to unroll the loops between the input channel dimension and output channel dimension in the computation process. Figure 4(a) shows the structure of the convolution acceleration module.

![Figure 4. Acceleration module of different layers.](image)

Convolution Engine consists of four parts: Multiply Add Accumulator(MAC) array, Arithmetic Logic Unit(ALU) array, activation array and output ping-pong, which are fully pipe-lined between each other. The MAC array comprises 8 MAC units, each MAC is composed of 10 multipliers and 10 adders. By performing full parallelism of each MAC unit, MAC array can perform up to 80 MAC operations per clock cycle. ALU array contains 8 accumulators, which is responsible for accumulating the output of MAC unit. The iteration number of the accumulating units is generated by Control Logic and depends on the filter size K together with the number of input channels D. If \( D \leq 10 \), all the input channels can be fully expanded on a MAC unit, so the iteration number should be \( K \times K \). If \( D > 10 \) and \( D/10 = P \) is an integer, MAC operations of all input channels should be calculated in batches, then the iteration number should be \( K \times K \times P \). If \( D \leq 10 \) and \( D/10 = P \) is not an integer, then the iteration number should be \( K \times K \times \lceil P \rceil \), where \( \lceil P \rceil \) represents the smallest integer not less than \( P \). Activation array is a hardware implementation of the RELU function. It consists of 8 RELU activation units, each of which is simply implemented by a comparator. As some convolutional layers do not use an activation function, the activation unit may be skipped under the influence of Control Logic in that case.

Pooling Engine has the same computational structure as the convolutional engine, we choose to unroll the loop between the input channel dimension to accelerate the pooling computation. As a result, the structure of the pooling engine is similarly to convolutional engine, which is shown in figure 4(b).

The computation of full-connected layers includes two kinds of parallelism, which are parallelism of input neurons and output neurons. The computation of full-connected layers involves a large number of MAC operations. The loop between the input and output dimension is unrolled to accelerate
the computation. The structure of Full Connect Engine is the same as that of Convolution Engine. The difference is that the number of iterations for accumulating unit depends only on the number of input neurons $D$. If $D \leq 10$, all the input neurons can be fully expanded on the 10 MAC units, so the iteration number equals to 1. If $D > 10$ and $D/10 = P$ is an integer, MAC operations across all input neurons should be calculated in batches, and the iteration number should be $P$. If $D > 10$ and $D/10 = P$ is not an integer, the iteration number should be $\lceil P \rceil$, where $\lceil P \rceil$ represents the smallest integer not less than $P$.

### 3.3. On-chip memory system

Parameters and pixels need to be acquired by the accelerator at the same time when it is running at full speed. According to the system structure of the accelerator, the accelerator has only one input data port, which allows a maximum bandwidth of one word per cycle. It is hard to meet the data requirements of the acceleration modules by purely using external memory interface to feed the compute engine with data. To solve this problem, an on-chip memory system is designed to transfer all the parameters and pixels into the accelerator through AXI4 stream bus in burst mode, which shows in figure 5.

![Figure 5. On-chip memory system.](image)

### 3.4. Runtime reconfigurability

Some configuration parameters must be transferred into a global register list to realize runtime reconfigurability. Transmission of parameters usually has two options: dedicated control bus or data advanced-agreed format via data bus. In this paper, the latter method is selected because no additional hardware resources are required and it’s easier to control. The data format of the input stream is shown as figure 6. The accelerator buffers the first 16 words in burst mode. The first word is regarded as an engine-select parameter, which indicates the type of the current computation. Then corresponding acceleration engine will be started by the control logic. Then the started acceleration engine continues to decode the rest parameters and generate control signals for the whole accelerator.

![Figure 6. Format of input data stream.](image)

### 3.5. Software framework

For fast deployment of CNN models on our accelerating platform, a 3-layer software framework is designed based on PYNQ open source framework. The CNNs mapping flow based on our framework is shown in figure 7. Driver layer is used to manage the hardware accelerator and programme bitstream file into FPGA. API layer provides a friendly interface by encapsulating the classes of the driver layer, users can call API to quickly deploy a CNN model into FPGA. The application layer contains a parser for extracting the structure parameters and weight parameters of the given Caffe model, and a model constructor which call the API to build a FPGA-based CNN model automatically.

![Figure 7. Software framework and mapping flow.](image)
4. Experiment and results

In reality, the structure and scale of CNN models vary according to different application scenarios. The benchmark CNNs model we choose to evaluate our design is a typical model for MNIST handwritten digit recognition. Figure 8 shows the structure of the network, the network is input with a 28x28 grayscale image and contains two convolutional layers, two pooling layers and two full connect layers. The output of two pooling layer is activated by RELU function. A Caffe model is pre-trained on host using MNIST handwritten digital data set to validate the FPGA-based end-to-end acceleration framework above. The accelerator is finally validated on Pynq-Z1 evaluation board. This board features a Xilinx XC7Z020-clg400 chip that integrates an ARM cortex-A9 dual-core CPU working at a clock frequency of 650MHz with an Artix-7 FPGA working at 100MHz. An ubuntu operating system is running on the ARM CPU which can be accessed via the ethernet interface.

