A Method for Accelerating Convolutional Neural Networks Based on FPGA

Mengxing Zhao[1], Xiang Li[2], Shunyi Zhu[3]
School of Microelectronics
ShanDong University
JiNan, China
e-mail: {zmx, Xiang_Li}@mail.sdu.edu.cn
e-mail: zsy@sdlearningtech.com

Li Zhou*
School of Microelectronics
ShanDong University
JiNan, China
e-mail: zhou_li@sdu.edu.cn

***Abstract*—Convolutional neural networks (CNNs) have achieved notable success in the field of computer vision, ranging from image classification to object detection. However, characteristics of intensive multiplication computing, large-scale parameters and resource consuming enormously restrict its application in embedded devices. In this paper, we propose a method that accelerates a target CNN module based on FPGA, and the network we used to demonstrate is LeNet-5. Firstly, the LeNet-5 module will be pruned using Dynamic Network Surgery. Secondly, we quantize the parameters to 8-bits in the pruned network by the quantization scheme called Incremental Network Quantization, and recode it to 5-bits for shift operation. Finally, we develop a model-friendly parallel hardware accelerator with 8-bits fixed-point precision using Verilog HDL codes, and integrate it to a SOC system. By converting multiplication operations to shift operations, we design a parallel LUT-based hardware accelerator with no DSP resources utilized. Processed by Xilinx ZC702 FPGA at 100MHz clock, this implementation reaches a peak performance of 33.6 GMACS with only 1.758 W of total on-chip power, and obtains 98.9% accuracy target 10000 images of MNIST test set.**

Keywords-FPGA; Convolutional Neural Networks; hardware accelerator; shift operation; MNIST dataset

I. INTRODUCTION

In recent years, CNNs have gained remarkable achievements in many machine learning applications [1], including image classification and object detection etc. Especially for image classification tasks, CNNs have demonstrated its excellent performance and high accuracy which outperform other machine learning algorithms.

However, CNNs have the characteristics of dense computation and large number of parameters, requiring tremendous floating-point multiplication and accumulation (MAC) operations and parameters storage during the feedforward computing stage for an un-preprocessed model, which brings great challenges to the application for embedded devices. General purpose processor like CPU process tasks based on specific instruction set, that leads to its superior efficiency in processing programs. But when MAC operations processed by CPU, computing pattern based on an instruction set becomes a bottleneck and cannot implement parallel computing sufficiently. Dedicated GPU is specially designed for graphics processing and is ideally suitable for MAC operations, but its higher power consumption makes it almost impossible to apply in embedded devices. ASIC devoted to processes CNNs can perform enormous MAC operations per second with extremely low power consumption [2], but it has the characteristics of low flexibility and long development cycle compared to Field Programmable Gate Array (FPGA). FPGA has a large number of logical units which can be developed for parallel MAC operations with relatively low power consumption, and has high flexibility and short development cycle due to its reconfigurable. Therefore, we choose FPGA as our premier platform to design a CNN hardware accelerator.

The key to developing a CNN hardware accelerator is to get a hardware-friendly CNN model which is suitable for implementation on FPGA, and developing a parallel computing architecture based on it, while minimizing data movement as far as possible. Han et al. introduce "deep compression" to reduce the model size by 39x on LeNet-5 without affecting accuracy by pruning, vector quantization and Huffman coding [3]. However, the parameters of compressed module are still using floating-point precision which is not friendly for implementation on FPGA. Yongmei et al. propose a CNN accelerator for MNIST digit recognition with 11-bits fixed-point precision using Vivado HLS tool, and a peak performance of 16.58 GMACS has been reached [4]. However, no pruning procedure has been applied in this research, and the tool of Vivado HLS is unable to precisely interfere with the design of the underlying circuit compared to Verilog HDL, resulting in a larger consumption of resources. Mankit Sit et al. develop a CNN accelerator using Incremental Network Quantization (INQ) to quantize parameters of the target model, and design "pre-computation units" in their research to eliminate most multipliers [5]. But there still remaining floating-point operations processed by DSP resources, so that's not a fixed-point computing architecture strictly speaking.

