IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 22, 2020, accepted September 7, 2020, date of publication September 14, 2020,
date of current version November 19, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3023739

AlphaGo Policy Network: A DCNN Accelerator on FPGA

ZHENNI LI, CAN ZHU, YU-LIANG GAO, ZE-KUN WANG, AND JIAO WANG
College of Information Science and Engineering, Northeastern University, Shenyang 110819, China

Corresponding author: Jiao Wang (wangjiao@ise.neu.edu.cn)

This work was supported by the Fundamental Research Funds for the Central Universities under Grant 2020GFYD011 and 2020 GFZD008.

ABSTRACT The game of GO has long been regarded as the most challenging game for artificial intelligence because of its enormous search space and the difficulty of evaluating its board positions. In early 2016, the defeat of Lee Sedol by AlphaGo became the milestone of artificial intelligence. AlphaGo's success lies in that it efficiently combines policy and value networks with Monte Carlo tree search (MCTS). And these deep convolutional neural networks (DCNNs) are trained by the combination of supervised learning and reinforcement learning. However, large convolution operations are computationally-intensive and typically require a powerful computing platform, for example, a graphics processing unit (GPU). Therefore, it is challenging to apply DCCNs in resource-limited embedded systems. Field programmable gate array (FPGA) is proposed to be an appropriate solution to implement real-time DCCNs models. However, the limited bandwidth and on-chip memory storage are the bottlenecks for DCCNs acceleration. In this article, an AlphaGo Policy Network is designed, and efficient hardware architectures are proposed to accelerate the DCCN model. The accelerator can be fit into different FPGAs, providing the balancing between processing speed and hardware resources. As an example, the AlphaGo Policy Network is implemented on Xilinx design suite VCU118, and the results show that our implementation achieved a performance of 3036.32 GOPS and achieved up to 56x speedup compared to CPU and 22.4x speedup compared to GPU.

INDEX TERMS AlphaGo, policy network, DCNN, accelerator, FPGA.

I. INTRODUCTION

The game of Go (weiqi in Mandarin pinyin) is a two-player board game tracing its origins to China more than 2,500 years ago. Although its rules are simple, Go has enormous complexity at the scale of 10^{170} in state space and 250^{150} in game tree. Therefore, the game of Go has long been considered as one of the hardest and the most difficult problems among all the major board games for Artificial Intelligence (AI) to solve. However, in 2016, AlphaGo, the Google Deepmind's computer Go program made a huge impact on both AI and the Go communities by defeating LEE Sedol, a multiple world championship winner, with four wins against one loss [1]. The main idea behind AlphaGo combines machine learning techniques of DCNNs and reinforcement learning with MCTS, where DCNNs play a very important role.

In recent years, there has been an increasing interest in DCNNs, due to their high accuracy and superior performance in tasks including image classification [2], [3], object detection [4], [5], and stereo vision [6], [7]. This technique has also become a successful algorithm in the region of AI, for example, in AlphaGo algorithm, and it is a strong candidate for many computer vision algorithms. Moreover, for some tasks, DCNNs are reported to be able to outperform humans [8], [9].

Despite the promising future and popularity of DCNNs, their implementations still exist many problems because of the computational complexity as well as the storage requirements. For example, the ImageNet dataset, which contains about 14 million tagged images [10], is the most frequently used data set for object recognition. With such a data set, DCNNs will consume vast amounts of computing power during training and deployment. Thus, DCNN is a computationally-intensive model, and specialized hardware such as GPU is needed. Binarized Neural Networks (BNNs) have very recently been proposed to offer an extremely more compact representation of network weights and neuron values than a normal DCNN [11], [12]. However, on account of BNN's poor expression ability and serious information loss

The associate editor coordinating the review of this manuscript and approving it for publication was Junxiu Liu.

caused by binarization, they are not suitable for complex tasks, such as the prediction in the game of Go.

The advantages of GPUs are that they have a high degree of parallelism and a good ability to accelerate DCNNs. As a deep learning platform, GPUs have become a common alternative to custom hardware in signal processing such as video processing in autonomous cars and medical devices [13], [14], because they have a comparatively low price, are powerful and easily programmable for training purposes. Nevertheless, the high-power consumption of GPUs limits their applications in mobile devices and in embedded systems. FPGAs, in contrast, are able to find an optimal trade-off between performance, energy efficiency, fast development and cost, therefore, they are widely adopted in neural network applications [15]–[17]. In addition, FPGA is a promising alternative target platform that can bridge the gap between power-hungry programmable architectures and fixed-efficient ASICs. For example, in the IoT (Internet of Things), training is usually performed in the cloud or on edge nodes, and inference is performed using edge devices, which are power- and size- (resource) constrained. The motivation behind deploying such a network on the edge with FPGA is to balance between low power consumption and high performance. Thus, FPGA design of DCNNs has become an active research topic in recent years [18]–[20].

Various hardware architectures have been proposed to accelerate DCNNs using FPGAs. In many cases, it becomes essential to know how the changes in hardware architectures will affect performance. Zhang *et al.* [21] proposed a uniformed convolutional matrix multiplication architecture to accelerate both computation-bound convolutional layers and communication-bound fully connected layers of CNN/DNN on FPGAs, which achieved up to 365 GOPS on KU060 board and 636 GOPS on VC707 board. However, the system could not achieve higher performance. In [22], Huang *et al.* presented a design and implementation of a 23-layer SqueezeNet on FPGA by using a new layer-based structured design method, its peak performance was 213.7 GOP/s under 110MHz with 79.05% top-5 accuracy. But in terms of power efficiency, the system still has a very large improvement space. In [23], a data quantization strategy was proposed to compress the bit-width used in DCNNs. Evaluated on the state-of-the-art DCNN models, this strategy brings negligible performance loss with 16- and 8-bit configuration. In [24], Qu *et al.* proposed a grain-adaptive computing structure for FPGA-based DCNNs acceleration, which can change flexibly and can optimally utilize the available DSP resources.

