Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 3768724, 12 pages
https://doi.org/10.1155/2021/3768724

Research Article

An FPGA-Based Convolutional Neural Network Coprocessor

Changpei Qiu, Xin'an Wang, Tianxia Zhao, Qiuping Li, Bo Wang, and Hu Wang

The Key Laboratory of Integrated Microsystems, Peking University Shenzhen Graduate School, Shenzhen 518055, China

Correspondence should be addressed to Xin'an Wang; anxinwang@pku.edu.cn

Received 6 April 2021; Accepted 31 May 2021; Published 14 June 2021

Academic Editor: Wenqing Wu

In this paper, an FPGA-based convolutional neural network coprocessor is proposed. The coprocessor has a 1D convolutional computation unit PE in row stationary (RS) streaming mode and a 3D convolutional computation unit PE chain in pulsating array structure. The coprocessor can flexibly control the number of PE array openings according to the number of output channels of the convolutional layer. In this paper, we design a storage system with multilevel cache, and the global cache uses multiple broadcasts to distribute data to local caches and propose an image segmentation method that is compatible with the hardware architecture. The proposed coprocessor implements the convolutional and pooling layers of the VGG16 neural network model, in which the activation value, weight value, and bias value are quantized using 16-bit fixed-point quantization, with a peak computational performance of 316.0 GOP/s and an average computational performance of 62.54 GOP/s at a clock frequency of 200 MHz and a power consumption of about 9.25 W.

1. Introduction

Hardware acceleration of artificial neural networks (ANNs) has been a hot research topic since the 1990s [1, 2]. The most representative one is the implementation of a shallow artificial neural network gas pedal, ETANN, by Intel in 1989, which supports only a small amount of data [3, 4]. Convolutional neural networks have been proposed since 1989 and did not become a research hotspot until 2006, mainly due to the difficulty of hardware computing power at that time. The convolutional and fully connected layers in CNN are mainly multiplicative and additive operations with relatively single control process; however, the training and prediction process of CNN models generally requires hundreds of millions of multiplicative and additive operations [5, 6], of which 90% of the computations are concentrated in the convolutional layer [7]. To achieve high computational performance, it is often necessary to design highly parallel temporal and spatial architectures [8–10].

Temporal parallelism is mainly for CPUs and GPUs, which improve computational efficiency mainly by increasing frequency, multilevel cache, single instruction multiple data, single instruction multiple threads, etc. All Arithmetic Logic Units (ALUs) share controllers and memory. In these computing platforms, the convolutional and fully connected layers are mapped into matrix multiplication to participate in the computation. CNN has high computational density, single task, and high data reuse and requires large-scale computational logic and storage bandwidth without complex control logic, so the CPU is not suitable for CNN computation. On the contrary, GPU has thousands of computational cores and is more suitable for CNN computation; however, the power consumption and area of GPU are large, and the energy efficiency ratio is low mainly deployed in the cloud. However, GPUs are not suitable for embedded applications, which are power sensitive. Spatial parallelism is mainly for ASICs and FPGAs, mainly through stream processing, local accumulation, proximity storage, etc. to improve data reuse and thus computational efficiency. FPGAs are highly programmable and configurable, with high energy efficiency and short development cycles, especially with tools such as High Level Synthesis and OpenCL, which accelerate the development of FPGAs.

Sankaradas et al. designed a coprocessor for CNN based on FPGA [11] with low precision data bit-width (20-bit fixed-point quantization for weights and 16-bit fixed-point quantization for feature map values), supporting only fixed size convolutional kernel size, frequent

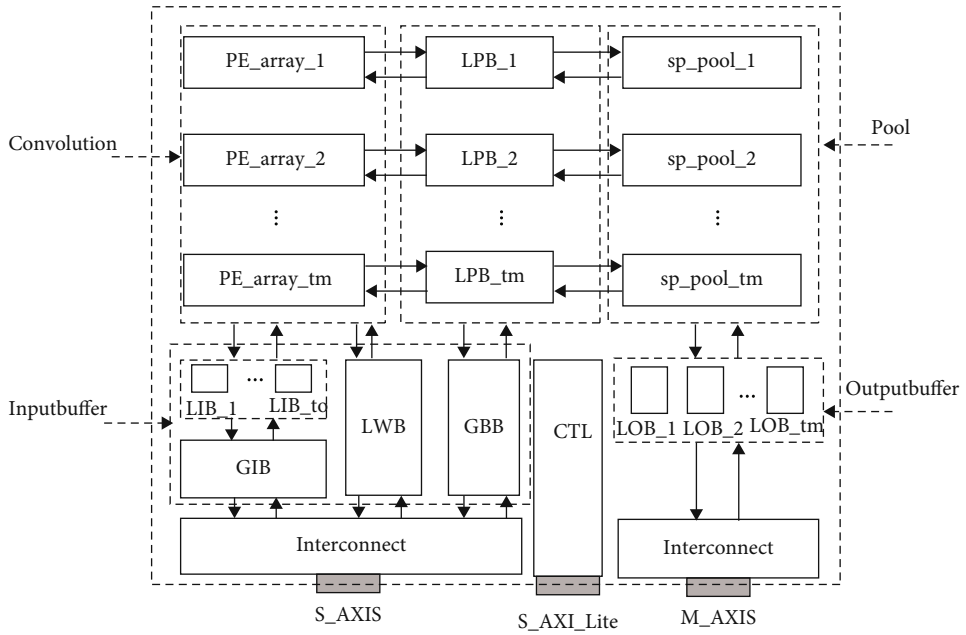

Figure 1: The schematic diagram of the overall architecture of the coprocessor.