In this paper, we propose a new method to accelerate a target CNN module base on FPGA, and the target module we choose is LeNet-5 which is widely used for handwritten digit recognition [6]. Although this model is relatively simple, it contains almost all the essential operations of state-of-art CNN models. Firstly, Dynamic Network Surgery (DNS) technology will be applied on the LeNet-5 module, resulting in a significant reduction in the number of parameters without any loss of accuracy. Secondly, we quantize the remained parameters using INQ technology,

978-1-7281-6297-3/19/$31.00 ©2019 IEEE
DOI 10.1109/ICCIS49662.2019.00049

and then recode the quantized parameters for shift operation which replaces expensive multiplication operations. Above two steps are accomplished in Caffe environment [7]. Finally, we utilize Verilog HDL to develop a model-friendly hardware accelerator based on the processed LeNet-5 module, and then integrate it to a SOC system.

The highlights of our design are as follows:

- The target CNN module is quantized to 5-bits representation and all feature maps in it are represented by 8-bits.
- The expensive multiplication operations can be converted into relatively cheaper shifting operations utilizing "weights recoding" and the proposed "shift multiplier", thereby eliminating the requirement for DSP resources.
- Results in pooling layer and activation layer can be generated during MAC process utilizing pipeline technology, which can bring about better frequency performance.
- Different tasks are rationally assigned to the CPU and the accelerator by exploiting software-hardware co-design.

The rest of the paper is organized as follows: In section II, a brief introduction to CNNs, DNS and INQ are given, and the advantages of applying DNS and INQ to LeNet-5 module are also discussed. In section III, we describe the implementation details of our design. Section IV shows the performance and experimental results of our implementation and Section V makes a conclusion.

II. BACKGROUND

A. Convolutional Neural Networks

A CNN model is composed of several typical essential layers, including convolutional layers, pooling layers, fully connected layers and activation layers. When an input image is given to a CNN model, the feature maps will be generated layer-by-layer base on it until the final feature vector generated by the last layer. Finally, a classifier, e.g., SoftMax is utilized to generate the classification result based on the final feature vector. The computation details of these essential layers are described as follows:

1) Convolution layer

When a convolution layer accepts N input feature maps with the size of H×W presented by N channels and generates M feature maps, it needs N×M filters with the size of F×F. One value of output feature maps is generated as follows if ignore the bias:

$$O[m][x][y] = \sum_{n=0}^{N}\sum_{h=0}^{F}\sum_{w=0}^{F} W[m][n][h][w] * I[n][S * x + h][S * y + w]$$

O, W and I represents multidimensional arrays of output feature maps, filters and input feature maps respectively, and $0 \le m \le M$.

2) Pooling layer

A pooling layer usually follows a convolution layer, sub-sampling feature maps using filters with functions, e.g. average or maximum. Output feature maps of pooling layer will have lower dimensions than input feature maps by sub-sampling functions. In LeNet-5 model, max-pooling is employed.

3) Fully-Connected layer

The input feature map and the output feature map of a FC layer is both 1-Dimension vectors. When an input feature vector with N elements is given to a FC layer to produce an output feature vector with M elements, it needs an array of weights with N×M elements. One value of output feature vector is generated as follows if ignore the bias:

$$O[m](0 \le m \le M) = \sum_{n=0}^{N} W[m][n] * I[n]$$

4) Activation layer

An activation layer processes each element of the input feature vector with nonlinear function. In LeNet-5 model, ReLU function is adopted, and described as follows when given an input feature vector with N elements:

$$O[n](0 \le n \le N) = \begin{cases} I[n], & I[n] > 0 \\ 0, & I[n] \le 0 \end{cases}$$

B. Dynamic Network Surgery

Dynamic Network Surgery (DNS) technology can compress a CNN model to reduce the model complexity significantly by making on-the-fly connection pruning [8]. There are a lot of redundant connections (i.e. parameters) in an original CNN model, which can be pruned with only a slight loss of accuracy. However, some connections make important contributions to the accuracy and should not be pruned. DNS technology can prune unimportant redundant connections precisely through two key operations: pruning and splicing. Pruning operation is utilized to compress model, and splicing operation is utilized recover important connections. Performing above two operations (as illustrated in Figure 1) circularly can maintain compression rate while preventing irreversible network damage.

