A novel FPGA-based convolution accelerator for addernet

Ke Ma, Lei Mao

School of Information, Production and System
Waseda University
Fukuoka, Japan
woshimark666@fuji.waseda.jp, maolei@fuji.waseda.jp

Shinji Kimura

School of Information, Production and System
Waseda University
Fukuoka, Japan
shinji_kimura@waseda.jp

Abstract— In FPGA-based CNN accelerators, most of the multiplications in convolutions are realized by DSPs. Thus, the number of DSPs limits the parallelism of convolution computation. Recently proposed addernet replaces multiplications in convolution with additions. Based on addernet, we designed a novel PE in which the adders can be reused to perform replaced additions and accumulation. This PE can calculate a 3*3 convolution in 3 clocks using only 9 adders, and can be efficiently constructed on LUT with no DSP. On a Ultra-96 board which has 360 DSPs, we implement an accelerator with 60 proposed 3*3 PEs, and gain a throughput of 2.18 GOPs.

I. Introduction

Convolution Neural Networks (CNNs) have demonstrated significant success in many applications such as image classification [2], detection [3] and segmentation [4]. Figure 1 shows the basic structure of a CNN net. In recent years, field-programmable gate arrays (FPGAs) are becoming the platform for accelerating the inference phase of CNN. FPGA-based CNN accelerators can achieve much less power consumption and low latency compared with GPU; they also more flexible compared with ASIC-based accelerators. Many researches have been done to optimize FPGA-based CNN accelerators. In [8] and [9], they proposed new algorithms which decrease total computation capacity in CNN. [5] [6] and [7] proposed CNNs with quantized inputs and weights which are more hardware-friendly. [10] [11] and [12] optimize the data-path of CNN accelerators to improve throughput of the accelerators.

Among the modern CNN models such as LeNet-5 [14], AlexNet [15], ResNet [16], VggNet [17], convolution layers are usually very compute-intensive; they consume 4.31% of the total weights but occupy more than 93.2% of the total arithmetic operations [13]. So it stands to reason that many FPGA-based accelerators [18] focus on optimizing convolution layers.

Mapping convolution layers into FPGA costs a great amount of FPGA resources such as LUTs and DSPs. DSP resources are usually very critical on many FPGA boards while they play an important role in MAC operations (multiply-accumulate operation). The number of DSP components limits the parallel computing of convolution in most of the cases. Recently, Hanting Chen et al. proposed a novel CNN algorithm called AdderNet [1]. In AdderNet, they replace all the multiplications in convolution operations with additions. By dedicated training, addernet still maintains high classification accuracies on imagenet [19], cifar-10 and cifar100 [20] datasets.

In this paper, in order to further enhance the performance of FPGA-based CNN accelerators, by exploiting the feature of addernet, we proposed a novel convolution layer accelerator for FPGA and achieved higher throughput compared with traditional accelerators. In summary, we made following contributions in this paper:

- We quantize addernet from 32-float format into 16-fixed point format for both input and weighs and still maintain a decent accuracies.
- We propose a novel processing elements for addernet which has a low latency and low resource utilization.
- Based on the PE, we build an accelerator for convolution layers in addernet on a Xilinx UltraScale+ MPSoC ZU3EG A484 board.

II. Background

In this section, a review of background knowledge of convolution layers and addernet is presented. We also show some basic PE designs in most of the FPGA-based CNN accelerators.

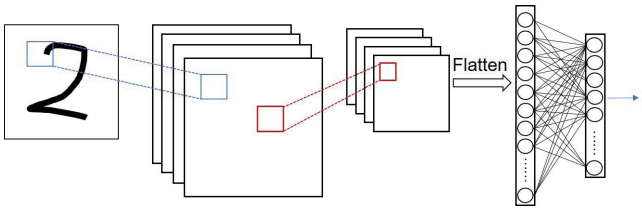

Fig. 1. The basic structure of a CNN net.