Different from GPU or other FPGA works which transform convolution to matrix multiplication, in this article, we present a pipelined Policy Network DCNN model which is adopted by AlphaGo. Moreover, we implement the system on Xilinx Design Suite VCU118, which provides enough memory resources. And experiments on GPU and CPU are also carried out in the same process of forward propagation. Compared with previous works, our experiments on FPGA show that the system designed obtains the overall performance is 3036.32 GOPS, which improves at least by 2.35x, and the power efficiency is 101.21 G/s/W, resulting in 2.35x improvement at least.

The rest of the paper is organized as follows. Section II provides the fundamental knowledge of AlphaGo Policy Network and the algorithm. Details of hardware architectures are shown in Section III. System implementation and its results are discussed in Section IV. The paper is concluded in Section V.

II. ALGORITHM

The algorithm of AlphaGo can be decomposed into two sequential parts: policy network and value network. In this article, an AlphaGo policy network is designed, and efficient hardware architectures are proposed to accelerate its DCCN model.

The recent breakthrough in DCNNs has led to improvements in the accuracy of both vision and auditory systems, such as image recognition, object detection and speech recognition [2]–[7]. DCNNs are generally composed of several cascades of basic layers. The most important layers are the convolutional layers. It can simplify calculations, reduce the number of weight parameters and the complexity of the entire convolutional neural network model by using local sensing and sharing connection weight. Generally, a convolutional layer can further contain three main functions: convolution kernel, non-linear neuron, and spatial pooling [25].

The convolutional layer can apply the matrix data of previous layer to the filter. Each filter is a local perceptual field and is called a convolution kernel. It can be expressed as follows:

$$g(x,y,z) = \sum_{i=0}^{C-1}\sum_{j=0}^{C-1}\sum_{k=1}^{I} f(x+i, y+j, k)\cdot c_z(i,j,k) = \vec{f}\cdot\vec{c}_z \quad (1)$$

where the vectors $\vec{f}$ and $\vec{g}$ represent the input and output feature map in the form of 3D matrix respectively, the size of the convolution kernel is $C \times C$, and I is the channel number.

The non-linear neuron is called activation function, which has the ability to solve non-linear problems. Rectified Linear Unit (ReLU) function is the most common one, as shown in equation (2). Paper [26] demonstrates that training convergence can be much faster in the network with ReLU neurons than that with other neurons.

$$f(x) = max(f(x), 0) \quad (2)$$

Spatial pooling is known as the pooling layer, which is cascaded after non-linear neurons. Pooling layer can be taken as a special convolution operation [27]. It merges the neighbor area of input feature map in order to reduce data amount and keep local invariance. There are two widely used pooling methods: the average pooling method and the maximum pooling method. The average pooling method averages all the inputs while the maximum pooling method chooses the maximum value [28].

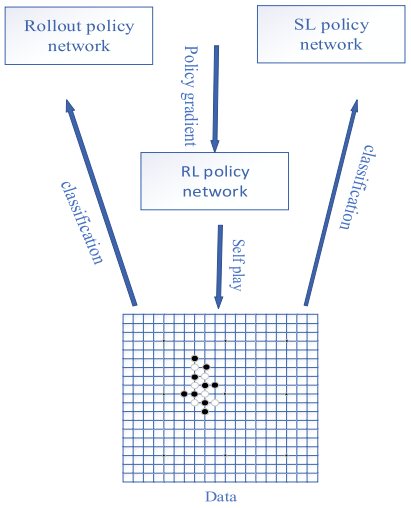


FIGURE 1. AlphaGo policy network training architecture. There are three parts in AlphaGo policy network: rollout policy network, SL policy network and RL policy network. Firstly, AlphaGo takes SL to train the SL policy network and rollout policy network in learning what the human experts will play under certain patterns. Then AlphaGo takes RL to gain further improvement of the policy network through millions of self-played games.

In our DCNN implementation, an AlphaGo policy network is designed. Each point in the 19 × 19 feature map is a simple representation of the board state in the game of Go, so that the data cannot be sampled or averaged. Besides, ReLU activation function is a linear function, which alleviates the disappearance of gradient and improves the efficiency in the calculation process by removing index calculation. Therefore, we choose ReLU function as non-linear neurons without spatial pooling.

As shown in Fig.1, there are two steps in the work flow of training AlphaGo policy network. Training supervised learning (SL) policy network ρ_σ and rollout policy network p_π is the first stage, which is designed to predict human expert moves in a data set of positions. The next step is to train a reinforcement learning (RL) policy network p_ρ, which improves the SL policy network by optimizing the outcome of games [1].

The policy network is trained on randomly sampled state-action pairs (s, a), using stochastic gradient ascent to maximize the likelihood of human move a selected in state s, which is a simple representation of the board state, as shown in equation (3) [1]:

$$\Delta\sigma \propto \frac{\partial \log p_\sigma (a \mid s)}{\partial \sigma} \tag{3}$$

where σ represents the weight, and $p_\sigma (a \mid s)$ denotes the probability distribution output of the SL policy network.

