A CNN Accelerator on FPGA Using Depthwise Separable Convolution

Lin Bai, *Student Member, IEEE*, Yiming Zhao, and Xinming Huang, *Senior Member, IEEE*

***Abstract*—Convolutional neural networks (CNNs) have been widely deployed in the fields of computer vision and pattern recognition because of their high accuracy. However, large convolution operations are computing intensive and often require a powerful computing platform such as a graphics processing unit. This makes it difficult to apply CNNs to portable devices. The state-of-the-art CNNs, such as MobileNetV2 and Xception, adopt depthwise separable convolution to replace the standard convolution for embedded platforms, which significantly reduces operations and parameters with only limited loss in accuracy. This highly structured model is very suitable for field-programmable gate array (FPGA) implementation. In this brief, a scalable high performance depthwise separable convolution optimized CNN accelerator is proposed. The accelerator can be fit into an FPGA of different sizes, provided the balancing between hardware resources and processing speed. As an example, MobileNetV2 is implemented on Arria 10 SoC FPGA, and the results show this accelerator can classify each picture from ImageNet in 3.75 ms, which is about 266.6 frames per second. The FPGA design achieves 20x speedup if compared to CPU.**

***Index Terms*—Convolutional neural network, FPGA, hardware accelerator, MobileNetV2.**

I. Introduction

NOWADAYS, convolutional neural networks (CNNs) have become the center of interest, due to their superior performance in tasks ranging from image classification, semantic segmentation, to object detection and tracking. This technique has also been widely used in the industry, such as autonomous driving, video surveillance, speech recognition, etc.

CNN is a computing intensive model. It consumes huge amounts of computing power during training and deployment. In practice, Graphics Processing Units (GPUs) are often selected as the platform. However, GPU's natural of high power consumption limits its application in embedded scenario such as portable devices and wearable systems. Therefore, Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs), as the replacement of GPUs, are adopted in neural network applications [1]–[12]. More specifically, increasing research attention is focused on FPGA-based CNN accelerator due to the possibility of trade-off between power consumption and reconfigurability.

To further lighten the computing burden of standard convolution, depthwise separable convolution is proposed in [13]. This has been applied in MobileNetV1 [14] and later MobileNetV2 [15], and thus achieved comparable results with much less multiply-accumulation operations and parameters.

Almost all the existed FPGA-based CNN implementation works were to explore memory bandwidth and computing parallelism limitations. To conquer the limitation of memory bandwidth, [2] and [3] stored the parameters in on-chip memory. However, as CNN goes deeper, parameters required by convolution increase sharply, which makes the on-chip memory solution inefficient. Other works like [4]–[6] alleviated the pressure on off-chip memory through limiting the parameters precision of the neural networks, as lower numerical precision were proved to be sufficient for CNN [16], [17]. In [7] and [8], computing engine was optimized for highly parallelism in computation. Reference [6] proposed a pipeline based solution for CNN for high throughput. Reference [9] made a comprehensive evaluation and comparison of Altera and Xilinx OpenCL frameworks for CNN. Reference [10] explored the sparsity-based optimizations, which could achieve up to 3x higher core energy efficiency and raise the device-level energy efficiency by around 70% through data compression. Both [11] and [12] implemented separable depthwise convolution with the example MobileNetV1, and achieved processing speed at 7.85ms per image and 231.7 frames per second (fps) respectively.

The key contributions of this brief are:

(1) A high performance CNN hardware accelerator framework is proposed where all layers are processed in a computing unit named matrix multiplication engine.

(2) The utilization of hierarchical memory structure and ping-pong on-chip buffer reduces the bandwidth limitation of off-chip memory.

(3) A methodology for scalable design is proposed, so that this framework can be implemented in various FPGAs, through balancing the on-chip resources and performance.

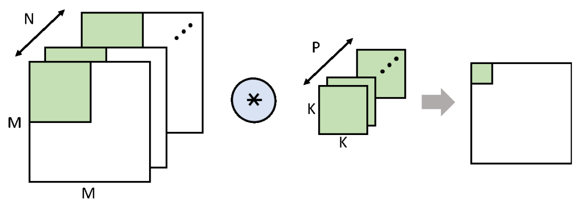
(4) By applying the proposed framework and methods, the state-of-the-art CNN, MobileNetV2 [15], for the first time, is implemented on Arria 10 SoC FPGA. The results show 266.6 frames per second and 170.6 Giga Operations Per Second (GOPS) at system clock frequency of 133MHz. This represents a 20x speedup comparing to that on CPU [15].

