FPGA Implementation of CNN for Handwritten Digit Recognition

Rui Xiao[1,2], Junsheng Shi[*1,2], Chao Zhang[1,2]
1. School of Physics and Electronic Information, Yunnan Normal University, Kunming, China
2. Yunnan Key Lab of Optic-electronic Information Technology, Kunming, China

***Abstract*—Convolutional neural networks (CNN) have been used very successfully in the field of handwritten digit recognition. CNN is difficult to deploy on the embedded platform because of its large computation, complex structure and frequent memory access. In this paper, a manual hardware-level design (i.e., RTL) CNN reconfigurable IP core method is proposed to construct the FPGA basic unit of the CNN structure. Reconfigurable convolution, pooling and fully connected modules in the CNN structure are designed. By setting the parameters of the reconfigurable modules and connecting these modules end-to-end, CNNs of different structure sizes can be quickly deployed. In the full connection module, using double cache technology can reduce the memory access operation of weight parameters and improve the calculation efficiency. In addition, this paper presents an FPGA implementation of CNN for handwritten digit recognition system. The system was built end-to-end using a reconfigurable IP core and was finally implemented on the Intel Cyclone10 FPGA hardware platform. Under the 150MHz clock, CNN only needs 0.0176 ms to recognize a handwritten digital picture, and the accuracy rate is 97.57%.**

Keywords—Convolutional neural networks; FPGA; RTL; Handwritten digit recognition

I. INTRODUCTION

In the past few years, deep learning has become one of the most popular research directions in the computer field. In particular, Convolutional Neural Networks (CNN) have performed well in application areas such as image classification, recognition, detection, and segmentation, and have attracted more and more attention. CNN has great application prospects in UAV, bionic robot, wearable devices and other embedded applications. However, CNN occupies a large amount of resources and a large amount of memory, and has a large amount of calculation. Deploying CNN on mobile devices has become one of the challenges facing current deep learning applications. Due to the high performance, high energy efficiency, reconfigurability, and low latency of FPGA, deploying CNN on FPGAs has more advantages than other hardware platforms (such as CPU, GPU, ASIC, ARM).

FPGA has become the preferred hardware platform for accelerating the forward inference phase of deep convolution neural network (CNN), but deploying CNN on FPGA also encountered many challenges [1]. The implementation of CNN needs to perform complex matrix multiplication and memory access operations. The bottleneck lies in how to improve the calculation speed and memory bandwidth. Most of the calculations in CNNs occur in the convolutional layers [2-3], which contain a large number of multiply-add operations. At present, most designs use spatial convolution or general matrix multiplication (GEMM) to implement convolution operations [4]. These two methods are relatively mature and can be quickly deployed on FPGAs. However, these two methods require more multiply-add operations and huge memory overhead. References [5-6] improves the performance of convolution calculations by optimizing the parallel strategy and pipeline design. There are also other fast algorithms [7] that effectively reduce the number of multiplications of the convolution operation by transforming the domain method. Reference [8] deployed a small CNN on a single-chip FPGA and implemented handwritten digit recognition.The recognition accuracy of the MNIST dataset reached 90%. Although there have been many successful solutions for deploying CNNs on FPGAs, whether they use high-level synthesis language (i.e., HLS) or manual hardware-level design (i.e., RTL), deployment is more complicated and the design cycle is long [9]. We hope to analyze the loop, unroll, and exchange operations of each layer of CNN, design a general IP core, and change configuration parameters to build CNN models with different network structures.

This article introduces a method of rapidly deploying CNN on FPGA. By analyzing the structure and calculation method of each layer of the CNN model, RTL-level design of CNN's convolution, pooling, and fully connected reconfigurable IP cores is performed. Analyzing the structure of the CNN model, rationally setting the parallelism of each layer, optimizing the structure of the calculation hardware circuit of each layer, and parameterizing the configuration of the general IP core can speed up the construction of CNN. In addition, the central idea of this paper is to use a reconfigurable general-purpose IP core, set up efficient parallel pipeline convolution circuits, and by using the principle of on-chip dual cache weight parameter, the bandwidth requirement of fully connected layer (FC layer) is reduced. Finally, using the network trained on Keras, a CNN-based handwritten digit recognition system was implemented on the FPGA.