The SL policy network is a 13-layer policy network. The network predicted expert moves on a held-out test. The fast rollout policy network p_π is trained using a linear softmax of the small pattern features, which has the capability to select an action faster but with the penalty of less accuracy.

Besides, the RL policy network p_ρ is identical in structure to the SL policy network. Its weights are updated by stochastic gradient ascent in the direction that maximizes the expected outcome, as shown in equation (4) [1]:

$$\Delta\rho \propto \frac{\partial \log p_\rho (a_t \mid s_t)}{\partial \rho} z_t \tag{4}$$

where ρ represents the weight, $p_\rho (a \mid s)$ denotes the probability distribution output of the RL policy network, and z_t is the terminal reward at the end of the game from the perspective of the current player at time step t.

III. HARDWARE ARCHITECTURE DESIGN

An overview of the proposed AlphaGo policy network architecture is discussed here. Policy network is designed to sample actions in the game of Go. Reinforcement learning is combined with supervised learning to adjust the policy network towards the correct goal of winning games. It takes a representation of the board position as its input, passes it through many convolutional layers with parameters (SL Policy Network or RL Policy Network), and outputs a probability distribution over moves a, which is represented by a probability map over the board. As we can see, Fig. 2 shows a high-level block diagram of the proposed architecture. It consists of a signal preprocessing module, 13 convolutional

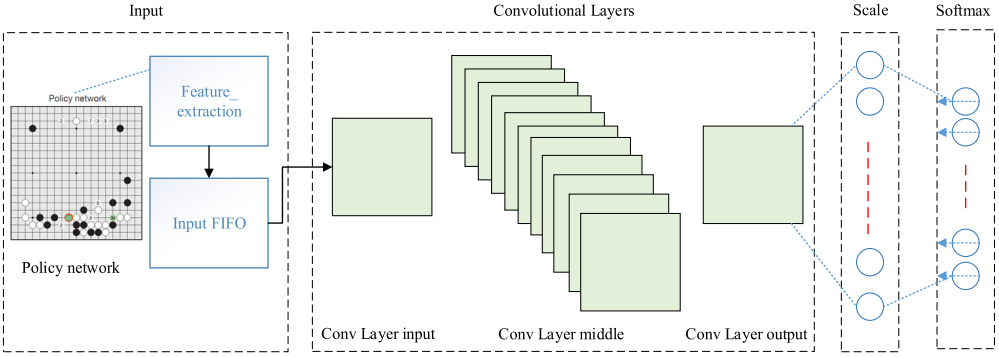


FIGURE 2. The proposed AlphaGo policy network architecture with different kinds of layers. The network takes the feature graph matrix data of the board as input, and consists of 13 convolutional layers, 12 ReLU functions, a scale layer and a softmax layer.

layers, a scale layer and a softmax layer that is equivalent in size to the number of class labels. Different from GPU or other FPGA works which transform convolution to matrix multiplication, the DCNN proposed is designed according to the original definition of convolutional neural network, and a pipelined architecture is designed to calculate 8 pixels of output feature map in parallel.

A. SIGNAL PREPROCESSING

Each position on the board is considered as a situation, which is the input of the policy network. Before training, feature graph matrix data of the board is extracted from the input data and then imported to the input FIFO. The input FIFO is a synchronous FIFO, which is composed of RAM with a depth of 4096 on FPGA, and it works in first-word fall-through mode.

B. CONVOLUTIONAL LAYERS

The DCNN of AlphaGo policy network consists of 13 convolutional layers, which are then divided into three modules based on the different amounts of features, that is, the input convolution module, the intermediate convolution module, and the output convolution module. And in order to maximize resource utilization of the FPGA used, the 9-bit fixed point parameter is chosen to be the quantization strategy.

1) THE INPUT CONVOLUTION MODULE

The input convolution module, including one layer, is shown in Fig.3, where 192 convolution kernels with a shape of 5×5 are chosen based on the differences in data and in the complexity of models. It includes a weight serial-parallel conversion module, a weight memory (WMEM), and a parallel multiply-sum operation module. In the first step, the weight serial-parallel conversion module converts the input serial weight data to parallel data and stores them in the weight memory WMEM. Then, the parallel data and feature map data which have a size of $4 \times 19 \times 19$ are sent into the parallel multiply-sum operation module for the convolution operations. Finally, the sum of the operation results and bias are sent to the intermediate convolution module.

The parallel multiply-sum operation module is designed to perform the basic operations. After the data of the feature map and the data of the convolution kernel is retrieved by the serial-parallel conversion module, the corresponding data needs to be multiplied. When the size of the convolution kernel is 5×5, five multiplication results are accumulated. The calculation process of this part is shown in Fig. 3.