Manuscript received March 31, 2018; revised June 13, 2018 and July 17, 2018; accepted July 18, 2018. Date of publication August 17, 2018; date of current version September 27, 2018. This work was supported in part by the U.S. NSF under Grant 1626236, and in part by MathWorks. This brief was recommended by Associate Editor J. M. de la Rosa. *(Corresponding author: Xinming Huang.)*

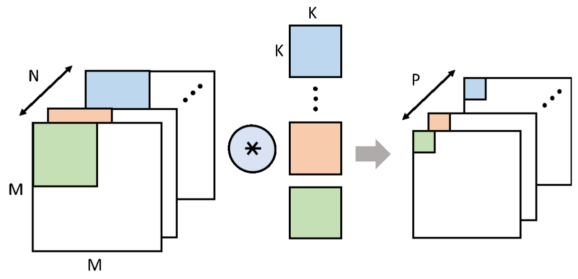
The authors are with the Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, Worcester, MA 01609 USA (e-mail: xhuang@wpi.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

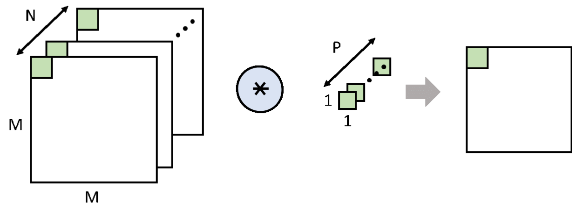
Digital Object Identifier 10.1109/TCSII.2018.2865896



(a) standard convolution



(b) depthwise convolution



(c) pointwise convolution

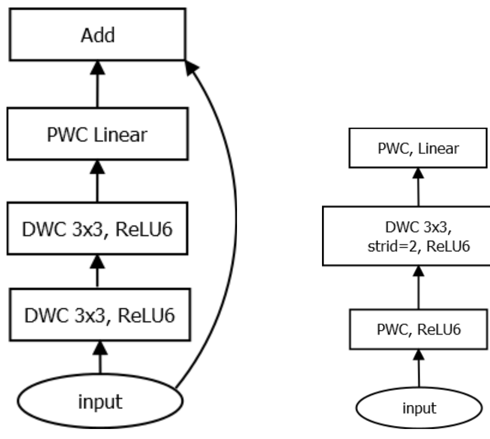
Fig. 1. Comparison of different convolution types.

This brief is organized as follows. Section II provides fundamental knowledge of depthwise separable convolution, followed by one of its application, MobilNetV2. Section III describes the architecture of the accelerator, including the matrix multiplication engine, and on-chip buffer organization. System implementation and its results are discussed in Section IV. The conclusion is given in Section V.

II. Depthwise Separable Convolution

Depthwise separable convolution was first introduced in [18]. As one kind of the factorized convolutions, depthwise separable convolution factorizes the standard convolution into a depthwise convolution plus a pointwise convolution. Fig. 1 demonstrates how the standard convolution (SC), depthwise convolution (DWC) and pointwise convolution (PWC) work. In standard convolution, each input channel has to do a convolution with one specific kernel, and then the result is the sum of the convolution results from all channels. While in depthwise separable convolution case, depthwise convolution is the first step, performing the convolution for each input channel individually. The next step is to do convolution in pointwise, which is actually a standard convolution with kernel size 1×1. Comparing to standard convolution, using depthwise separable convolution considerably reduces the number of mathematical operations and the number of parameters.

As it is shown in Fig. 1, considering the input feature map with size $M \times M \times N$ and kernel size $K \times K \times N \times P$,



(a) stride = 1 (b) stride = 2

Fig. 2. Bottleneck operations in different strides.

in case of stride length of 1, the number of weights needed for standard convolution is [14]

$$W_{SC} = K \times K \times N \times P \tag{1}$$

and the corresponding number of operations is

$$O_{SC} = M \times M \times K \times K \times N \times P \tag{2}$$

In case of depthwise separable convolution, the total number of weights is

$$W_{DSC} = K \times K \times N + N \times P \tag{3}$$

and the total number of operations is

$$O_{DSC} = M \times M \times K \times K \times N + M \times M \times N \times P \tag{4}$$

