A Reconfigurable Convolutional Neural Networks Accelerator Based on FPGA

Yalin Tang[1(✉)], Haoqi Ren[2], and Zhifeng Zhang[2]

[1] Tongji University, Shanghai, China
2030815@tongji.edu.cn

[2] School of Electronics and Information Engineering, Tongji University, Shanghai, China

Abstract. With the development of lightweight convolutional neural networks (CNNs), these newly proposed networks are more powerful than previous conventional models [4, 5] and can be well applied in Internet-of-Things (IoT) and edge computing. However, they perform inefficiently on conventional hardware accelerators because of the irregular connectivity in the structure. Though there are some accelerators based on unified engine (UE) architecture or separated engine (SE) architecture which can perform well for both standard convolution and depthwise convolution, these versatile structures are still not efficient for lightweight CNNs such as EfficientNet-lite. In this paper, we propose a reconfigurable engine (RE) architecture to improve the efficiency, which is used in communications such as IoT and edge computing. In addition, we adopt integer quantization method to reduce computational complexity and memory access. Also, the block-based calculation scheme is used to further reduce the off-chip memory access and the unique computational mode is used to improve the utilization of the processing elements. The proposed architecture can be implemented on Xilinx ZC706 with a 100 MHz system clock for EfficientNet-lite0. Our accelerator achieved 196 FPS and 72.9% top-1 accuracy on ImageNet classification, which is 27% and 18% speedup compared to CPU and GPU of Pixel 4 respectively.

Keywords: convolutional neural network · depthwise convolution · quantization · hardware accelerator · EfficientNet

1 Introduction

In recent years, convolutional neural networks (CNNs) play significant roles in researches on some hot issues like Artificial Intelligence (AI) and Deep Learning (DL), especially their applications in image processing [1, 2, 3]. Since AlexNet won the Image-Net Large Scale Vision Recognition Challenge competition in 2012 [4], more and more deeper CNNs for image classification with high predictive accuracy have been proposed. This comes with some challenges such as increasing computations, parameters and memory accesses. In order to solve these problems, lightweight CNNs such as ShuffleNet [6], MobileNet [7] and EfficientNet [8] were proposed, which adopt depthwise separable convolution to reduce the size of parameters and enhance the speed of inference with limited loss in accuracy.

F. Gao et al. (Eds.): ChinaCom 2022, LNICST 500, pp. 259–269, 2023.
https://doi.org/10.1007/978-3-031-34790-0_20

Even though these lightweight CNN models are more powerful than previous conventional models, they are still costly to be implemented on hardware. Thus, the call for efficient hardware implementations which are able to tackle the above problems arises. At the very beginning, Central processing units (CPUs) are used for the infrastructure of the CNN models [9]. But they are very inefficient in dealing with a large number of complicated convolutional calculations. Graphics processing units (GPUs) are more widely used with remarkable performance in handling large amounts of parallel computations because of its plentiful computing resources and high memory bandwidth [10]. However, it is not suitable for edge computing or Internet of Things (IoT) due to the disadvantage of its high energy consumption. Application-specific integrated circuit (ASIC) is much better in terms of power, performance and area [11]. Nevertheless, it is a fully custom designed chip for certain functionality and the hardware structure remains fixed once it is produced. With the rapid development of CNN models, it is lack of flexibility to be adapted to the advances. In this scenario, FPGAs have the advantages of flexible reconstruction and customizability with its programmable architecture to implement any target applications on hardware while maintain the benefits of low power and high performance of ASICs.

Despite the use of FPGAs to accelerate CNN models has become an academic hotspot, some FPGA accelerators [12, 13, 14], which aimed at large and conventional models such as AlexNet, VGG16 and ResNet, are not well applied in lightweight CNN models with irregular connectivity mentioned above. On account of the depthwise separable convolution adopted in these lightweight CNNs, those FPGA accelerators mentioned above are inefficient and underutilized. Therefore, it is necessary to design customized architectures to deploy the lightweight CNNs.

Shivapakash proposed a Multi-bit accelerator to enhance the power efficiency, which achieves 4.5 $\times$ lower power consumption than 32-bit architectures by truncating the bits of parameters and activations sequentially [15]. But it still needs too much off-chip memory access. [16] proposed a SE architecture optimized for MobileNet to schedule the different data-path for standard convolution and depthwise convolution. It is underutilized due to the workload imbalance in EfficientNet. LETA, proposed in [17], adopted UE architecture to improve the efficiency and resource utilization. Its main problem is the high off-chip memory traffic.