In the proposed network, there are thirteen convolutional layers. Therefore, improving the speed of convolution operations is the top priority for parallel acceleration. The mathematical operations are very simple, however, in terms of hardware implementation, it is necessary to consider the corresponding timing arrangement of the feature map data. If the feature map data of the convolutional layer needs to be filled, it should not affect the characteristics of the input matrix. The multiplication and other operations of the convolution

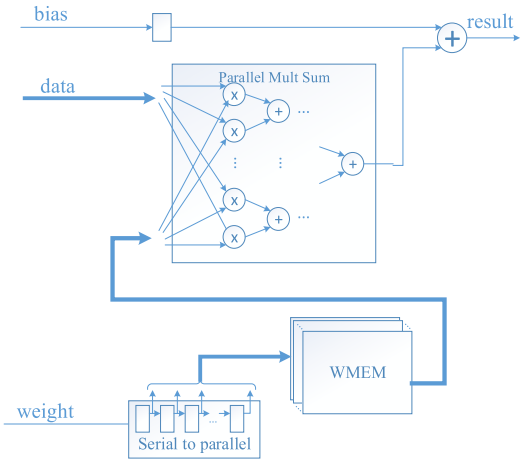


FIGURE 3. Input convolution module architecture.

mapping module are performed simultaneously, in order to achieve parallel acceleration. Besides, the data of each module is cached separately in order to relieve the pressure of cache and meet the overall design requirements of the policy network.

The process of convolution operations between the feature map of the convolutional layers and the parameters of the weight matrix is shown in Fig. 4, where Conv1_data denotes the input data of the feature map in the convolutional layer,

and Conv_data denotes the data used in the convolution operations performed by the convolution kernel. The reconstruction of the stored procedure takes advantage of ping-pong operation of FPGA. To process the convolution operation of a two-dimensional matrix in FPGA, it is necessary to ensure that the data flow and the operations can be mapped one by one in the correct order. The size of the convolution window in the feature map is the same as the size of the convolution kernel Conv_kernel1. After the same number of clocks, Conv1_data and Conv_data in the feature map need to be buffered, then are output according to the control of the clock signal and finally are transmitted to the basic unit of the convolution operation.

In the process of cache management and convolution calculations, a state transition machine is designed. There are two states: WAIT (waiting state) and CALC (calculation state). The state transition diagram is shown in Fig. 5.

In this state transition machine, when the reset signal is ready, the FSM (Finite State Machine) will not stop switching between the WAIT state and CALC state until the power is cut off. As long as the buffering data is completed, the state transition will be performed by a simple enable signal. In the buffered clock, the status of this module is always WAIT until the enable signal changes. When the flag bit calc_rdy that detects whether calculation can be performed is effective, the status of this module will get into CALC state. In CALC state, the parameters such as the number of cycles calculated, should be adjusted. These parameters are used as counters, and could adjust and output the data for the next mapping operation.

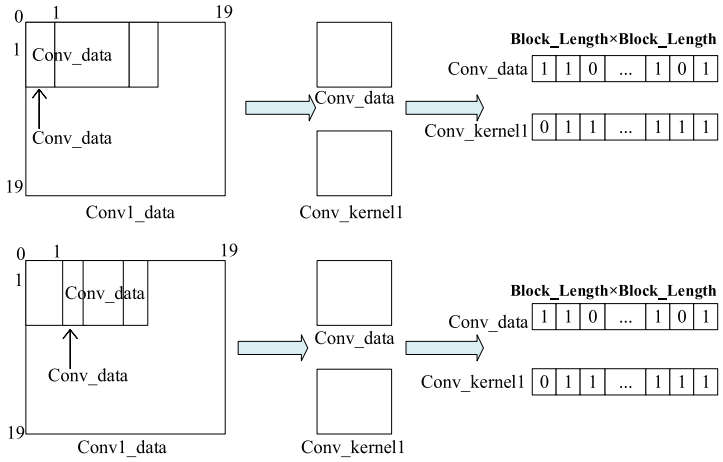


FIGURE 4. Split of feature maps and computation with a convolution kernel.

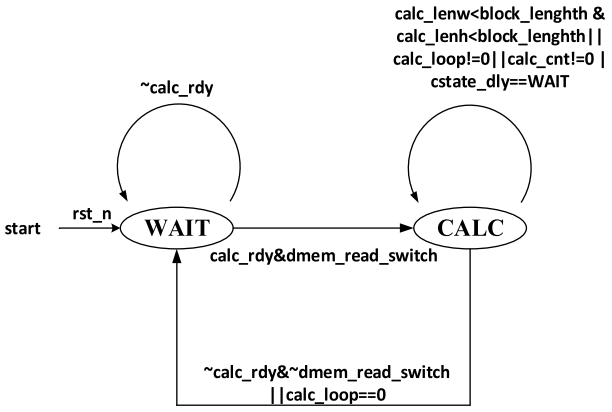


FIGURE 5. States transition in convolution layer.

2) THE INTERMEDIATE CONVOLUTION MODULE

The proposed intermediate convolution module shown in Fig. 6 has a more complex structure compared to the input convolution module. It consists of the second to the twelfth convolutional layers, and each convolutional layer contains 192 convolution kernels with a shape of 3×3. ReLU function is applied twelve times in the convolutional layers of the input convolution module and the intermediate convolution module.

In the intermediate convolution operations, we need to take the data of adjacent rows, however, they cannot be sent to each other. Therefore, the data to be calculated needs to be stored at first. For the input of a two-dimensional matrix, a block fetch module is designed to perform data caching of the convolution feature map and to convert the data into multiple data blocks. In this case, the data stream is input by rows, from left to right, and from up to bottom.

Fig. 7 gives an overview of block fetch module. After several clocks in the convolution window, the block fetch module outputs multiple feature maps that need to be convolved. It is the first step of the convolution operations. If the size of the input feature map is K$\times$ K, and the step size is 1, this module is used to split the feature map into (K/N) $\times$(K/N) data blocks of size N$\times$ N. In order to save time, after the first N lines of the input data stream and the Nth data of the N-1th line are input, the data of the first data block will be stored and output. When each clock cycle ends, the data of the input line and the data of $N \times (N-1)$ region in the upper left corner will be combined into a data block. In terms of resource allocation, we use shift registers to perform shifts and data buffering.