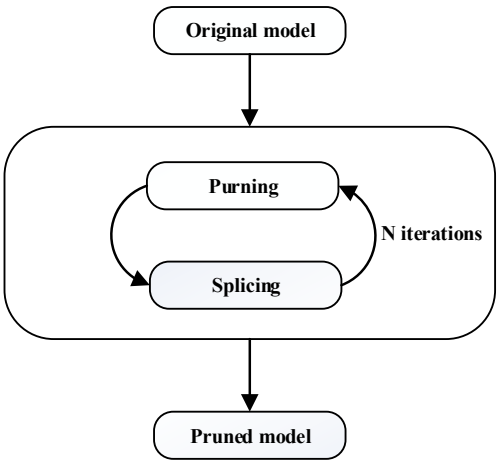

Figure 1. The process of pruning an original CNN model utilizing DNS technology by N iterations.

The number of parameters in the LeNet-5 model is compressed 108× through DNS technology while maintaining a prediction accuracy of 99.09%. The model structure transformation of the original LeNet-5 module after being pruned utilizing DNS technology can be found in Figure 2.

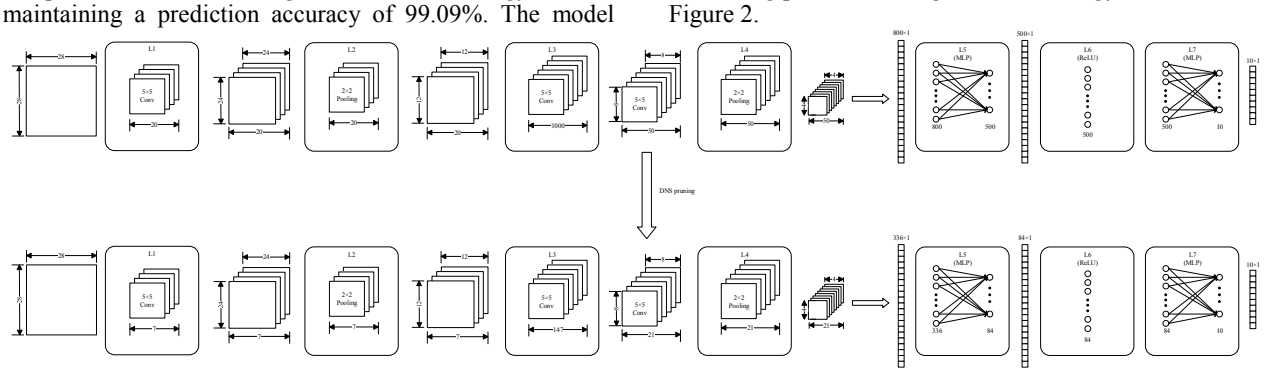

Figure 2. Model structure transformation of the original LeNet-5 module after being pruned utilizing DNS technology.

C. *Incremental Network Quantization*

Incremental Network Quantization (INQ) is a lossless quantization technology, which can convert a full-precision floating-point CNN model into a low-precision fixed-point precision model and quantize weights to be integer power-of-two, or zero [9]. It is composed of three sequential operations: weights partition, group-wise quantization and re-training. Weights partition divides the weights in each layer into two disjoint groups. Group-wise quantization is to quantize the first group and remain the second group unchanged. Finally, keeping the quantized weights fixed while re-training the remaining second group to compensate the model accuracy loss due to quantization of the first group. Perform the above three operations on the full-precision group iteratively until the final full-precision group has been quantized. Model accuracy loss due to quantization can be reduced as far as possible through the re-training process compared to simultaneous quantization. The absolute value of all weights in the LeNet-5 module can be quantized to $2^n(-7 \leq n \leq 0)$ utilizing INQ technology.