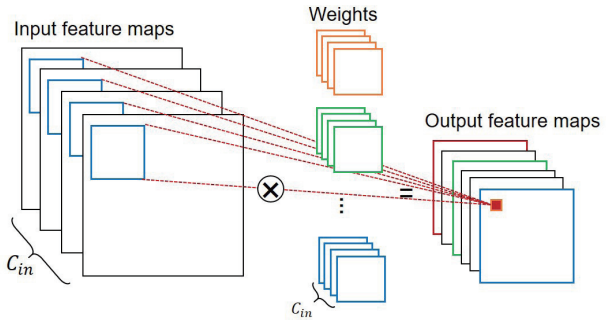

Fig. 2. The basic operation in convolution layer.

A. *Convolutional neural network*

The inference phase of CNN can extract features in an image and perform classification tasks. The convolution layers and pooling layers are main layers in extracting features while fully connect (FC) layers realize classification. Since convolution (conv) layers contribute most computation capacity in CNNs, we will focus on introducing and optimizing conv layers. Figure 2 demonstrates the operations in convolution layers.

Typically, convolution operations apply one or several filters to feature maps to extract features. Assuming $F(i, j, k, n)$ represents the weight in position (i, j, k) from the $t - th$ filter. So the output $Y(m, n, t)$ can be formulated as:

$$Y(m,n,t) = \sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{C_{in}} X(m+i, n+j, k) \times F(i,j,k,t) \tag{1}$$

the number d denotes the kernel size of $d \times d$ and C_{in} represents the number of channels of both input feature map X and filter F. By sliding through the whole input feature map using different filter kernels, the output of one conv layer can be obtained.

B. *Addernet*

In CVPR 2020, Hanting Chen et al. proposed a novel algorithm in which they replace multiplications in conventional conv layers with additions. They reformulate the

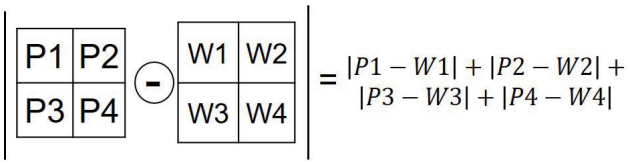

Fig. 3. A sample of 2×2 convolution in addernet.

TABLE I
CLASSIFICATION RESULTS ON THE CIFAR-10 AND CIFAR-100 DATASETS.

Model	Method	CIFAR-10	CIFAR-100
ResNet-20	BNN [5]	84.87%	54.14%
	Addernet	91.84%	67.60%
	CNN	93.80%	68.14%
ResNet-32	BNN [5]	86.74%	56.21%
	Addernet	93.01%	69.02%
	CNN	93.29%	69.74%

Eq. (1) into:

$$Y(m,n,t) = -\sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{C_{in}} |X(m+i, n+j, k) - F(i,j,k,t)| \tag{2}$$

Figure 3 demonstrates a basic 2×2 convolution in addernet. It is an element-wise subtraction and accumulation of the subtraction results. The theory behind it is that l_1 distance can be used to measure the similarity between filters and input feature maps. By their dedicated training method, they received decent accuracies on different datasets using different network structures. Table I demonstrates the classification accuracies of addernet compared with normal CNNs under float 32 data format.

C. *FPGA accelerator and PE*

Parallel computing is the technique which most of these accelerators make use of. Figure 4 shows the general design of PE and data path of these accelerators. Part of the input feature maps and weights are loaded into the buffers on FPGA through a direct-memory-access (DMA) from external memory. Then the input pixels from feature maps and weights are streamed into the accelerator for calculation. After calculation, output data are stored in another buffer for output data before they are streamed back to external memory through DMA. The accelerators are usually composed of a number of basic PEs which perform basic MAC operations in parallel. For different designs, PEs are different as shown in Figure 5. Figure 5(a) is a sequential circuit for MAC operation and (b) is composed of multiple multipliers followed by an adder tree. Obviously, (a) costs less FPGA resources but takes more

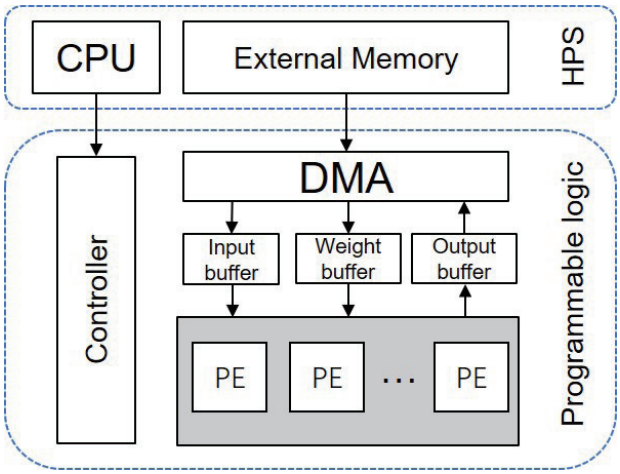

Fig. 4. General architecture of a FPGA CNN accelerator.