All the input feature map data generated by the input convolution module is stored in data memory DMEM. Since the depth of the convolutional layer is considerable, so that the feature map data and weight data it transformed are sent to the intermediate parallel multiply-sum operation module for convolution operations. After the operation results and the bias data are summed, the bit width is adjusted, and the results which are controlled by the FSM are output to the next module.

3) THE OUTPUT CONVOLUTION MODULE

The output convolution module shown in Fig. 8 has only one convolutional layer, which is the thirteenth convolutional layer of the policy network, and the shape of the convolution kernel is 1×1. The final results are the feature map data which will be sent to the scale module.

C. SCALE MODULE

The scale module, on one hand, improves the accuracy of the input data, and on the other hand, converts the fixed-point input data to 32-bit floating-point data which will be then transferred to the softmax module. At run time, RAM is used to read and write data. When the write enable signal is valid, the two input data signals are written into the RAM. Fig. 9 shows the architecture of the scale module.

In this layer, there is no ReLU function, and the input data is the output of the 13^{th} convolutional layer. Since FPGA resource utilization of softmax layer is much lower

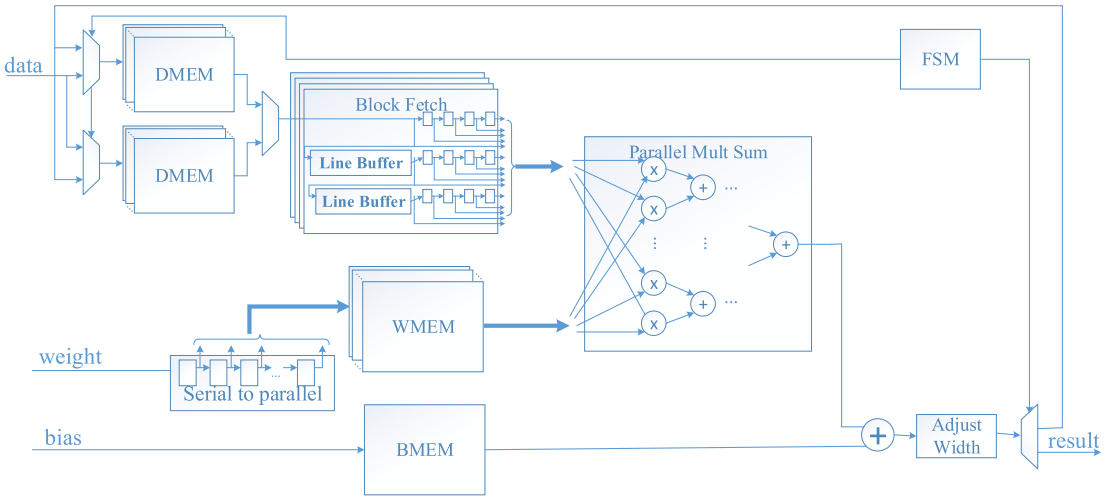


FIGURE 6. The intermediate convolution module architecture includes eight modules: the data memory DMEM, the data serial-parallel conversion module, the intermediate weight serial-parallel conversion module, the weight memory WMEM, the bias memory BMEM, the intermediate parallel multiply-sum operation module, the finite state machine module FSM and the bit width adjustment module.

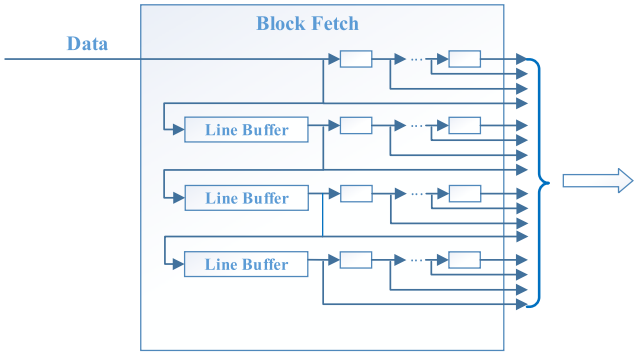


FIGURE 7. Block Fetch sub-module.

than convolutional layers, and its fixed-point implementations can introduce large errors, which can be alleviated by converting between fixed-point and floating-point formats for particularly accuracy-sensitive operations. Therefore, 32-bit floating-point data is used directly in softmax layer.

D. SOFTMAX MODULE

There are no pooling layers in the network, and the softmax module is utilized to normalize the input data. The softmax module performs exponential calculation on the data stream generated by the scale module as shown in Fig. 10, and all the exponential calculation results are then transferred to the accumulator. Then, we take each numerical exponential calculation result as the dividend and the output of the accumulator as a divisor. The final result of the softmax module is calculated as in equation (5):

$$a_i = \frac{e^{x_i}}{\sum_{i=0}^{n} e^{x_i}} \tag{5}$$

where x_i denotes the input of the softmax module, and a_i denotes the output of the softmax module, i corresponds to the output probability for each class.