II. CNN MODULE DESIGN

CNN is a typical supervised feedforward neural network, which usually includes multi-layer of convolutional layer (CONV layer), pooling layer, and fully connected layer (FC layer). These layers are connected in order from input to output. The output feature map of one layer is the input feature map of the next layer, which has natural inter-layer flow characteristics. This section uses RTL level to design reconfigurable CNN IP

* Send all correspondence to: Junsheng Shi (e-mail: shijs@ynnu.edu.cn)

core, which can not only accelerate CNN computing, but also improve the speed of building CNN application system.

A. Convolution module design

Convolution layer is the most computationally intensive part of CNN, which is used to extract image features. The specific operation is to convolve the input feature map with the convolution kernel, weight the summation of the convolution results, and then obtain the feature map of this layer after processing by the activation function, As shown in formula (1).

$$y_j^l = f\left(\sum_{i\in \mathrm{M}_j} x_i^{l-1} * k_{ij}^l + b_j^l\right) \tag{1}$$

In formula (1), y_j^l represents the feature map of the j-th convolution kernel convolution result of the l-th layer, and M_{j} represents the selection of the previous input feature map by the current convolution. x_i^{l-1} represents the previous input feature map, and k_{ij}^l represents the i-th weighting coefficient of the j-th convolution kernel of the l-th layer. b_j^l represents the bias parameter of the j-th convolution kernel of the l-th layer, and f is a non-linear activation function.

Convolution involves a 2-dimensional multiply-add calculation (MAC). In mathematical form, it can be regarded as consisting of several multiply-accumulate operations. In order to realize convolution operation, various structures are developed [10]. For example, reference [11] is a classical convolution hardware implementation method, and the 3 × 3 convolution circuit shown in Fig. 1. This structure makes full use of the parallelism of the convolution operation. By setting the row cache structure, the convolution window sliding of the image is realized, and a convolution calculation circuit of efficient parallel pipeline operation is formed. After the input feature map data fills the entire pipeline, each clock cycle is equivalent to completing a 3 × 3 convolution window sliding and outputting a convolution result. It is equivalent to completing 9 multiplication operations and 8 addition operations in one clock cycle, which achieves a 17 times speedup compared to a serial computing processor. The larger the size of the convolution kernel, the more obvious the acceleration effect. However, this structure is only for fixed-size convolutions, and it is difficult to adapt to different convolution methods of multiple convolutional layers in actual CNNs, which is not flexible enough.

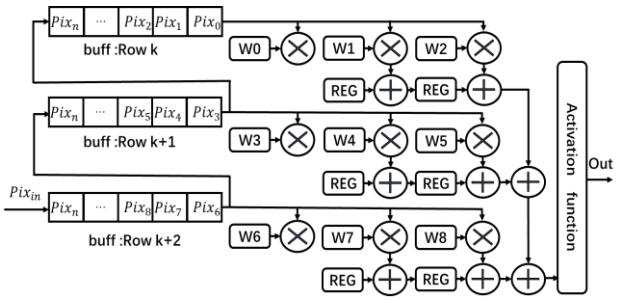

Fig. 1. 3×3 convolution calculation circuit.

In this paper, a reconfigurable convolution module is used to design the structure, which can be dynamically adjusted into different convolution modules by changing the parameters. This paper analyzes the loop nesting operation of CNN's 3-D convolution computation, and its parallel characteristics can be divided into four types: inter layer parallelism, inter output parallelism, inter convolution kernel parallelism and inter convolution kernel parallelism. Four kinds of parallel features can be composed of four parallel acceleration structures: single input feature map single convolution kernel, single input feature map multi convolution kernel, multi input feature map single convolution kernel, and multi input feature map multi convolution kernel.