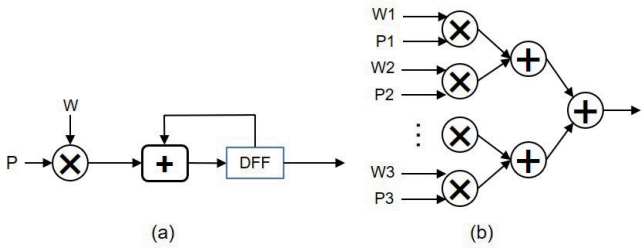

Fig. 5. Two PE designs for 2×2 convolution: (a)Sequential MAC. (b)Multiplier followed by adder-tree. W denotes input weights and I denotes input pixels

clock cycles to run while (b) costs much more resources but runs faster. Note both (a) and (b) require DSP resources to perform multiplications. Because of this, the number of PEs one FPGA board can deploy are limited and the throughput of the accelerator is limited. In order to break through this limitation, by exploiting addernet, we propose a novel FPGA-based convolution accelerator for addernet.

III. Proposed accelerator for addernet

A. Low precision addernet

Since the addernet is a new algorithm which only trained and verified on GPUs under float 32 data format, before implementing the CNN on hardware, we verified the addernet under 16 fixed-point data format using the quantization library [21] for Pytorch. By quantizing all the input pixels and all the weights to 16 fixed point number, we obtained the following accuracies on CIFAR-10 and CIFAR-100 using Resnet as shown in Table II. The addernet still remains decent accuracies under low precision data format.

TABLE II
Accuracies of Addernet under float and fixed point data format

Model	Data format	CIFAR-10	CIFAR-100
Resnet-20	32-float	91.58%	66.65%
	16-fixed	91.35%	66.35%
Resnet-32	32-float	92.48%	69.50%
	16-fixed	90.86%	69.21%

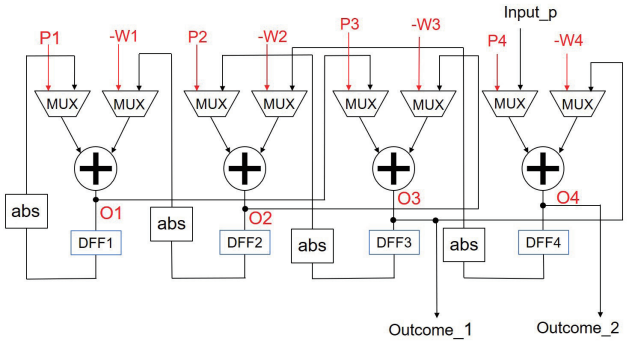

Fig. 6. Proposed PE basic4. Red arrows represents input weights with corresponding input pixels. Select signals of the MUXs and clock signals of DFFs are omitted

B. Proposed PE

It seems to be an easy job if we replace the multipliers in Figure 5(a) and 5(b) with adders. However, with (a) being too slow and (b) being expensive in resource utilization, we decide to come up with a different PE as shown in Figure 6. This PE is capable of performing a 2×2 add convolution so we name it as basic4 PE in the following.

In the beginning, the select signal of these multiplexers (MUXs) are set to 0 so that input pixels and weights can be streamed into four adders. Note weights are preprocessed to be their own negative. After one positive edge of clock, the added numbers are pushed into D Flip-Flops. Then, the select signal sets to 1 so that adders are recombined into a adder tree to sum the numbers from DFF. Figure 7 demonstrates two structures for addition (a) and accumulation (b). Table III shows how signal changes for the basic4 PE. Finally, the output can be obtained at port outcome_1 . In outcome_2 port, we can obtain $Input_p + Outcome_1$ if there is a extra input.