IV. RESULTS

A. EXPERIMENTAL SETUP

In this work, the hardware platform used to evaluate our design is Xilinx VCU118 with a FPGA chip named XCVU9P-L2FLGA2104E. The working frequency is 200MHz. This FPGA provides many common functions of the development system, such as, dual DDR4 memory and RLD3 constitute memory, PCI Express interface, Gen3 x16 interface, Ethernet interface, several indicators, general I/O interface, two UART interfaces, FPGA Mezzanine Card FMC) interface, two four-channel SFP fiber module with the 28 interface, and RS-FEC150G Interlaken core module. Besides, the module of the development board can be compatible with VITA-57.1 FMC interface and CITA-57.4 FMC+HSPC, which can realize more functional characteristics. FMC connection board is used to connect with the host, which can provide data for the forward propagation process of the convolutional neural network. The physical connection diagram of the realized system is shown in Fig. 11.

B. RESULT ANALYSIS AND COMPARISON

A comparison with prior FPGA works is discussed in this section. And the goal is to compare the throughput performance and power efficiency of the DCCNs realized in our work with other FPGA works. It is worthy to mention that, because the reasonable design of DCNNs architecture, we achieve a more satisfactory result in both overall throughput performance and power efficiency. Table 1 shows the experimental results. The results reveal a big difference between our hardware design and other approaches. As can be seen, our overall performance is 3036.32 GOPS, resulting in 11.41x and 14.2x, 36.01x and 2.35x improvement compared with the sate-of-the-art implementation of DCNNs on FPGA in [16]–[18] and [19], correspondingly. Besides, in terms of

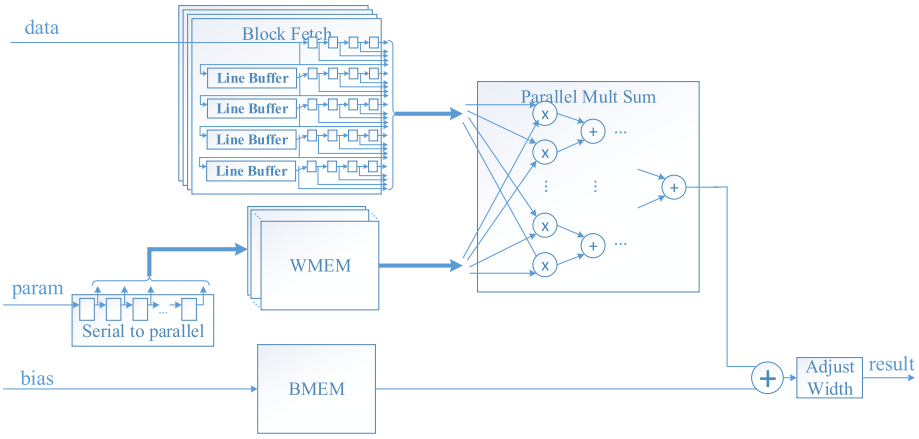


FIGURE 8. Output convolution module architecture.

TABLE 1. Comparison with other accelerators.

	[16]	[17]	[18]	[19]	ours
CNN	VGG-16	SqueezeNet	VGG-16	AlexNet	AlphaGo PolicyNet
Number of Parameters	13554712	421098	13554712	60965128	4444032
FPGA	Ultrascal-e KU060	Xilinx VC709	Zynq XC7Z020	Xilinx xcku115	Xilinx VCU118
CLOCK(MHz)	200	110	214	150	200
Quantization Strategy	16 bit fixed	8-16 bit fixed	8 bit fixed	16 bit fixed	9 bit fixed
Throughput (GOPS)	266	213.7	84.3	1292.4	3036.32
Power(W)	25	27.7	3.5	30.05	30
Power Efficiency (G/s/W)	-	7.71	24.1	43.01	101.21
DSP	1058(38%)	1897(53%)	190(86.4%)	5027(91.1%)	6840(100%)
FF	2276(11%)	315465(36%)	35489(33%)	242265(18.3%)	1111306(47%)
LUT	485(31%)	324793(75%)	29867(56%)	203347(30.7%)	673877(57%)

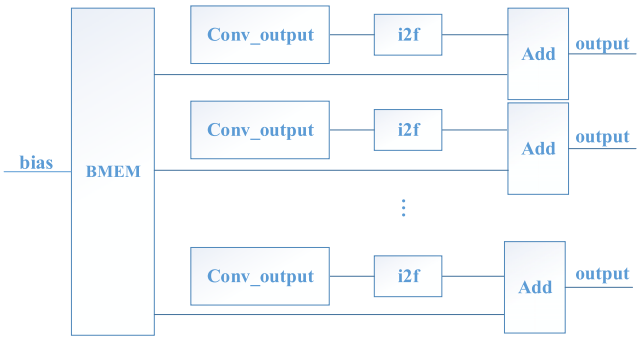


FIGURE 9. Scale module architecture.

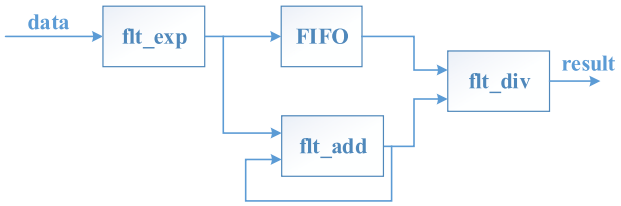


FIGURE 10. Softmax module architecture.

power efficiency, our system obtained a better performance, which is 101.21 G/s/W. We also evaluate a GEMM (General Matrix Multiplication) method for proposed DCNN architecture on FPGAs and got an 867.27 GOPS. Compared with previous works, we improve the power efficiency by 2.35x at least.