Thus, the reduction factors on weights and operation are calculated in (5)-(6):

$$F_W = \frac{W_{DSC}}{W_{SC}} = \frac{1}{P} + \frac{1}{K^2} \tag{5}$$

$$F_O = \frac{O_{DSC}}{O_{SC}} = \frac{1}{P} + \frac{1}{K^2} \tag{6}$$

One of the typical application of depthwise separable convolution is MobileNetV2, the successor of MobileNetV1 [14]. Comparing to its first version, the newly proposed MobileNetV2 further decreased the number of weights by shrinking the output channels in some layers. It also improves its performance through importing one more pointwise convolution layer before the depthwise separable convolution. The new operation is called bottleneck (Fig. 2).

The network structure of MobileNetV2 is illustrated in Table I.

III. System Design

A. Architecture Overview

The block diagram in Fig. 3 gives an overview of this accelerator. The proposed matrix multiplication engine (MME) array in this brief is responsible for all the CNN operations, including convolution, normalization, ReLU and pooling. All the parameters and input images are stored on off-chip

TABLE I
MOBILENETV2 STRUCTURE [15], WHERE EACH LINE REPRESENTS A SEQUENCE OF 1 OR MORE IDENTICAL (EXCEPT STRIDE) LAYERS. ALL DEPTHWISE CONVOLUTIONS USE 3 X 3 KERNELS

input	operator	extend factor	output channel	repeat time	stride
224x224x3	standard conv.	-	32	1	2
112x112x3	bottleneck	1	16	1	1
112x112x16	bottleneck	6	24	2	2
56x56x24	bottleneck	6	32	3	2
28x28x32	bottleneck	6	64	4	2
14x14x64	bottleneck	6	96	3	1
14x14x96	bottleneck	6	160	3	2
7x7x160	bottleneck	6	320	1	1
7x7x320	pointwise conv.	-	1280	1	1
7x7x1280	avgpool 7x7	-	-	1	-
1x1x1280	pointwise conv.	-	1000	-	

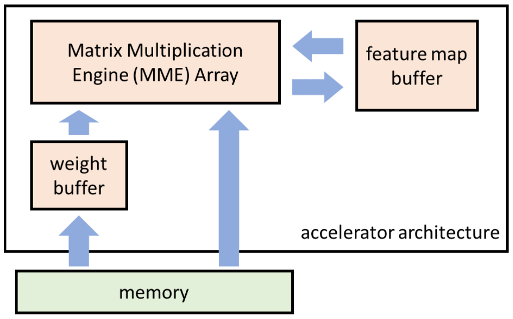


Fig. 3. Block diagram of accelerator system.

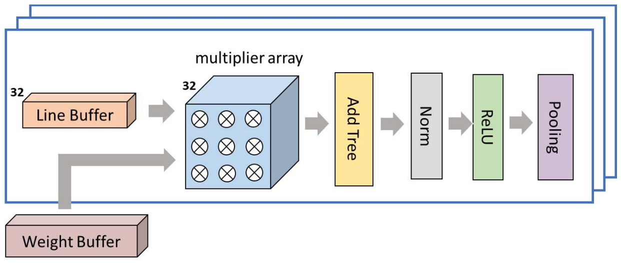


Fig. 4. Block diagram of an MME.

memory. A ping-pong weight buffer is placed between MME array and memory to maximize the bandwidth. Biases are loaded to the registers in MME array. Feature map buffer stores all the intermediate feature maps to avoid the latency brought by off-chip memory read and write. The accelerator is controlled by a general finite state machine (FSM).

B. *Matrix Multiplication Engine*

In this brief, each MME consists of 32 slices line buffer, 32 slices 3 $\times$ 3 multiplier array, 1 adder tree, 1 normalization (Norm) block, 1 ReLU block and 1 pooling block (Fig. 4). In each convolution, MME loads the feature maps and weights to line buffers. After multiplication in multiplier array, adder tree sums the products according to the selected convolution type. The following operations are optional normalization, ReLU and pooling.

1) Line Buffer: The working length of line buffer can be selected by control FSM to fit different input sizes, as

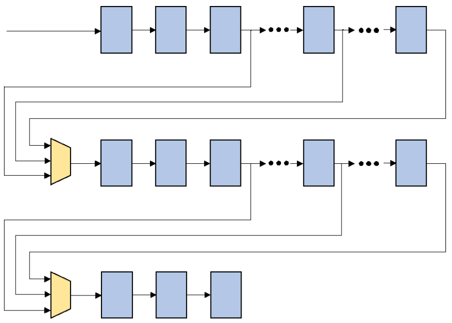


Fig. 5. Line buffer in MME.