TABLE I. RECODED WEIGHTS

0	00000		
2^{-7}	00001	-2^{-7}	10001
2^{-6}	00010	-2^{-6}	10010
2^{-5}	00011	-2^{-5}	10011
2^{-4}	00100	-2^{-4}	10100
2^{-3}	00101	-2^{-3}	10101
2^{-2}	00110	-2^{-2}	10110
2^{-1}	00111	-2^{-1}	10111
1	01000	-1	11000

III. IMPLEMENTATION DETAILS

A. *Data Process*

In MNIST dataset, all images are grayscale and the values range from 0 to 255. For an input image, we divide all the pixel values by 64 and round off them to the nearest whole number as the input image fed to the accelerator, that result in the computing results of all layers are almost equal proportion reduction compared to feed an un-preprocessed image and has no effect on the shape of handwritten digit. It makes it possible to use signed 8-bits integer to represent the values of all the feature maps and almost has no effect on utilizing SoftMax function to generate a classification result. Quantized weights can be recoded to 5-bits representation as shown in Table I. 1 bit represents the sign bit and 4 bits represent the absolute value for shifting operation. As the module structure described in Figure 2, all convolution kernels with the size of 5×5. Therefore, in the proposed convolution-pooling module in section3.4.2, an input feature vector consisting of 25 pixels can be fed simultaneously per cycle to achieve the purpose of parallel computing. On account of a pooling layer follows a convolution layer and performs max-pooling function on four spatially adjacent convolution results utilizing a 2×2 filter with the stride of 2, the input feature vectors can be organized as shown in Figure 3, cooperating with the convolution-pooling module that can generate a max-pooling result after every four convolution operations.

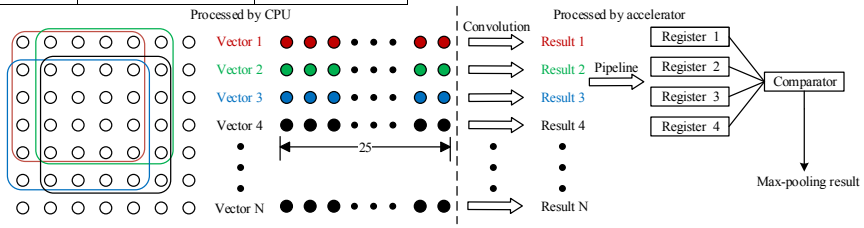

Figure 3. The input feature vectors organization form of the convolution-pooling module when pooling operation is needed, one input feature vector is the pixels framed by the box with the same color to itself. If the bandwidth is ideal, each max-pooling result can be generated every four cycles consumed by four convolution operations owing to pipeline technology are applied.

B. System on Chip Architecture

As shown in Figure 4, three components constitute the proposed SOC architecture, an ARM Cortex A9 CPU, a DDR3 external memory and a LUT-based accelerator, respectively. The CPU is responsible for reading the feature maps (including the input image), converting the feature maps to the input feature vectors required by the accelerator and writing it to the DDR3 memory, interacting the signals of computation process state indication and control computation state with the accelerator through AXI4-Lite bus. The DDR3 memory is responsible for interacting data with the CPU and the accelerator as a data cache, storing the feature maps generated by the accelerator and the input feature vectors converted by the CPU, that requires attention to prevent read-write conflicts. The accelerator read the input feature vectors from the DDR 3memory and write the calculated feature maps to the DDR3 memory through AXI4 bus, generating the computation process state indication signal read by the CPU and receiving the control computation state signal from the CPU through AXI4-Lite bus.

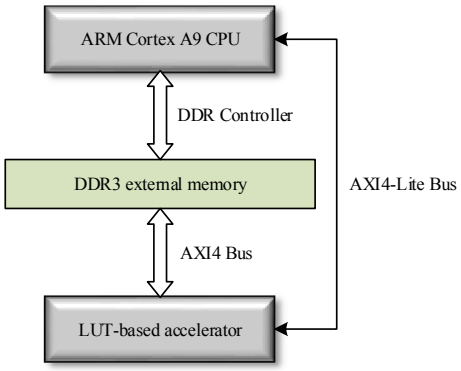

Figure 4. Overall view of the SOC architecture.

C. Processing System

The processing system is responsible for controlling the computation process and performing data processing tasks as described in the previous sections, including an ARM Cortex A9 CPU, a DDR3 external memory, a UART interface and so on.

AXI4-Lite bus is utilized for transmitting the signals of control and state in our design, where the CPU as a master device and the accelerator as a slave device. The state indication signals are generated by the accelerator and stored in its state register. The CPU reads the state register data with a polling mechanism, and performs different tasks according to different values in the state register.