Considering the latency of the MUXs, abs and adders, the final output is not stable immediately after the second positive edge clock. So,the final output will not be fetched until the third positive edge of clock. Note that we use a controlled clock signal at the clock ports for DFFs. So the new data will not be pushed into DFF when the third positive edge of clock comes.

By adding additional adders and more basic4 components, convolution kernels with different sizes can be realized. We use two basic4 PEs with one additional adder to

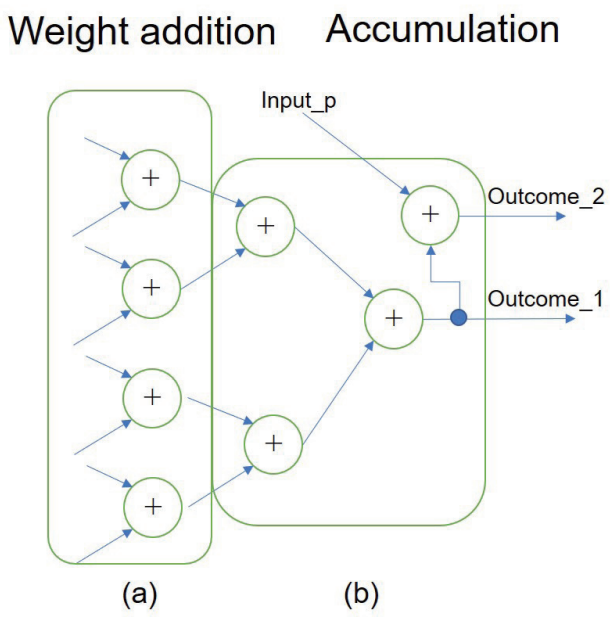

Fig. 7. Two structures of basic4 PE: (a) when Sel=0, addition of pixels and weights, (b) when Sel=1, accumulation of additions (abs module is omitted).

TABLE III
SIGNAL CHANGES IN PROPOSED BASIC4 PE

CLK	Sel	$O1$	$O2$	$O3$	$O4$
0	0	$s1 = P1 - W1$	$s2 = P2 - W2$	$s3 = P3 - W3$	$s4 = P4 - W4$
1	1	$s12 = \|s1\| + \|s2\|$	$s34 = \|s3\| + \|s4\|$	$s12 + s34$	$s12 + s34 + Input\_p$

combine a new PE for 3×3 convolution kernels basic9 as shown in Figure 8. By connecting the Input_p port with outcome_2 port, we can easily sum the result from other PEs by reusing the fourth adder of basic4 PE.

In summary, 4 adders perform subtraction when Sel=0 and when Sel=1, 3 of 4 adders are mapped to a 4-input adder tree with one being used for extra accumulation for 3×3 kernel. By reusing adders, the proposed design can save much resources which can be seen from Table IV. This gives a potential to implement more PEs on board to increase parallelism.

C. Convolution accelerator for addernet

Figure 9 demonstrate the architecture of proposed convolution accelerator. We duplicate a nunber of basic9 PEs into a static systolic array. Weight buffer stores the weights of different filters; input buffer stores the input feature map pixels and output buffer stores the outputs. Data router fetches the pixels and stream them into PEs. Note that the pixel data and weight data that streamed

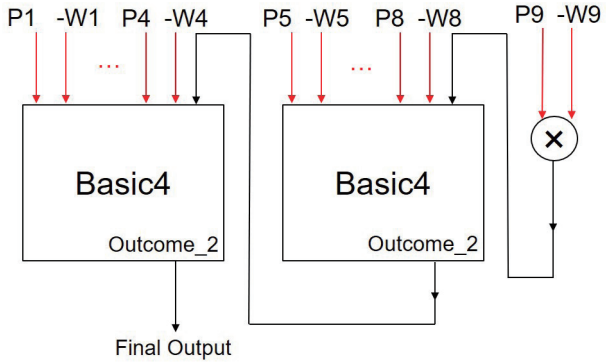

Fig. 8. Basic9 PE for a convolution with 3×3 kernel.