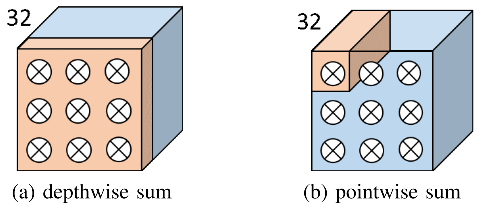


Fig. 6. Adder tree modes for different convolution.

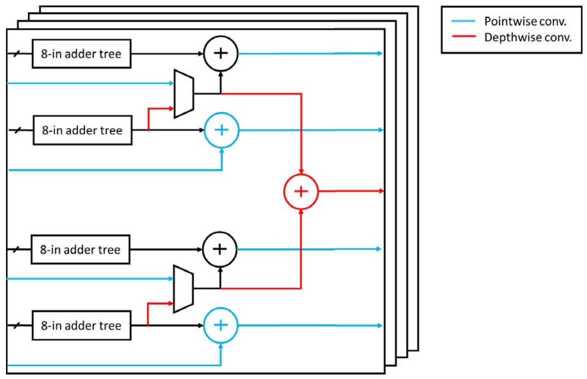


Fig. 7. Block diagram of adder tree.

it is illustrated by Fig. 5. The implementation length is $(K-1) \times M+K$.

2) Adder Tree: Adder tree is configurable to do the summing operation in depthwise or pointwise (Fig. 6). In Fig. 7, black lines or blocks are shared by both types of convolution. Blue part is used when doing depthwise convolution. While red part works if pointwise convolution is selected. All the biases all added in this stage.

3) Standard Convolution: To avoid losing too much information, standard convolution is adopted to do the first layer convolution. Therefore, this accelerator is adapted to be able to do the standard convolution with input feature map channel is 3. For vision applications, the channel number of input feature map is always 3.

4) Depthwise Convolution: Depthwise convolution performs convolution for each feature map separately. As shown in Fig. 8, adder tree is configured to sum up the products from each slice of multiplier array in parallel. For one MME, the output channel number is 32.

5) Pointwise Convolution: Pointwise convolution is actually standard convolution with kernel size 1 $\times$ 1 (Fig. 9). To

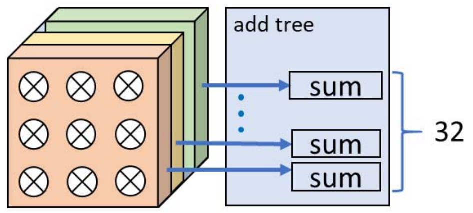


Fig. 8. Depthwise convolution in MME.

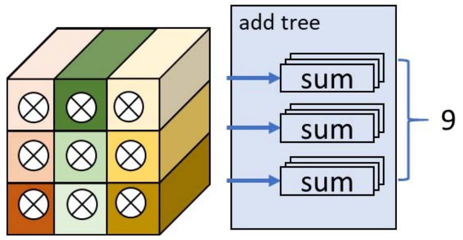


Fig. 9. Pointwise convolution in MME.

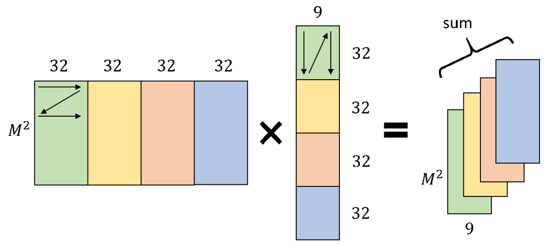
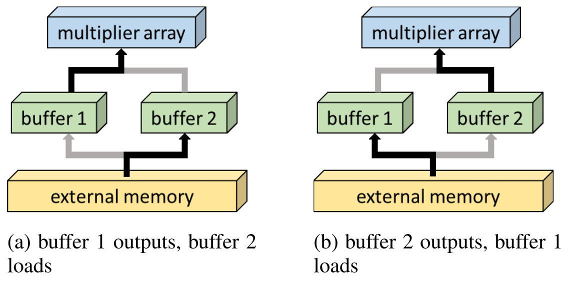


Fig. 10. Divide-conquer for large matrix multiplication.