This paper presents a customized hardware accelerator with a reconfigurable calculation core based on FPGA which targets quantized lightweight CNNs: both activations and weights are 8 bits. This accelerator is used in communications (IoT or edge computing). The main contributions of this work are as follows:

- A reconfigurable engine architecture with a reconfigurable computational core for both standard convolution and depthwise convolution.
- A block-based convolutional calculation scheme (pointwise-depthwise-pointwise) is designed to reduce off-chip memory access.
- Efficiently scheduling the calculation mode and blocking the image to maximize the utilization of both processing elements and on-chip memory.

The rest of the paper is organized as follows. Section 2 introduces the inverted residual with linear bottleneck module which contains depthwise separable convolution, the EfficientNet-lite model and quantization. In Sect. 3, we present the proposed customized

architecture of the accelerator and some details. The results are shown in Sect. 4, and Sect. 5 gives the conclusions.

2 Backgroud

2.1 Inverted Residual with Linear Bottleneck Module

Recent state-of-the-art CNNs such as MobileNet, EfficientNet are based on an inverted residual structure with linear bottleneck. It consists of a pointwise convolution (*Pwcv*) layer, a depthwise convolution (*Dwcv*) layer and a pointwise convolution layer again sequentially which is similar to the basic structure of ResNet [5]. If skipping connections is applied, the input feature maps will add to the results after convolution. The main difference is that the standard convolution (*Scv*) in the middle layer is replaced by the *Dwcv* which greatly reduces the size of parameters. Figure 1 shows the difference. Both of the *Pwcv* is actually *Scv* with a 1×1 filter while the former one is used to expand the dimensionality, the other is the opposite. The *Dwcv* in the middle layer, the channels of which are computed independently, is spilt from the *Scv*. The separable channels make the *Dwcv* faster and smaller than the *Scv*.

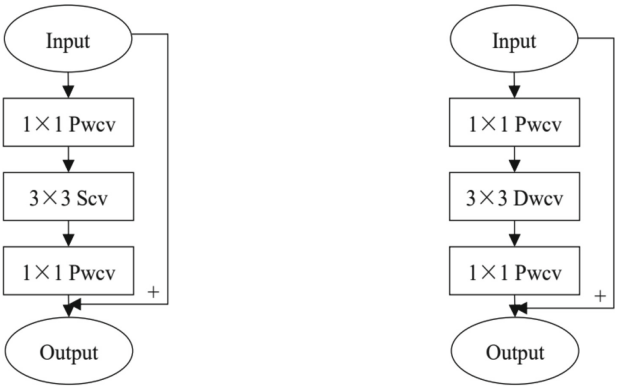

Fig. 1. The basic residual block in ResNet (left) and EfficientNet (right).

Assume that the height and width of the input feature map are H_i and W_i respectively, and the number of input channels is C_i. *Scv* and *Dwcv* apply the same convolutional kernel $K \times K$ with the stride length of 1 to generate the output feature map of which size is $H_o \times W_o \times C_o$. But the amount of operations is different as follows (In the case of *Dwcv*, $C_i = C_o$):

$$O_{Scv} = C_i \cdot H_o \cdot W_o \cdot C_o \cdot K^2 \tag{1}$$

$$O_{Dwcv} = H_o \cdot W_o \cdot C_o \cdot K^2 \tag{2}$$

Thus, we can see that *Scv* requires C_i times as much operations as *Dwcv*.

2.2 EfficientNet-Lite

EfficientNets are generated by using neural architecture search (NAS) and compound scaling method, which is based on a mobile-size baseline network manually designed. The main structure of EfficientNets is the *mobile inverted bottleneck block* like MobileNets, but with an additional *squeeze-and-excitation* optimization.

EfficientNet-lite makes some changes. All nonlinear activations called *swish* in EfficientNets [8] are replaced with *RELU6* activations for easier post-quantization and all *squeeze-and-excitation modules* are removed which makes the model to be easily implemented on hardware. All these changes is to make the model mobile and IoT friendlier. As shown in Table 1, EfficientNet-lite is more powerful than MobileNets with similar parameters and higher accuracy. The most accurate model, EfficientNet-lite4, is about 6% more accurate than MobileNetV3.