All operations in the processing system are implemented by a C program code stored in an address space of the DDR memory, and run by an ARM Cortex A9 CPU. The unified 4GB address mapping mechanism of the ZYNQ processing system is utilized to access different devices, and the UART interface is utilized as a communication serial port to print debug information.

D. Accelerator Design

We implement pipeline technology to all of the following modules using Verilog HDL to achieve better frequency performance and higher throughput, and describe this accelerator as follows by bottom-up analysis.

1) Shift multiplier

The LUT-based shift multiplier (as shown in Figure 5) performs the function of signed multiplication operation, including two comparators, an OR gate, a shift calculator (including a XOR gate) and a data selector, and its operands is an 8-bits pixel and a 5-bits weight, generating a 16-bits intermediate result which is composed of 1 sign bit, 1 redundant bit and 14 absolute value bits. The redundant bit is designed to prevent overflow in an extreme case where successive positive or negative accumulation. The operands are first fed into the two comparators comparing with 0, generating two 1-bit comparison results ("1" if equal to 0) fed into the OR gate, and then generate the enable signal to the shift calculator. No calculation will be performed in the shift calculator and the intermediate result is assigned to 0 directly if one of the comparison results is "1". Otherwise, the sign bit of the intermediate result is generated by XOR operation on the sign bits of the two operands, and the 14 absolute value bits are generated by shifting operation. The above two results are fed into the data selector and selected by the enable signal generated by the OR gate to produce the 16-bits intermediate result.

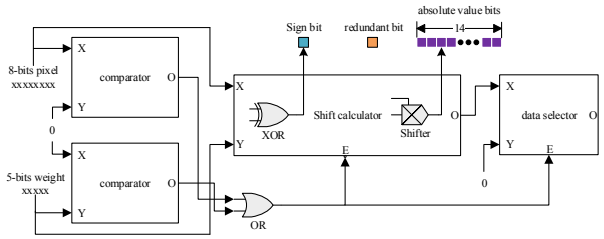

Figure 5. Overall architecture of the shift multiplier.

2) Convolution-Pooling module

Owing to all convolution operations with the filter size of 5×5, a convolution-pooling module consists of 25 shift multipliers, 26 complement converters, an accumulator and a comparison unit. In this module, 25 16-bits signed intermediate results of 25 shift multipliers are first transformed into complement by 25 complement converters, and then accumulated by the accumulator. Applying 16-bits fixed-point decimal precision can minimize the loss of precision compared to truncated 8-bits fixed-point integer precision during the accumulation process, and an accumulation result is converted to the true form by the remaining complement converter, generating a 16-bits convolution result. The computing mode of this module can be configured to convolution or convolution-pooling by the CPU side. In the convolution mode, the 16-bits convolution result is output directly to the output register in each convolution operation. In the convolution-pooling mode, four 16-bits convolution results are sequentially temporarily stored in four internal registers during four convolution

operations. By comparing the values in four internal registers using the comparison unit, every four convolution operations generate a max-pooling result, which is then output to the output register. Finally, a 16-bits convolution result in the output register is truncated to 8-bits as an output of this module.

3) FC-ReLU module

A FC-ReLU module consists of 336 shift multipliers, 337 complement converters, an accumulator and a ReLU unit. The MAC process is similar to the convolution-pooling module, generating a 16-bits FC result. Whether the activation function is applied to a 16-bits FC result through the ReLU unit can be configured by the CPU side. The ReLU unit result is assigned to 0 if the sign bit of an input 16-bits FC result is "1", and maintains the original value if the sign bit is "0". A final result is truncated to 8-bits as an output of this module.