The size of the input feature map is represented by $X_{\mathrm{i}}\times Y_i$, and N_i represents the number of channels of the input feature map. The size of the convolution kernel is represented by $K_s\times K_{\mathrm{s}}$, and K_n represents the number of convolution kernels. The size of the output feature map is represented by $P_{\mathrm{x}}\times P_y$, and P_n represents the number of channels of the output feature map. By setting these parameters, the convolution modules of different structures can be reconstructed. The following four parallel acceleration structures are analyzed. Fig. 2(a) shows the structure of the single input feature map single convolution kernel, N_i=1, K_n=1, which can be fully expanded by using the pulse array calculation structure. The hardware calculation circuit design is similar to Fig. 1, and only the array size needs to be changed according to the size of the convolution kernel. $Pix_n=K_S^2$, The number of multipliers it uses is $\mathrm{mult}_n=K_S^2$, and the number of adders is $add_n=(K_n^2-1)$. This structure is the basic hardware circuit structure of convolution parallel acceleration operation, which can be used to build several other parallel acceleration structures. Fig. 2(b) shows the structure of the single input feature map multi convolution kernel, N_i=1. Expand the structure, the number of multipliers used in the structure is $\mathrm{mult}_n=K_S^2K_n^2$, and the number of adders is $add_n=(K_n^2-1)K_n$. Fig. 2(c) shows the structure of the multi input feature map single convolution kernel, K_n=1. Expand the structure, the number of multipliers used in the structure is $\mathrm{mult}_n=K_S^2N_i$, and the number of adders is $add_n=K_s^2N_i-1$. Fig. 2(d) shows the structure of the multi input feature map multi convolution kernel. Expand the structure, the number of multipliers used in the structure is $\mathrm{mult}_n=K_S^2N_iK_n$, and the number of adders is $add_n=(K_s^2N_i-1)K_{\mathrm{n}}$. If the entire convolution operation is unrolled and an input feature map of size $X_{\mathrm{i}}\times Y_i$ is calculated, it only takes $X_{\mathrm{i}}\times Y_i$ clock cycles, including the time to fill the pipeline. Full expansion can improve the computing speed, but it also consumes more hardware resources and requires higher bandwidth.

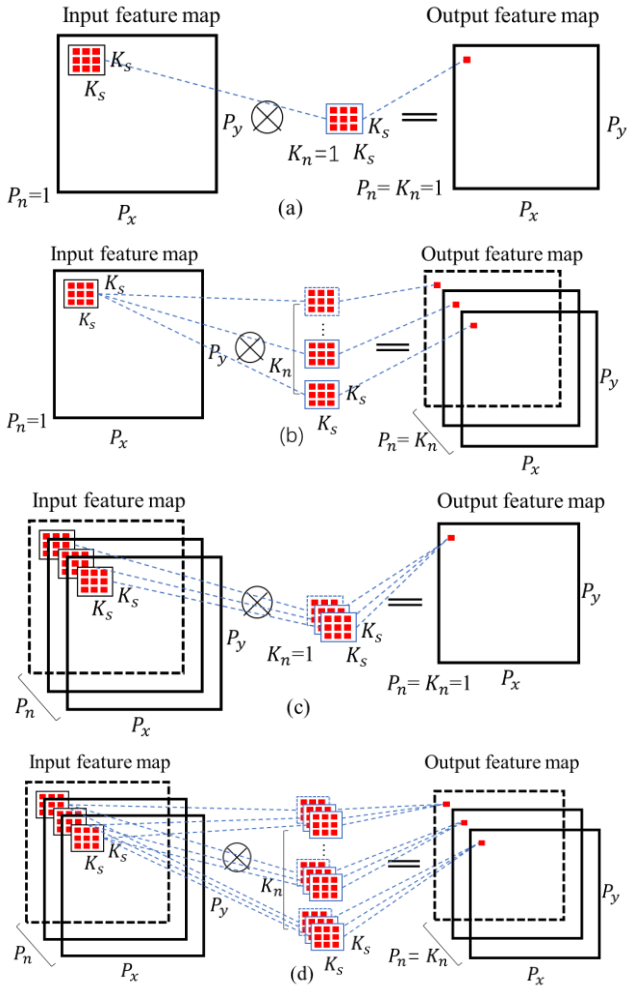

Fig. 2. Convolution parallel acceleration structure.