Table 1. EfficientNet-lite and MobileNet Performance Results on ImageNet.

Model	Params	MAdds	Top-1 Acc	CPU
EfficientNet-lite0	4.7M	407M	75.1%	12 ms
EfficientNet-lite1	5.4M	631M	76.7%	18 ms
EfficientNet-lite2	6.1M	899M	77.6%	26 ms
EfficientNet-lite3	8.2M	1.44B	79.8%	41 ms
EfficientNet-lite4	13.0M	2.64B	81.5%	76 ms
MobileNetV1	4.2M	575M	70.6%	113 ms
MobileNetV2	3.4M	300M	72.0%	75 ms
MobileNetV2(1.4)	6.9M	585M	74.7%	143 ms
MobileNetV3	5.4M	219M	75.2%	51.2 ms

* The CPU used for EfficientNet-lites is Snapdragon 855, while the CPU used for MoblieNets is Snapdragon 821.

Considering the area and real time, we target EfficientNet-lite0 to be implemented on hardware accelerator. Other EfficientNet-lite models can be well extended.

2.3 Integer Quantization

Model integer quantization has been widely applied in academia and industry as an optimization technology that can effectively reduce the size of the model and speed up the inference of CNNs with limited loss in accuracy. The width of quantized data can be 8-, 4-, 2- or 1-bit. Since 8-bit low-precision inference is commonly used, we target 8-bit integer quantization in the following discussion.

The aim of the quantization algorithm is to map 32-bit floating-point values to 8-bit integers. This can be written as:

$$q = round(\frac{r}{s} + Z) \tag{3}$$

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}} \tag{4}$$

$$Z = round(q_{max} - \frac{r_{\max}}{S}) \tag{5}$$

where q, r is quantized integer and floating-point real number respectively. S denotes the scaling factor that represents the map ratio between q and r, and Z denotes the zero-point that represents the quantized integer of zero in real number. Function *round()* means the value is rounded to the nearest integer. R_{min} and r_{max} are the maximum and minimum values of r respectively, and q is the same. For 8-bit integer quantization, q_{min} is –128 and q_{max} is 127 for symmetric scheme while q_{min} is 0 and q_{max} is 255 for asymmetric scheme.

In order to reduce the accuracy loss caused by quantization, we adopt different schemes for weight and activation. For the weight quantization, the per-channel symmetric scheme is used, which means each channel of the feature map has independent scaling factor S_W and zero-point Z_W. To simplify the calculation, we just apply S_W and Z_W to the bias quantization. And we just use a constant scaling factor S_A and zero-point Z_A within a layer for activation quantization with asymmetric scheme. Particularly, we need to collect the range of activations during the inference and choose the appropriate r_{min} and r_{max} to minimize the KL-Divergence between the quantized activations distribution and the original since the range of activations is not fixed for different inputs.

Assume that the weight of convolution is W, the bias is B, the input is X, and the output is A. Since convolution is essentially a matrix operation, the computation can be written as:

$$\mathrm{A} = \sum\nolimits_{i}^{N} W_i X_i + B \tag{6}$$

After quantization and adjustment, it becomes:

$$q_A = \frac{S_W S_X}{S_A}\left(\sum\nolimits_{i}^{N}(q_W - Z_W)(q_X - Z_X) + q_B\right) + Z_A \tag{7}$$

where q_X, q_A, q_W, q_B are quantized input, output, weight and bias respectively. And S_X, S_A, S_W denote the scaling factors of input, output and weight respectively, which is same to Z. In order to compute with 8-bit integer during the quantitative inference, the results need to be multiplied by a scaling factor in the above equation for each channel after convolution.

Table 2 shows the performance of the quantized EfficientNet-lite0 based on Pytorch compared with the floating-point model and the official implementation based on Tensorflow [8]. Note that the batch normalization layers and activation layers have been merged into the convolutional layers during the quantization to enhance the accuracy.

Table 2. Performance Results of Different Framework After Quantization or Not.

Model	Framework	Precision	Top-1 Acc	CPU
EfficientNet-lite0	Pytorch	FP32	73.6%	57.3 ms
EfficientNet-lite0	Pytorch	INT8	72.9%	13.6 ms
EfficientNet-lite0	Tensorflow	FP32	75.1%	12 ms
EfficientNet-lite0	Tensorflow	INT8	74.4%	6.5 ms

* The CPU we used (Intel Core i5-6300HQ) is different from the official one (Snapdragon 855).