(a) buffer 1 outputs, buffer 2 loads

(b) buffer 2 outputs, buffer 1 loads

Fig. 11. Weight buffer in ping-pong structure.

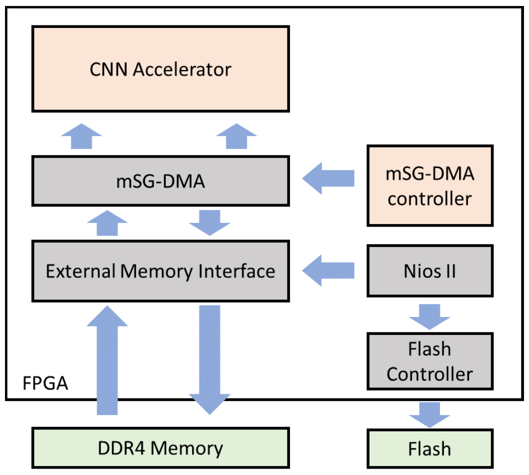


Fig. 12. System architecture of the FPGA design.

fully take advantage of all the multipliers in MME, the input feature map is divided into several $M \times M \times 32$ sub-matrices, and these sub-matrices are shifted into line buffers one after another. This idea comes from divide and conquer algorithm in large matrix multiplication illustrated in Fig. 10, which consists in dividing large matrix into several small matrices and sum the results up after doing small matrix multiplication. For one MME, it is able to do $M^2 \times 32$ and 32×9 multiplication at once. The adder tree sums up the 32 products in each cell as revealed by Fig. 9. Thus the output channel number is 9.

6) Normalization: After training, parameters of batch normalization are fixed [19]. Thus the complex normalization is downgraded into multiplication and add operation.

7) Pooling: Average pooling and max pooling are treated differently. As pixels of a feature map channel are output one by one, average pooling could be easily calculated by adding one more multiply-accumulate stage by a factor of $1/S$, where S is average pooling size. On the other hand, max pooling needs one more comparison stage.

8) ReLU: Same as the pooling layer, a ReLU stage is added after the normalization stage. Three options: no ReLU, standard ReLU and ReLU6 are selectable.

C. Memory Organization

To have an efficient memory organization, one has to balance on-chip memory resources and external memory bandwidth. On-chip memory is limited on FPGA but supplies very high bandwidth. Contrarily, external memory has the capability to store large amount of data but with the penalty of limited bandwidth. Therefore, in this proposed accelerator, we adapt the hierarchical memory methodology. Weight buffer loads the needed parameters from external memory before each convolution starts. This, on one hand, reduces the latency caused by parameters loading, and on the other hand, avoids the latency brought the limited bandwidth of external memory. Besides, weight buffer is built as a ping-pong buffer (Fig. 11), which means that when weight buffer 1 outputs data for convolution, the weight buffer 2 loads the data from external memory for the next one and vice versa.

Intermediate feature maps is another way chosen during system design to reduce processing time. Its size depends on the number of MME instantiated and the size of feature map.

IV. Results

The proposed accelerator architecture (Fig. 12) is demonstrated by implementing the MobileNetV2 network on the Arria 10 SoC Development Kit (10AS066N3F40E2SG), which contains 251680 ALMs, 2131 M20K, and 1687 DSP blocks. The design consideration will be described below and then followed by implementation results with resource utilization.

A. Implementation Consideration

As mentioned in Section I, lower numerical precision is sufficient for CNN. So 16-bit quantization strategy

TABLE II
RESOURCE USAGE OF MOBILENETV2

Name	ALM	DSP	RAM
MME	66127(26.3%)	1278(75.9%)	51(2.4%)
Weight Buffer	9317(3.7%)	0(0%)	0(0%)
Feature Map Buffer	1(0%)	0(0%)	1779(83.4%)
Others	6308(2.5%)	0(0%)	14(0.6%)
Totally	81753(32.5%)	1278(75.9%)	1844(86.5%)

TABLE III
COMPARISON TO OTHER IMPLEMENTATION

	[11]	[12]	this paper
Network	RR-MobileNet	MobileNetV1	MobileNetV2
Platform	Zynq UltraScale+	Stratix-V	Arria 10 SoC
Speed	127.4 fps	231.7 fps	266.2 fps

is chosen because it is widely selected by previous works [2], [3], [6], [20].