How to balance the computing time and hardware resource consumption of convolution acceleration is the key to design the convolution layer. If the FPGA's on-chip resources are rich enough and the structure of the convolution layer is not too complicated, the method shown in Fig. 2(d) is used to fully expand to obtain a very good speedup. However, when deploying a more complex network like VGG16 CNN, it is unrealistic to expand it all. In general, the fig. 2(d) is divided into several fig. 2(c), which can be composed of fig. 2(a) and fig. 2(b). Evaluate CNN's convolution structure and FPGA's on-chip resources, set up a reasonable parallel structure, and reduce the consumption of hardware resources by reusing the convolution calculation circuit multiple times.

B. Pooling module design

The pooling layer is usually located behind the convolution module, and is usually used to reduce the size of the input feature map. The pooling module generally has two processing methods, maximum pooling and average pooling. The maximum value and average value of the sampling area are used as the output data of the sampling area, and the size of the sampling area is generally 2×2. The design of the 2×2 pooling module is shown in Fig. 3 The pooling mode of the pooling module can be configured through the multiplexer. The maximum pooling module is implemented with multiple parallel comparators and data selectors. Every two data in the sampling area are divided into a group for comparison, and the maximum value of each group is compared until the maximum value in the sampling area is obtained. The average pooling module is similar to the convolution operation. The values in the sampling area are summed by the parallel addition tree. Because the number of data in the sampling area is an integer power of 2, the shift operation can be used instead of the division operation. The pooling module is inserted directly after each convolutional module.

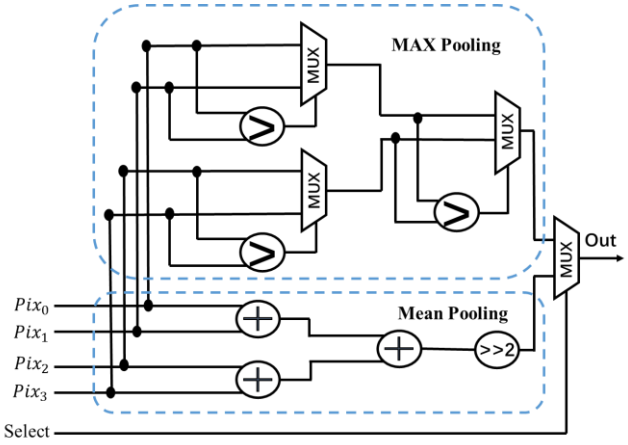

Fig. 3. Pooling circuit.

C. Fully connected module design

The essence of the calculation of fully connected layers (FC layers) is the multiply-accumulate operation, which is characterized by a small amount of calculation but requires a large number of weight parameters, and each operation requires access to memory. Therefore, it is unrealistic to fully expand the FC layer parallel computing. This article uses a locally parallel acceleration strategy, assuming that the output feature map size of the previous layer of the FC layer is $P_x \times P_y$, P_n is the number of channels, and the number of neurons in the FC layer is M_n. As shown in Fig. 4, if the feature map output from the previous layer is converted into a one-dimensional vector, the total length of the one-dimensional vector $L_N = P_x P_y P_n$, and the length of each channel $L_n = P_x P_y$. If serial multiply-accumulate is used to calculate the FC layer, $(P_x P_y P_n+1)M_n$ clock cycles are required, hardware resources are consumed less, but the delay is long. Due to the natural parallelism between the channels of the output feature map of the upper layer of the FC layer, a local acceleration strategy is adopted. The calculation operation of the entire FC layer is divided into M_n batches, and each batch uses the parallelism calculation of P_n to multiply and accumulate operations. The hardware structure is shown in Fig. 5.