![Figure 8. Configuration a typical CNN model.](image)

Table 1 shows the resource utilization. Since our design fully explores the parallelism of CNNs and 32-bit float point data format is used for calculation, 195 DSP48Es are used. And up to 156 BRAMs are used due to the ARRAY_PARTION optimization applied to the on-chip memory system. However, it can be seen from table 1 that the on-chip resources of the FPGA are not fully utilized, that provides further space of acceleration for future works. According to the power analysis reported by Vivado IDE, the average power consumption of the accelerator is 1.59W, in which the ARM CPU occupies 88% of the total dynamic power consumption due to its higher working frequency. And the dynamic power consumption of FPGA side is only 0.179W, which is relatively low.

| Resource          | BRAM 18K | DSP48E | FF   | LUT   |
|-------------------|----------|--------|------|-------|
| Total             | 156      | 195    | 40945| 31201 |
| Available         | 280      | 220    | 106400| 53200 |
| Utilization(%)    | 56       | 89     | 38   | 59    |

To evaluate the performance of the accelerator, we compared our work with reference[15] and pure software implementation based on ARM CPU. Table 2 shows the comparison results of different works. For the same MNIST handwritten digit recognition task, our accelerator achieves 25.9x speed up compared to ARM CPU, and 1.15x speed up compared to the design in reference [15]. Our work suffers an accuracy loss of 0.54% compared to reference[15], mainly because the model structure used for testing is slightly different with ours. Due to the use of 32-bit float point data for calculation in our design, the accelerator achieves the same accuracy with ARM CPU-based implementation.

| Comparison      | CPU          | [15]'s Work | Our Work |
|-----------------|--------------|-------------|----------|
| Platform        | ARM Cortex-A9| XC7Z020     | XC7Z020  |
| Data Type       | float32      | float10     | float32  |
| Performance     | 0.82         | 19.7        | 22.65    |
| Recognition     | 98.08%       | 98.62%      | 98.08%   |

5. Conclusion

In this paper, a FPGA-based end-to-end CNN acceleration framework is proposed for fast deploying CNN models. By fully exploring the parallelism of CNNs' layers, parallel computation and pipeline optimization were used to achieve high performance. Global register list is designed to realize runtime reconfigurability. A framework based on PYNQ was designed to extract pre-trained CNNs models.
into FPGA to realize different applications. Bseides, MNIST handwritten digital recognition was selected to validate our work. Experimental result shows that our accelerator can reach a recognition speed of 22.65FPS under the clock frequency of 100MHz, 25.9 times faster compared with ARM cortex-A9 CPU working at 650MHz, and the average power is only 1.59W.

In summary, our acceleration framework is runtime reconfigurability, high performance and low power, which can be used to quickly deploy lightweight CNNs applications in embedded systems. In the future, a FPGA evaluation board with more dsp48es can be used, so as to make more effective use of other resources inside of FPGA, which is believed to obtain better performance.

References
[1] Ren S, He K, Girshick R, et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks[J]. 2015.
[2] Lecun Y L, Bottou L, Bengio Y, et al. Gradient-Based Learning Applied to Document Recognition[J]. Proceedings of the IEEE, 1998, 86(1):2278-324.
[3] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks[C]// International Conference on Neural Information Processing Systems. 2012.
[4] Ji S, Xu W, Yang M, et al. 3D Convolutional Neural Networks for Human Action Recognition[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2013, 35(1):221-31.
[5] Szegedy C, Liu W, Jia Y, et al. Going Deeper with Convolutions[J]. 2014.
[6] He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition[J], 2015
[7] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. Computer Science, 2014.
[8] Yu Zijian, Ma De, Yan Xiaolang, et al. FPGA-based Accelerator for Convolutional Neural Network [J]. Computer Engineering, 2017, 43(1): 109-14,119.
[9] Ma Y, Cao Y, Vrudhula S, et al. Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2018:1-14
[10] Li H, Fan X, Jiao L, et al. A high performance FPGA-based accelerator for large-scale convolutional neural networks[C]// 2016 26th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2016
[11] Ma Y, Cao Y, Vrudhula S, et al. Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks[C]// the 2017 ACM/SIGDA International Symposium. ACM. 2017.
[12] Zhang C, Li P, Sun G, et al. Optimizing fpgabased accelerator design for deep convolutional neural networks[C]//Acmsigda International Symposium on Field-programmable Gate Arrays. ACM, 2015
[13] Zhou Y, Jiang J. An FPGA-based accelerator implementation for deep convolutional neural networks[C]// International Conference on Computer Science & Network Technology. IEEE 2016
[14] Huang Y, Shen J, Wang Z, et al. A High-efficiency FPGA-based Accelerator for Convolutional Neural Networks using Winograd Algorithm[J]. Journal of Physics: Conference Series, 2018, 1026:012019.
[15] Ghaffari S, Sharifian S. FPGA-based convolutional neural network accelerator design using high level synthesizer[C]// 2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS). IEEE, 2016