The experimental results show that the performance of our design is higher than most of the previous works. Also, our work significantly outperforms all three prior studies in all terms of performance. In our work, we nearly approach the peak performance of the DCNNs accelerator on FPGA.

C. COMPARISON WITH CPU AND GPU

In this section, we compare our FPGA design with CPU and GPU solutions. Table 2 shows the comparison between our accelerator on FPGA and that on CPU and GPU. In order to verify the parallel acceleration effect under this network model, the comparison experiment uses a CPU with higher performance and a single GPU as the device platform for verification and uses the Caffe deep learning platform for calculations. We compare four terms of performance: 1) power, 2) average performance, 3) the speed up and 4) the performance on different platforms. On the VCU118 platform,

FIGURE 11. Physical connection diagram of the realized system.

TABLE 2. Comparison with CPU/GPU platform.

Platform	CPU	GPU	FPGA
Device	E5-2640	GTX1080	VCU118
Technology	32nm	16nm	16nm
Freq	2.4GHz	1607MHz	200MHz
Power(W)	115	250	30
Average performance(Gops)	58.6	1186	3036.32
Speed up	1x	22.4x	56x
Energy efficiency (Gops/W)	1x	9.31x	198.64x

our accelerator achieves an average performance of 56x over CPU solution and 22.4x over GPU solution, as is shown in TABLE 3. We then evaluate the performance of the energy efficiency with the same DCNN architecture. The FPGA exceeds the CPU by 198.64x and exceeds the GPU by 21.34x.

V. CONCLUSION

In this work, we present the Policy Network model, which is adopted by AlphaGo and propose a hardware accelerator to improve the system throughput and performance. The Policy Network includes signal preprocessing module, convolutional layers, scale module, and softmax module. We propose to divide the DCNN's convolutional layers into three parts and present a parallel acceleration mechanism in each part. To exploit the parallelism in DCNNs, multiple hardware optimization strategies are adopted in the convolutional layers, achieving a good balance among speed, resource usage and power consumption. Besides, we use a pipelined method to transmit and process the data stream and make full use of the on-chip resources and the parallelism of FPGA with Xilinx VCU118 as the hardware platform. Experimental results show that our implementation achieved a performance of 3036.32 GOPS and achieved up to 9.31x energy efficiency compared to CPU and 198.64x energy efficiency compared to GPU. More prominently, in terms of processing time, the proposed DCNN accelerator is superior to CPU and GPU. The hardware design and the corresponding verification experiments have shown that the hardware architecture proposed could not only complete the DCNN of the Policy Network in AlphaGo but also take the advantages of FPGA to optimize the DCNN performance. What's more, the aim of the presented implementation is to maximize resource utilization of the FPGA used, and the bit width could be decreased on account of some accuracy degradation, and that this could be investigated in our future works.

REFERENCES

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–529, 2016, doi: 10.1038/nature16961.

[2] M. Al-Hami, M. Pietron, R. Casas, and M. Wielgosz, "Methodologies of compressing a stable performance convolutional neural networks in image classification," *Neural Process. Lett.*, vol. 51, no. 1, pp. 105–127, Jul. 2019, doi: 10.1007/s11063-019-10076-y.

[3] Y. Li, H. Zhang, X. Xue, Y. Jiang, and Q. Shen, "Deep learning for remote sensing image classification: A survey," *Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery*, vol. 8, no. 6, p. e1264, Nov. 2018, doi: 10.1002/widm.1264.

[4] H. Wang, J. Xu, L. Li, Y. Tian, D. Xu, and S. Xu, "Multi-scale fusion with context-aware network for object detection," in *Proc. 24th Int. Conf. Pattern Recognit. (ICPR)*, Beijing, China, Aug. 2018, pp. 2486–2491.

[5] T. Ching *et al.*, "Opportunities and obstacles for deep learning in biology and medicine," *J. Roy. Soc. Interface*, vol. 15, no. 141, Apr. 2018, Art. no. 20170387, doi: 10.1098/rsif.2017.0387.

[6] K. Ichimaru, R. Furukawa, and H. Kawasaki, "Multi-scale CNN stereo and pattern removal technique for underwater active stereo system," in *Proc. Int. Conf. 3D Vis. (3DV)*, Verona, Italy, Sep. 2018, pp. 61–70.

[7] C. Qu, M. Moiseikin, S. Voth, and J. Beyerer, "CNN-based image denoising for outdoor active stereo," in *Proc. 16th Int. Conf. Mach. Vis. Appl. (MVA)*, Tokyo, Japan, May 2019, pp. 1–6.

[8] T. H. Krishna and C. Rajabhushnam, "Mammographic cancer identification by using DCNN (deep convolutional neural network)—A review," in *Proc. 3rd Int. Conf. I-SMAC (IoT Social, Mobile, Analytics Cloud) (I-SMAC)*, Palladam, India, Dec. 2019, pp. 440–445, doi: 10.1109/I-SMAC47947.2019.9032607.

[9] H.-J. Lee, K.-S. Hong, H. Kang, and S. Lee, "Photo aesthetics analysis via DCNN feature encoding," *IEEE Trans. Multimedia*, vol. 19, no. 8, pp. 1921–1932, Aug. 2017, doi: 10.1109/TMM.2017.2687759.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Miami, FL, USA, Jun. 2009, pp. 20–25.

[11] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Xi'an, China, Dec. 2016, pp. 77–84.