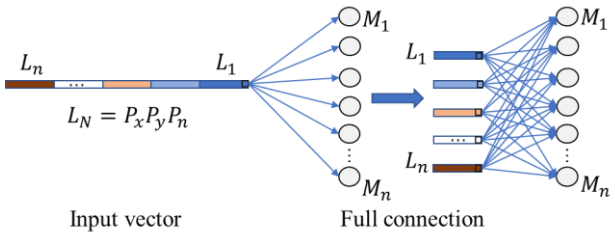

Fig. 4. FC layer local acceleration structure.

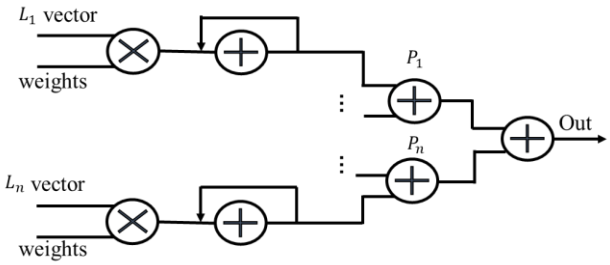

Fig. 5. FC layer calculation circuit.

The calculation of the FC layer requires a large number of weight parameters. Even a small CNN such as LeNet-5, it is unrealistic to save its weight parameters in on-chip memory and must use off-chip memory. The throughput of the FC layer mainly depends on the speed of the external memory to the on-chip cache area. As shown in Fig. 6, an on-chip double-buffering weight parameter method is used. First, the weight parameters of the next batch to be calculated are written into the Buff2 on-chip cache. Buff2 includes P_n on-chip RAMs, and the weight parameters of each RAM can be read in parallel during calculation. When calculating the current batch, read the prepared weight parameters from Buff1. Ping-Pong switching operation is adopted in the two on-chip caches to reduce the waiting time of FC layer calculation. Multiple batches reuse the FC layer calculation circuit, and save the calculation results of each batch in the on-chip cache (fifo), which can complete the acceleration of the entire fully connected layer.

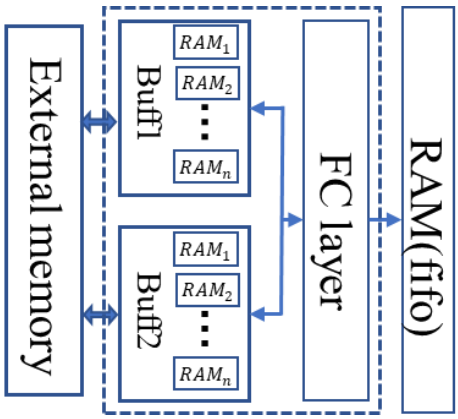

Fig. 6. FC layer calculation circuit.

III. FPGA IMPLEMENTS FOR HANDWRITTEN DIGIT RECOGNITION

Handwritten digit recognition is a typical application on CNN. This section mainly introduces building a CNN network on a PC-side platform for training and testing. Finally, the constructed handwritten digit recognition network is deployed on the FPGA to complete the process of forward propagation inference.

A. Network structure design

Firstly, we use the deep learning framework keras to build CNN for handwritten numeral recognition. Its network structure is improved on the lenet-5 network, as shown in Fig.7. The network consists of two convolutional layers, two pooling layers, a FC layer, and an output layer. The convolution layer uses the Relu activation function, which is defined as $y = \max(x, 0)$, and the window sliding step is 1. The pooling layer uses maximum pooling. The network was trained using the MNIST handwritten digit dataset, which was trained using 60,000 pictures, 10,000 pictures for testing, and the weight data was saved locally [12].