3 Proposed Architecture

3.1 Overall Architecture

Figure 2 Presents the block diagram of the proposed architecture. The accelerator is a reconfigurable engine architecture and is driven by pre-compiled instructions. In the whole system, the Processing System (PS) tiles the input image into blocks and schedules the computations sequentially. The data blocks and instructions are put into the off-chip memory from PS before run time.

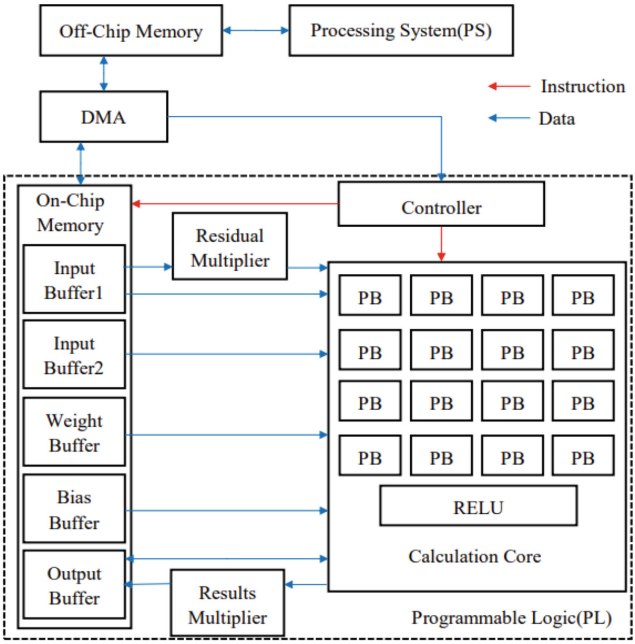

Fig. 2. Overall Proposed Architecture.

When the accelerator is started, the controller fetched instructions from off-chip memory. These instructions will be parsed and dispatched to the computational core.

The input feature maps, weights and biases will be filled into on-chip memory from the off-chip memory. Both standard convolution and depthwise convolution are computed in the core as well as the *RELU* function. The residual multiplier is used to scale the input feature maps for skipping connection, and the result multiplier is to quantize the results with the factor as in *Eq. (7)*. After convolution computations, the results are written back into off-chip memory. The final results of all these convolution layers are sent back to PS. Since we target EfficientNet-lite model, the pooling and full connection operation are computed in PS.

3.2 Reconfigurable Calculation Core

There are 16 processing blocks (PBs) inside the core, where each consists of 8 processing elements (PEs) and adder tree. The details are shown in Fig. 3. The calculation core can compute up to 1024 multiply-accumulate operations per cycle.

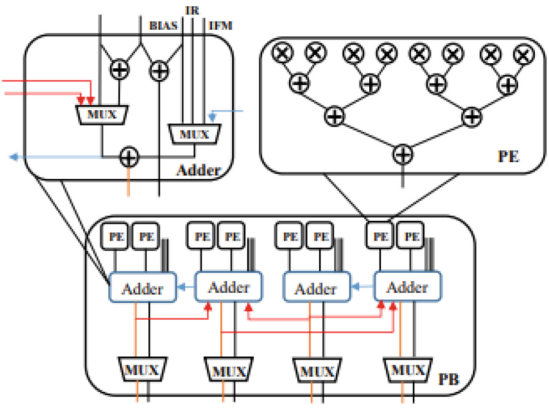

Fig. 3. The details of processing block.

In the inverted residual with linear bottleneck module, the number of the channels first increases then decreases. In order to improve the resource utilization of the processing elements, we enhance the structure and dataflow in [17]. For standard convolution, we only perform parallel computing along the channel dimension. And the parallelism of I/O channels can be configured as 16–64, 32–32 and 64–16. For depthwise convolution, we perform both the channel parallelism and the pixel parsallelism. The pixel parallelism can be configured as 2 or 4 (the corresponding channel parallelism is 64 or 32) according to the computation mode of the preceding pointwise convolution layer, and the total output is 128. The different parallelism can be performed by changing the output connections of the Adders. The red line input of the mux in Fig. 3 is the yellow line output from other adder within the PB and the black line output of the Adder in Fig. 3 is only used in depthwise convolution.