4) Overall architecture

Figure 6 is the overall architecture of the proposed accelerator, based on the processed LeNet-5 module, 7 convolution-pooling modules and a FC-ReLU module are instantiated in this accelerator for parallel computing. The internal memory, which is responsible for storing all the 5-bits weights required by the accelerator, consists of 116 rows of weights to meet the computation requirement by the input feature maps in the total 116 channels of the processed LeNet-5 module. Cooperating with the accelerator architecture, all input feature maps in the same channel are computed in parallel utilizing one row weights, and each row weights only be read once, which significantly reduces the requirements for reading weights frequently. The high performance AXI-HP interface is utilized to access to the DDR3 memory through AXI4 bus, where the accelerator as a master device and the DDR3 memory as a slave device. When all operations of the convolution layers and the pooling layers are completed, the last results of the 7 convolution-pooling modules are directly connected to the FC-ReLU module and written back to the DDR3 memory is not required, thus eliminating the requirements for the DDR3 memory accesses and data transmission through bus in the FC-ReLU module. The final feature vector with 10 8-bits pixels is stored in the DDR3 memory and read by the CPU.

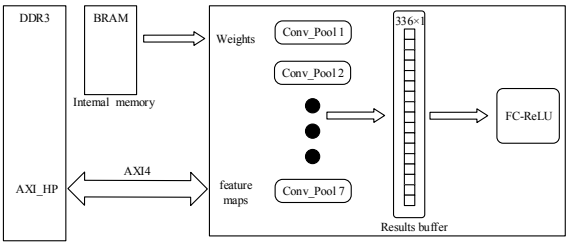

Figure 6. Overall architecture of the accelerator.

IV. EVALUATION

In this section, we introduce the implementation process of our design, and make a comparison to the reference designs.

A. Implementation Process

We compose the proposed modules in Section 3.4 into an accelerated IP core using Verilog HDL, where pipeline technology is applied. We debug and verify the IP core utilizing VCS simulation environment, and create a test bench based on it to obtain the accuracy of the IP core by testing 10000 images of MNIST test set. In Vivado IDE, we package the IP core with the interface of AXI4 and AXI4-Lite and then integrate this packaged IP core with the ZYNQ processing system and the block ram (for storing weights) to a SOC system. After synthesis, implementation, and generating the bit-stream file, the hardware message (including bit-stream file) is exported to the Xlinx Software Development Kit (SDK). In the SDK environment, the developed program run by one of the two ARM Cortex A9 CPUs is responsible for driving the IP core and reading an input handwritten digit image with the size of 28×28 in a SD card.

B. Experimental Results

1) Resource consumption

In Vivado IDE, fixed-point multiplication operations can be represented by Verilog HDL, where the multiplication operator "*" is synthesized to the DSP resources in a target FPGA, but the proposed shift multiplier is synthesized to the LUT resources. For the convolution-pooling module which performs 25 MAC operations per cycle, resource consumption comparison of utilizing multiplier with DSP and the shift multiplier is shown in Table II. In the ZC702 FPGA, 26 out of 220 DSP48E resources are consumed when the multiplication operator "*" is utilized, which accounts for 11.8% of the total DSP resources. However, utilizing the shift multiplier only consume 3.28% of the total LUT resources, so it is a reasonable choice for parallel computing owing to the abundant LUT resources in a FPGA can be utilized and the MAC operations are not limited by expensive DSP resources.

TABLE II. RESOURCE CONSUMPTION COMPARISON OF TWO SCHEMES

	Multiplier with DSP	**Shift multiplier**
LUT	115(0.22%)	1746(3.28%)
FF	111(0.1%)	984(0.92%)
DSP	26(11.8%)	0(0%)

2) Results comparison

As shown in Table III, we compare the implementation results between the proposed design with other designs using different FPGA chips and design schemes. It can be observed that our design is the unique hardware accelerator that does not utilize any DSP resources, and achieves better performance with relatively few resources, lower power consumption and most of all, a cheaper FPGA. Furthermore, a brief summary is given in Table IV, including the time consumption of the accelerator during the process of the developed program run by a CPU. We can see that it only accounts for 1.9% of the total time consumption, so the time consumption of the developed program can be optimized by

another CPU with better performance. Implementation in the development board with a xc7z020clg400 FPGA chip is shown in Figure 7.