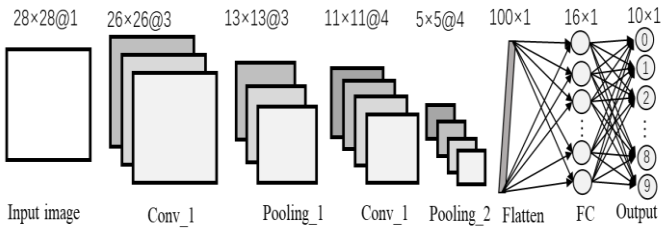

Fig. 7. CNN structure.

B. FPGA architecture design

A reconfigurable convolutional module is used to quickly build the convolutional layer. As shown in Fig. 7, the structure of the convolutional layer in a CNN that implements handwritten numeral recognition is relatively simple, and the size and number of convolution kernels are small. The structure of the Conv_1 layer is similar to Fig. 2(b), which is fully expanded. The set parameters are $X_i = Y_i = 28$, N_i=1, $K_s = 3$, K_n=3, and three parallel feature maps are output. The structure of the Conv_2 layer is similar to Fig. 2(d). The set parameters are $X_i = Y_i = 13$, N_i=3, $K_s = 3$, K_n=4, and four parallel feature maps are output. The acceleration ratio of CNN convolution is improved by fully expanding operation. In addition, the output of convolution module realizes 2×2 window sliding through line buffer, which directly connects the input of pooling module. Because the output feature map of Pooling_2 layer is $P_x = P_y = 5$, P_n=4. The number of neurons in the FC layer is $M_n = 16$. Therefore, the processing of FC layer is divided into 16 batches, each batch is set with 4 parallelism. Using the structure described in Fig. 6, the on-chip double buffer uses ping-pong operation to read and write weight parameters, and finally sets the on-chip buffer (fifo) to save the calculation results of each batch. The calculation result of each batch is the output result of each FC layer neuron.

IV. EXPERIMENTAL RESULTS

The implementation of the CNN module and the construction of the handwritten digit recognition system are implemented using RTL coding. Compared with a high-level synthesis (e.g. HLS, OpenCL) platform, it has better hardware implementation efficiency and stricter timing control. The handwritten digit recognition system is constructed using reconfigurable modules, which greatly improves the efficiency of system deployment. The entire system has been implemented on the lntel Cyclone10 FPGA platform. Compared with other references [6,8,13], the CNN structure of this paper is more streamlined than LeNet-5, and the convolution calculation circuit is fully expanded, so it can obtain a better speedup. In the experiment, 10,000 pictures of the MNIST test set were used as input feature maps of the CNN, and the classification results were statistically calculated.

TABLE I. PERFORMANCE COMPARISON OF HANDWRITTEN DIGIT RECOGNITION FPGA

Reference	[6]	[8]	[13]	This work
FPGA	Virttex-7 Xc7vx485	Artix-7 XC7A100	Zynq-ZC702	Cyclone 10
Frequency	150M	300	166M	150M
Time/ms	0.0254	0.041	0.151	0.0176
DSP	638	0	95	274
FF	66346	106400	27664	48765
LUT	51125	15769	388361	12588
Accuracy	96.8	90	99	97.57

In order to accelerate the forward inference process of the CNN network, all weight parameters and calculation processes are represented by fixed-point data, which can reduce the occupation of FPGA hardware resources. In the implementation of handwritten digit recognition experiments, all data are represented by 18 fixed-point decimal places, with the highest digit being the sign bit, 4 integer digits, and 13 decimal places. The benefits of using fixed-point numbers can also be used to build multipliers using LUTs, saving DSPs resources on the FPGA. Due to CNN does not need the same high parameter accuracy in the forward inference phase as in the back-propagation phase, the classification accuracy of the entire handwritten digital network does not change significantly after the data is fixed. Classify and identify the MNIST test set, and its confusion matrix is shown in Fig. 8