[12] C. Fu, S. Zhu, H. Chen, F. Koushanfar, H. Su, and J. Zhao, "SimBNN: A similarity-aware binarized neural network acceleration framework," in *Proc. IEEE 27th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, San Diego, CA, USA, Apr. 2019, p. 319.

[13] A. A. Shvets, A. Rakhlin, A. A. Kalinin, and V. I. Iglovikov, "Automatic instrument segmentation in robot-assisted surgery using deep learning," in *Proc. 17th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Orlando, FL, USA, Dec. 2018, pp. 624–628.

[14] A. A. Shvets, V. I. Iglovikov, A. Rakhlin, and A. A. Kalinin, "Angiodysplasia detection and localization using deep convolutional neural networks," in *Proc. 17th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Orlando, FL, USA, Dec. 2018, pp. 612–617.

[15] S. I. Venieris and C.-S. Bouganis, "FpgaConvNet: Mapping regular and irregular convolutional neural networks on FPGAs," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 2, pp. 326–342, Feb. 2019, doi: 10.1109/TNNLS.2018.2844093.

[16] A. Ahmad and M. A. Pasha, "Towards design space exploration and optimization of fast algorithms for convolutional neural networks (CNNs) on FPGAs," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Florence, Italy, Mar. 2019, pp. 1106–1111.

[17] T. Kowsalya, "Area and power efficient pipelined hybrid merged adders for customized deep learning framework for FPGA implementation," *Microprocess. Microsyst.*, vol. 72, Feb. 2020, Art. no. 102906, doi: 10.1016/j.micpro.2019.102906.
[18] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of FPGA based neural network accelerator," 2018, *arXiv:1712.08934*. [Online]. Available: https://arxiv.org/abs/1712.08934
[19] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, "Accelerating CNN inference on FPGAs: A survey," 2018, *arXiv:1806.01683*. [Online]. Available: http://arxiv.org/abs/1806.01683
[20] A. G. Blaiech, K. B. Khalifa, C. Valderrama, A. C. M. Fernandes, and M. H. Bedoui, "A survey and taxonomy of FPGA-based deep learning accelerators," *J. Syst. Archit.*, vol. 98, pp. 331–345, Sep. 2019, doi: 10.1016/j.sysarc.2019.01.007.
[21] C. Zhang, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *Proc. 35th Int. Conf. Comput.-Aided Design*, Nov. 2016, pp. 1–8.
[22] C. Huang, S. Ni, and G. Chen, "A layer-based structured design of CNN on FPGA," in *Proc. IEEE 12th Int. Conf. ASIC (ASICON)*, GuiYang, China, Oct. 2017, pp. 1031–1047.
[23] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2018, doi: 10.1109/TCAD.2017.2705069.
[24] X. Qu, Z. Huang, N. Mao, Y. Xu, G. Cai, and Z. Fang, "A grain-adaptive computing structure for FPGA CNN acceleration," in *Proc. ASICON*, Chongqing, China, 2019, pp. 1–4.
[25] Y. Wang, L. Xia, T. Tang, B. Li, S. Yao, M. Cheng, and H. Yang, "Low power convolutional neural networks on a chip," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 129–132.
[26] I. Sacramento, M. Roisenberg, R. Exterkoetter, L. P. de Figueiredo, and B. Rodrigues, "Combined convolutional neural network for high frequency restoration in acoustic impedance images," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Rio de Janeiro, Brazil, Jul. 2018, pp. 1–8.
[27] Y. Tian, W. Zhang, Q. Zhang, G. Lu, and X. Wu, "Selective multi-convolutional region feature extraction based iterative discrimination CNN for fine-grained vehicle model recognition," in *Proc. 24th Int. Conf. Pattern Recognit. (ICPR)*, Beijing, China, Aug. 2018, pp. 3279–3284.
[28] G. Tolias, R. Sicre, and H. Jégou, "Particular object retrieval with integral max-pooling of CNN activations," in *Proc. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–12.

ZHENNI LI was born in 1982. She received the degree in engineering and the M.S. degree in electronics from ENSEEIHT, France, in 2008, the M.S. degree from Beihang University, in 2009, and the Ph.D. degree in detection technology and automatic equipment from Northeastern University, Shenyang, China, in 2018.

She is currently a Lecturer with Northeastern University. Her research interests include embedded system and video processing, asynchronous NoC design, and intelligent perception.

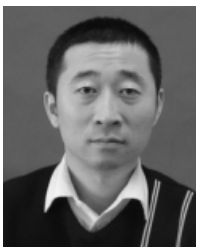
CAN ZHU received the B.S. degree in automation from Northeastern University, Shenyang, China, in 2019, where she is currently pursuing the master's degree in pattern recognition and intelligent systems.

Her research interests include self-driving car and deep reinforcement learning.

YU-LIANG GAO received the B.S. degree in electronic science and technology from Northeastern University, Shenyang, China, in 2017, where he is currently pursuing the master's degree in circuits and systems.

His research interests include hardware computing, wireless communications, and visible light communications.

ZE-KUN WANG received the M.S. degree in circuits and systems from Northeastern University, Shenyang, China, in 2014. He is currently an IC Design Engineer at Broadcom Corporation.



JIAO WANG received the Ph.D. degree in pattern recognition and intelligent systems from Northeastern University, Shenyang, China, in 2006.

He is currently a Professor with the College of Information Science and Engineering, Northeastern University. His research interests include hardware computing and computer games.