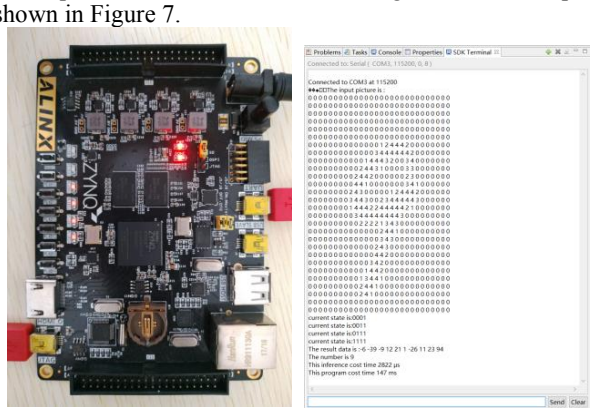

Figure 7. Implementation in the development board with a xc7z020clg400 FPGA chip and the results of the program run by an ARM Cortex A9 CPU.

TABLE III. RESULTS COMPARISON WITH PREVIOUS WORKS

	Yongmei [4]	**Kiran [10]**	**Jianfei [11]**	**This work**
Device	Virtex7 VX485T	Artix-7 AC701	ZYNQ ZC702	ZYNQ ZC702
Design scheme	Vivado HSL	Not Available	Not Available	Verilog
Frequency	150MHz	300MHz	Not Available	100MHz
Precision	11-bits fixed-point	32-bits floating-point	8-bits fixed-point	8-bits fixed-point
Latency (CC)	3815	17400	192100	177477
Peak performance	16.58 GMACS	Not Available	3.74 GOPS	33.6 GMACS
DSP consumed	83	616	92	0
Power	Not Available	12W	Not Available	1.758W
Accuracy	Not Available	90%	Not Available	98.9%

TABLE IV. SUMMARY

	DSP48E	**BRAM**	**LUT**	**FF**	**Time**
Total	220	140	53200	106400	147ms
Used	0	24	39898	25161	2816us
Utilization (%)	0	17.14	75	23.65	1.9

V. CONCLUSION

In this paper, we present an efficient method to implement a LUT-based FPGA accelerator for convolutional neural networks, and choose LeNet-5 as a demonstration. Results show that our scheme with a novel hardware architecture achieves great improvements on performance and resource utilization compared to the previous schemes. In future, we will attempt to implement a larger and more complicated convolutional neural network accelerator based on FPGA.

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems* 25 (2012), 84–90. DOI=http://dx.doi.org/10.1145/3065386.

[2] Yu-Hsin Chen, Tushar Krishna, Joel S.Emer, and Vivienne Sze. 2016. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52 (2016), 127–138. DOI=http://dx.doi.org/10.1109/ISSCC.2016.7418007.

[3] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *CoRR* abs/1510.00149 (2015).

[4] Yongmei Zhou and Jingfei Jiang. 2015. An FPGA-based accelerator implementation for deep convolutional neural networks. *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)* (2015), 829–832. DOI=http://dx.doi.org/10.1109/ICCSNT.2015.7490869.

[5] Mankit Sit, Ryosuke Kazami, and Hideharu Amano. 2017. FPGA-based accelerator for losslessly quantized convolutional neural networks. *2017 International Conference on Field Programmable Technology (ICFPT)* (2017), 295–298. DOI=http://dx.doi.org/10.1109/FPT.2017.8280164.

[6] Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. 1998. Gradient Based Learning Applied to Document Recognition. *Proc. IEEE* 86, 11 (1998), 2278 – 2324. DOI=http://dx.doi.org/10.1109/5.726791.

[7] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia* (2014). DOI=http://dx.doi.org/10.1145/2647868.2654889.

[8] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic Network Surgery for Efficient DNNs. *CoRR* abs/1608.04493 (2016).

[9] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. *CoRR* abs/1702.03044 (2017).

[10] Shahmustafa Mujawar, Divya Kiran, and Hariharan Ramasangu. 2018. An Efficient CNN Architecture for Image Classification on FPGA Accelerator. *2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC)* (2018), 1–4. DOI=http://dx.doi.org/10.1109/ICAECC.2018.8479517.

[11] Junye Si, Jianfei Jiang, Qin Wang, and Jia Bin Huang. 2018. An Efficient Convolutional Neural Network Accelerator on FPGA. *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)* (2018), 1–3. DOI=http://dx.doi.org/10.1109/ICSICT.2018.8565671.