		Detected Class									
		0	1	2	3	4	5	6	7	8	9
Target Class	0	99.1%	0%	0.1%	0.1%	0.1%	0%	0.2%	0.1%	0.3%	0%
	1	0%	99.03%	0.4%	0%	0%	0.1%	0.2%	0%	0.4%	0%
	2	0.6%	0%	97.8%	0.1%	0.1%	0%	0.2%	0.4%	0.9%	0%
	3	0%	0%	0.4%	98.3%	0%	0%	0%	0.3%	0.6%	0.4%
	4	0.2%	0%	0.4%	0.1%	97.8%	0%	0.4%	0%	0.3%	0.8%
	5	0.3%	0%	0%	1.9%	0.1%	95.2%	0.7%	0.2%	1.1%	0.3%
	6	0.5%	0.3%	0.2%	0.1%	0.3%	0.3%	98.1%	0%	0.1%	0%
	7	0%	0.5%	1.3%	0.7%	0.1%	0%	0%	96.1%	0.5%	0.9%
	8	0.5%	0%	0.3%	0.6%	0.1%	0.1%	0.1%	0.2%	97.9%	0.1%
	9	0.4%	0.5%	0%	1%	0.8%	0.2%	0%	0.4%	0.5%	96.2%

Fig. 8. Confusion matrix.

V. CONCLUSIONS

In this paper, RTL coding is used to accelerate CNN on FPGA platform, and the basic modules of CNN can be reconstructed. A handwritten digit recognition system based on CNN was built on the FPGA platform and implemented on the lntel Cyclone10 platform. After the fixed-point processing, the recognition rate of the MNIST dataset reached 97.57%. Because the system is built using reconfigurable IP cores, CNNs with different network architectures can be easily built and can be extended on different FPGAs.

ACKNOWLEDGMENT

This work is funded by grants from the National Science Foundation of China (grant number 61875171) and the Yunnan Education Commission of China (grant number ZD2014004).

REFERENCES

[1] Ma. Y, Cao. Y, Vrudhula. S, Seo. J. S, "Optimizing the convolution operation to accelerate deep neural networks on FPGA." IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2018, 26(7): 1354-1367.

[2] Krizhevsky. Alex, Ilya. Sutskever, Geoffrey. E. Hinton, "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[3] Lin. Min, Qiang. Chen, Shuicheng. Yan, "Network in network." arXiv preprint arXiv: 1312.4400, 2013.

[4] Qiu. J, Wang. J, Yao. S, Guo. K, Li. B, Zhou. E, Wang. Y, "Going deeper with embedded fpga platform for convolutional neural network." Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2016.

[5] Guan. Y, Liang. H, Xu. N, Wang. W, Shi. S, Chen. X, Cong. J ,"FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates." 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, 2017.

[6] Zhou. Yongmei, Jingfei. Jiang, "An FPGA-based accelerator implementation for deep convolutional neural networks." 2015 4th International Conference on Computer Science and Network Technology (ICCSNT). Vol. 1. IEEE, 2015.

[7] Lu. L, Liang. Y, Xiao. Q, Yan. S, "Evaluating fast algorithms for convolutional neural networks on FPGAs." 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, 2017.

[8] Giardino. D, Matta. M, Silvestri. F, Spanò. S, Trobiani. V, "FPGA implementation of hand-written number recognition based on CNN." International Journal on Advanced Science, Engineering and Information Technology, 2019, 9(1): 167-171.

[9] Mittal. Sparsh, "A survey of FPGA-based accelerators for convolutional neural networks." Neural computing and applications , 2018: 1-31.

[10] Du. Z, Fasthuber. R, Chen. T, Ienne. P, Li. L, Luo. T, Temam. O, "ShiDianNao: Shifting vision processing closer to the sensor." ACM SIGARCH Computer Architecture News. Vol. 43. No. 3. ACM, 2015.

[11] Farabet. C, Poulet. C, Han. J. Y, LeCun. Y, "Cnp: An fpga-based processor for convolutional networks." 2009 International Conference on Field Programmable Logic and Applications. IEEE, 2009.

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

[13] Feng. G, Hu. Z, Chen. S, Wu. F, "Energy-efficient and high-throughput FPGA-based accelerator for Convolutional Neural Networks." 2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT). IEEE, 2016.