3.3 Image Blocking and Block-Based Data-Path

Due to the limited on-chip memory, we partition the input image and design a unique computation scheme as shown in Fig. 4. The inter-channel computation is scheduled with weight reuse and input feature map fully reuse within a convolution layer. The inverted residual block is regarded as a basic computing block. The input feature maps are fetched into the first input buffer as shown in Fig. 2. After the pointwise convolution layer, the results are sent to the second input buffer to prepared for depthwise convolution. Then, the results are sent to the first input buffer again to be used in the next pointwise convolution layer. The output buffer is used to store the intermediate results and the final results of the above convolution. After the basic block complete all the computations, the final results are written back to the off-chip memory. The data path of activations is shown in Fig. 5 briefly.

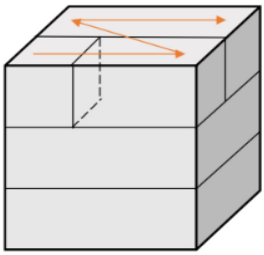

Fig. 4. Image blocking and computation schedule.

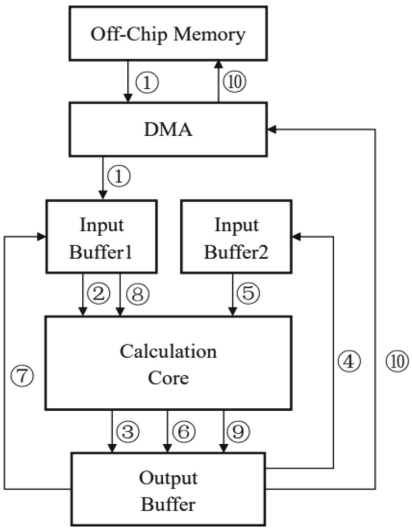

Fig. 5. The brief data path of activations.

4 Experiments

We evaluate the performance of the proposed architecture and compare it with previous works in this section. The whole system is implemented by Verilog HDL and runs at 100 MHz frequency on Xilinx ZC706 board. This platform is based on Zynq-7000 SoC with a dual-core ARM Cortex-A9 core processor. Vivado (v17.4) is used to synthesize and place-and-route the hardware design. The data set used for inference in this experiment is from ImageNet ILSVRC2012. The size of the images is $224 \times 224 \times 3$ and the batch size is 20. We implement EfficientNet-lite0 on this hardware accelerator and it takes about 384M multiply-accumulate operations to compute the inference of one single image.

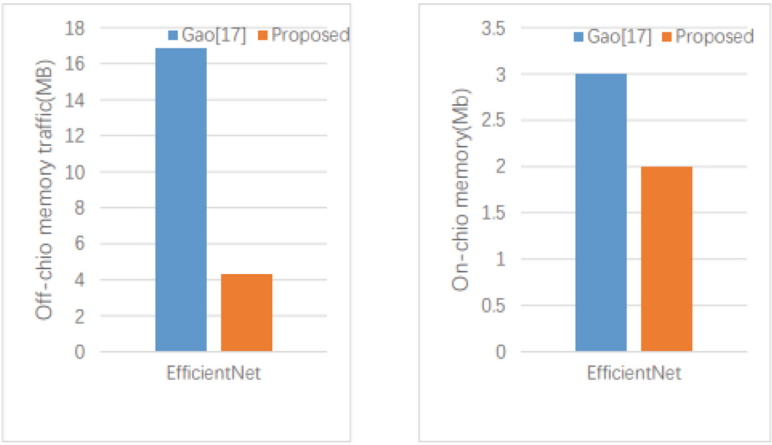

Fig. 6. Comparisons of off-chip memory traffic and on-chip memory.

Table 3 shows the results of the proposed architecture and comparisons with other works. The CPU of Pixel 4 is Snapdragon 855 and the GPU is Adreno 640. The proposed architecture is faster than CPU and GPU of Pixel 4 with limited accuracy loss (1–2% lower). Our reconfigurable engine (RE) architecture uses large on-chip memory size while reducing off-chip memory traffic. And compared with the unified engine (UE) architecture in [17], our architecture has about 4 $\times$ less off-chip traffic and still less on-chip memory, achieve a performance of 196 FPS in EfficientNet-lite0 as shown in Fig. 6. Though there is a small gap in latency because of lower frequency, our architecture still performs well.

Table 3. Performance Comparisons.