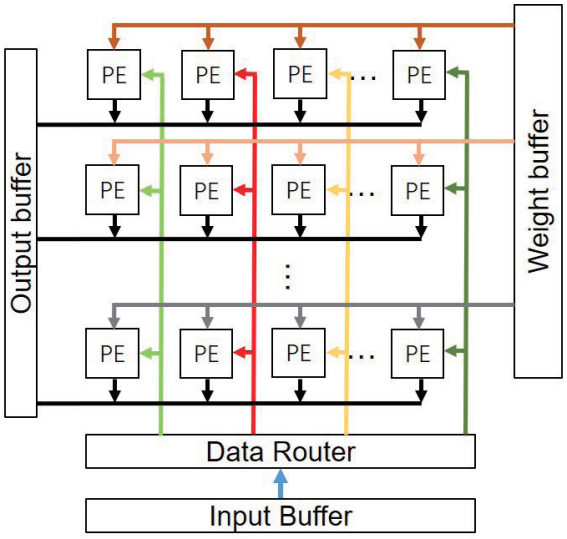

Fig. 9. The architecture of proposed convolution accelerator for addernet.

out are being reused by multiple PEs. For different FPGA boards, the size of the buffer and the number of PEs implemented can be different. In our implementation, 60 PEs are implemented and 2 weight data lines are being shared by these PEs.

IV. IMPLEMENTATION RESULTS

We design the proposed PE and accelerator using Vivado 2020 and implement them on a ultra-96 board which is composed of an Arm-core and a Xilinx Zynq UltraScale+ MPSoC ZU3EG A484 programmable logic.

A. Processing element

Before diving into the accelerator design, we first design and implement 3 kinds of PEs which are A) multipliers followed by adder tree like Figure 5(b), B) switch multipliers in Figure 5(b) with adders and C) our proposed

TABLE IV
Resource utilization and latency of three PEs

PE	LUT	Filp-flop	CARRY8	DSP	Latency
A	1419	0	119	0	2 clocks
B	706	0	56	9	3 clocks
C	809	120	63	0	3 clocks

basic9 PE in Figure 8. All of these PE are designed for a 3×3 kernel. Table IV shows the resource utilization of these PEs. Compared with design A, proposed PE uses much fewer LUT resources and compared with design C, proposed PE uses no DSP resources.

B. The accelerator

We then implement the proposed accelerator on the ultra-96 board. Considering the limited resources, we set 60 basic9 PEs and pull out 2 weights data buses from weights buffer as shown in Figure 9. For data flow design, we followed the architecture in Figure 4. The bandwidth between external memory and DMA is 128-bit wide. The whole design consumes 66.6k (92.9%) LUTs, 17.8k (12.65%) flip-flops and 4.87k (53.57%) CARRY8. The total number of DSPs on board is 360, so theoretically the maximum parallelism for a single 3×3 convolution is $360 \div 9 = 40$. And the proposed design expands the parallelism from 40 to 60 and also remains a decent throughput.

C. Evaluation result

In order to evaluate the performance of proposed design, we use a $32 \times 32 \times 10$ input feature map convolving with a $3 \times 3 \times 20$ weight kernel with stride 1 as the target task. We picked up some CNN FPGA-implementations and evaluate their convolution accelerating modules and the comparison results are shown in Table V. Parallelism denotes the maximum multiplications the accelerator can perform at the same time. Generally, the throughput increases with parallelism. However, due to different dataflow designs, the throughput will not necessarily increase when parallelism increases like design [22]. For design [18], they increase the parallelism to 1176 by implementing LUT-based MAC unit. Obviously, implementing multiplication uses more LUTs compared with implementing LUT-based additions. Since DSP is not used in our design, there is a potential to increase parallelism by implementing DSP-based addition.

TABLE V
Comparison with previous CNN fpga implementations

	[22]	[23]	[18]	Proposed
Board	Zedborad	Altera Cyclone V	Stratix V GXA7	Ultra-96
Data format	16 fixed	16 fixed	16 fixed	16 fixed
Frequency (MHz)	100	100	150	100
DSP	64 (29.09%)	28 (32.18%)	256 (100%)	0
LUT	28.9k (54.25%)	11k (34.43%)	176k (75%)	66.6k (94.38%)
FF	41.8k (39.31%)	N/A	N/A	17.8k (12.61%)
Parallelism	64	32	1176	540
Throughput (GOPs)	2.09	12.11	121.31	2.18

V. Conclusions

In this paper, we test the accuracy of recently proposed addernet with-16 bit fixed point number data and proposed a novel PE with no DSP for addernet which has a low latency and low resource utilization. Based on the PE with no DSP, we designed our own convolution accelerator and gained up to a throughput of 2.18 GOPs for convolution. As a future work, we will do the combination of DSPs and our LUT based PEs to gain higher performance.