Based on the description in Section III, 4-MME array is decided to instantiate in this design after carefully balancing the resources usage and processing time. The weight buffer size is 36Kb as a ping-pong buffer. Since the update rate of weights when performing depthwise separable convolution is every $M \times M$ clock cycles. The size of intermediate feature map buffer is 24.5Mb.

B. Implementation Results

Fig. 12 presents the system architecture on Arria 10 SoC. Since HPS is not used in this design, only FPGA part is shown. The DDR4 memory is the one connected to the FPGA part. The CNN accelerator runs at frequency 133MHz. Its adder tree limits this frequency. A Nios II softcore micro-processor is implemented for loading weights and input images from flash memory to DDR4 external memory. An external memory interface IP combined with a Modular Scatter-Gather Direct Memory Access (mSG-DMA) IP are used to bridge the buffers in the CNN accelerator and the FPGA memory, whose maximum bandwidth is 8.5GB/s. This structure avoids the host's intervention during multiple transfers back and forth with DDR4 memory and makes non-continuous data movement more efficient. The function of customized mSG-DMA controller makes it possible to drive mSG-DMA to read/write different sizes of data from/to specific addresses, in order to fit convolutions in various sizes.

The implementation result is listed in Table II.

Table III provides a comparison between the solution proposed in this brief and other similar ones. Note that MobileNetV2 has more complex structure and higher accuracy on benchmarks.

V. CONCLUSION

In this brief, a high-performance, scalable CNN accelerator is proposed. This structure is optimized for depth separable convolution, which results in remarkably less operations and parameters. This makes it possible to run the CNNs on portable devices. By choosing different number of MMEs and variable on-chip memories, this accelerator can be fit into a large or small FPGA. As an example, the latest MobileNetV2 is implemented on Arria 10 SoC FPGA, which achieves 266.6 fps and 170.6 GOPS.

REFERENCES

[1] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[2] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE ACM Int. Symp. Microarchit. (MICRO)*, 2014, pp. 609–622.

[3] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," *ACM SIGARCH Comput. Archit. News*, vol. 43, no. 3, pp. 92–104, 2015.

[4] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y.-W. Tai, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, 2017, pp. 1–6.

[5] S. I. Venieris and C.-S. Bouganis, "fpgaConvNet: Automated mapping of convolutional neural networks on FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2017, pp. 291–292.

[6] H. Li *et al.*, "A high performance FPGA-based accelerator for large-scale convolutional neural networks," in *Proc. 26th Int. Conf. Field Program. Logic Appl. (FPL)*, 2016, pp. 1–9.

[7] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2017, pp. 45–54.

[8] J. Qiu *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2016, pp. 26–35.

[9] R. Tapiador *et al.*, "Comprehensive evaluation of openCL-based convolutional neural network accelerators in Xilinx and Altera FPGAs," *arXiv:1609.09296 [cs]*, Sep. 2016.

[10] A. Aimar *et al.*, "NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *arXiv:1706.01406v2 [cs]*, Mar. 2018.

[11] J. Su *et al.*, "Redundancy-reduced mobilenet acceleration on reconfigurable logic for ImageNet classification," in *Proc. Appl. Reconfig. Comput. Archit. Tools Appl.*, 2018, pp. 16–28.

[12] R. Zhao, X. Niu, and W. Luk, "Automatic optimising CNN with depthwise separable convolution on FPGA: (Abstact only)," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2018, p. 285.

[13] L. Sifre and S. Mallat, "Rigid-motion scattering for texture classification," *arXiv:1403.1687 [cs]*, Mar. 2014.

[14] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861 [cs]*, Apr. 2017.

[15] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," *arXiv:1801.04381v3 [cs]*, Apr. 2018.

[16] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv:1412.7024v5 [cs]*, Sep. 2015.

[17] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 1737–1746.

[18] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *arXiv:1610.02357v3 [cs]*, Apr. 2017.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv:1502.03167v3 [cs]*, Mar. 2015.

[20] Y. Ma, N. Suda, Y. Cao, J.-S. Seo, and S. Vrudhula, "Scalable and modularized RTL compilation of convolutional neural networks onto FPGA," in *Proc. 26th Int. Conf. Field Program. Logic Appl. (FPL)*, 2016, pp. 1–8.