	CPU[8]	GPU[8]	Gao[17]	Proposed
Platform	Pixel4	Pixel4	Xilinx XCVU37P	Xilinx ZC706
Model	lite0	Lite0	B0	Lite0
Precision	INT8	FP32	INT8	INT8
Frequency	2.84 GHz	2.84 Hz	300 MHz	100 MHz
Off-chip Traffic	N/A	N/A	16.9 MB	4.3 MB
On-chip Memory	N/A	N/A	3 Mb	2 Mb
MAC Operation	407 M	407 M	390 M	384 M
Latency	6.5 ms	6.0 ms	4.1 ms	5.1 ms
GOPS	N/A	N/A	95.1	75.3
Accuracy	74.4%	75.1%	73.9%	72.9%

5 Conclusion

In this paper, a reconfigurable engine architecture for EfficientNet-lite which can be applied in communications such as IoT and edge computing is proposed. Integer quantization method is used to improve the efficiency with limited accuracy loss. Also, we design a block-based calculation scheme and schedule the dataflow to reduce the off-chip traffic and on-chip memory. Our architecture can achieve 4 $\times$ less off-chip traffic than the latest FPGA-based EfficientNet accelerator. And it is 27% and 18% speedup compared to CPU and GPU of Pixel 4 respectively.

References

1. Can, F., Eyüpoğlu, C.: Convolutional neural network architectures used in computer vision. In: 5th International Symposium on Multidisciplinary Studies and Innovative Technologies, pp. 305–311. IEEE Press, New York (2021)
2. Dong, Y., Liu, Q., Du, B., Zhang, L.: Weighted feature fusion of convolutional neural network and graph attention network for hyperspectral image classification. IEEE Trans. Image Process. **31**, 1559–1572 (2022)
3. Pinckaers, H., Ginneken, B.V., Litjens, G.: Streaming convolutional neural networks for end-to-end learning with multi-megapixel images. IEEE Trans. Pattern Anal. Mach. Intell. **44**, 1581–1590 (2022)
4. Russakovsky, O., et al.: ImageNet large scale visual recognition challenge. Int. J. Comput. Vision **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y
5. He, K., Zhang X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR, pp. 770–778. IEEE Press, New York (2016)
6. Ma, N., Zhang, X., Zheng, H.-T., Sun, J.: ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) Computer Vision – ECCV 2018. LNCS, vol. 11218, pp. 122–138. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01264-9_8

7. Howard, A., et al.: Searching for mobileNetV3. In: ICCV, pp. 1314–1324. IEEE Press, New York (2019)
8. Tan, M., Le, Q.V.: EfficientNet: rethinking model scaling for convolutional neural networks. In: ICML, pp. 6105–6114. PMLR, California (2019)
9. Abadi, M.: TensorFlow: a system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation, pp. 265–283. USENIX Association, Savannah, GA (2016)
10. Hazelwood, K., et al.: Applied machine learning at Facebook: a datacenter infrastructure perspective. In: IEEE International Symposium on High Performance Computer Architecture, pp. 620–629. IEEE Press, New York (2018)
11. Jouppi, N.P., et al.: A domain-specific supercomputer for training deep neural networks. Commun. ACM **63**, 67–78 (2020)
12. Yin, S., et al.: A high energy efficient reconfigurable hybrid neural network processor for deep learning applications. IEEE J. Solid-State Circuits **53**, 968–982 (2018)
13. Nguyen, D.T., Nguyen, T.N., Kim, H., Lee, H.: A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection. In: IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, pp. 1861–1873 (2019)
14. Venieris, S.I., Bouganis, C.: fpgaConvNet: mapping regular and irregular convolutional neural networks on FPGAs. IEEE Trans. Neural Netw. Learn. Syst. **30**, 326–342 (2019)
15. Shivapakash, S., Jain, H., Hellwich, O., Gerfers, F.: A power efficiency enhancements of a multi-bit accelerator for memory prohibitive deep neural networks. IEEE Open J. Circ. Syst. **2**, 156–169 (2021)
16. Wu, D., et al.: A high-performance CNN processor based on FPGA for MobileNets. In: 29th International Conference on Field Programmable Logic and Applications, pp. 136–143. IEEE Press, New York (2019)
17. Gao, J., et al.: LETA: a lightweight exchangeable-track accelerator for efficientnet based on FPGA. In: 2021 International Conference on Field-Programmable Technology, pp. 1–9. IEEE Press, New York (2021)