References

[1] Chen, Hanting and Wang, Yunhe and Xu, Chunjing and Shi, Boxin and Xu, Chao and Tian, Qi and Xu, Chang, "AdderNet: Do we really need multiplications in deep learning?" *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1468-1477, 2020.

[2] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, and others, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, Vol. 115(3), pp. 211-252, 2015.

[3] Ross Girshick, "Fast R-CNN," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1440-1448, 2015.

[4] Jonathan Long, Evan Shelhamer, and Trevor Darrell, "Fully Convolutional Networks for Semantic Segmentation," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, 2015.

[5] Courbariaux, Matthieu and Hubara, Itay and Soudry, Daniel and El-Yaniv, Ran and Bengio, Yoshua, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint*, arXiv:1602.02830, 2016.

[6] Li, Fengfu and Zhang, Bo and Liu, Bin, "Ternary weight networks," *arXiv preprint*, arXiv:1605.04711, 2016.

[7] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," *Proceedings of the European Conference on Computer Vision*, pp. 525-542, 2016.

[8] Andrew Lavin and Scott Gray, "Fast Algorithms for Convolutional Neural Networks," *arXiv preprint*, arXiv:150, 2015.

[9] Chi Zhang and Viktor Prasanna, "Frequency Domain Acceleration of Convolutional Neural Networks on CPU-FPGA Shared Memory System," *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 35-44, 2017

[10] Murugan Sankaradas, Venkata Jakkula, Srihari Cadambi, Srimat Chakradhar, Igor Durdanovic, Eric Cosatto and Hans Peter Graf, "A Massively Parallel Coprocessor for Convolutional Neural Networks," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 53-60, 2009.

[11] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi, "A Dynamically Configurable Coprocessor for Convolutional Neural Networks," *ACM SIGARCH Computer Architecture News*, Vol. 38(3), pp. 247–257, 2010.

[12] Vinayak Gokhale, Jonghoon Jin, Aysegul Dundar, Berin Martini, and Eugenio Culurciello, "A 240 G-ops/s mobile coprocessor for deep neural networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 696–701, 2014.

[13] Zhang, Chen and Wu, Di and Sun, Jiayu and Sun, Guangyu and Luo, Guojie and Cong, Jason, "Energy-efficient CNN implementation on a deeply pipelined FPGA cluster," *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pp. 326-331, 2016.

[14] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, Vol. 86, pp. 2278-2324, 1998.

[15] Krizhevsky, Alex and Sutskever, Ilya and Hinton, Geoffrey E, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, pp. 84-90, 2017.

[16] Simonyan, Karen and Zisserman, Andrew, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint* , arXiv:1409.1556, 2014.

[17] Simonyan, Karen and Zisserman, Andrew, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint* , arXiv:1409.1556, 2014.

[18] Y. Ma, Y. Cao, S. Vrudhula and J. Seo, "Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 26, pp. 1354-1367, 2018.

[19] Deng, J. and Dong, W. and Socher, R. and Li, L.-J. and Li, K. and Fei-Fei, L, "ImageNet: A Large-Scale Hierarchical Image Database," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[20] Krizhevsky, Alex and Hinton, Geoffrey and others, "Learning multiple layers of features from tiny images," *Citeseer*, 2009.

[21] Tianyi Zhang, Zhiqiu Lin, Guandao Yang, Christopher De Sa, "QPyTorch: A Low-Precision Arithmetic Simulation Framework," *arXiv preprint*, arXiv:1910.04540, 2019.

[22] Y. Cao, X. Wei, T. Qiao and H. Chen, "QFPGA-based accelerator for convolution operations," *2019 IEEE International Conference on Signal, Information and Data Processing (ICSIDP)*, pp 1-5, 2019.

[23] Motamedi, Mohammad and Gysel, Philipp and Ghiasi, Soheil, "PLACID: A platform for FPGA-based accelerator creation for DCNNs," *ACM Transactions on Multimedia Computing, Communications, and Applications*, Vol. 13, pp 1-21, 2017.