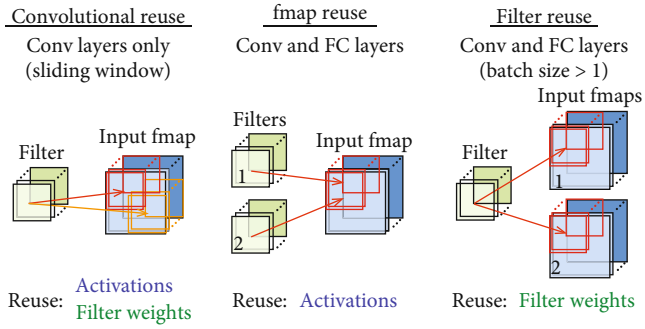

Figure 2: The form of data reuse in CNN.

accesses, high power consumption, and low computational efficiency because intermediate values need to be stored in DDR. Gokhale et al. designed the coprocessor nn-X for mobile embedded end [12, 13] with a peak computational performance of 200 GOP/s on Xilinx ZC706 platform. With the increasing complexity of CNN models, FPGA logic resources, and memory bandwidth, the design space of FPGAs is also expanding. In order to find the optimal gas pedal design solution, MIT proposed Eyeriss [14], a highly efficient and reconfigurable deep convolutional neural network accelerator chip. Eyeriss supports different sizes of input feature maps and convolutional kernel sizes, uses RLB (run-length-based) compression to reduce the average image data transfer bandwidth by a factor of 2, reduces the interaction between computational units and on-chip storage through data reuse and local accumulation, and reduces the interaction between data and DDR through hierarchical storage. This reduces power consumption.

In this paper, we analyze the convolutional neural network algorithm and design a high-performance and flexible FPGA-based convolutional neural network coprocessor by combining the hardware implementation and optimization with the characteristics of FPGA platform. The paper is organized in the following sections. Section 2 introduces the coprocessor architecture. Section 3 focuses on the design of each major module in the coprocessor. In Section 4, we perform FPGA hardware verification. Finally, Section 5 concludes.

2. Coprocessor Architecture

In this paper, we provide a coprocessor implementation for convolutional neural networks, which is aimed at accelerating the convolutional and pooling layers of convolutional neural networks on FPGAs and applying them to heterogeneous accelerated systems or embedded terminals. The design solution includes a controller, input buffer, output buffer, convolutional unit, and pooling unit. Figure 1 shows the schematic diagram of the overall architecture of the coprocessor. The input buffers include a global image buffer (GIB), tc local image buffers (LIBs), a global bias buffer (GBB), tm local weight buffers (LWBs), and tm local partial sum buffers (LPBs). The output buffers include tm local output buffers (LOBs); the convolution operation unit includes tm PE arrays; the pooling unit includes tm splice pools (referred to as sp_pool, a pooling unit containing image stitching function); tm and tc in the figure are given optimal values by theoretical analysis.

The main functions of each cell module in this architecture are as follows:

(1) The data I/O interface supports AXI4-Stream and AXI4-Lite bus protocols. The off-chip memory sends

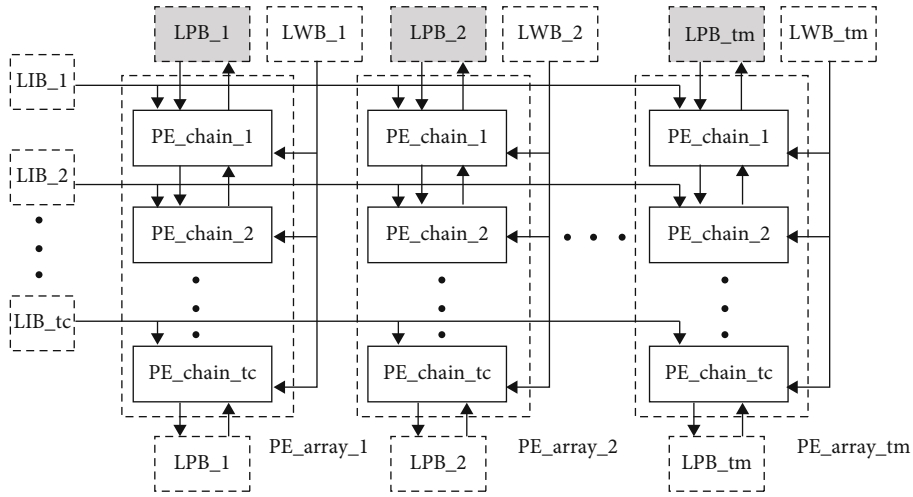

Figure 3: The schematic diagram of the proposed convolutional computation unit.

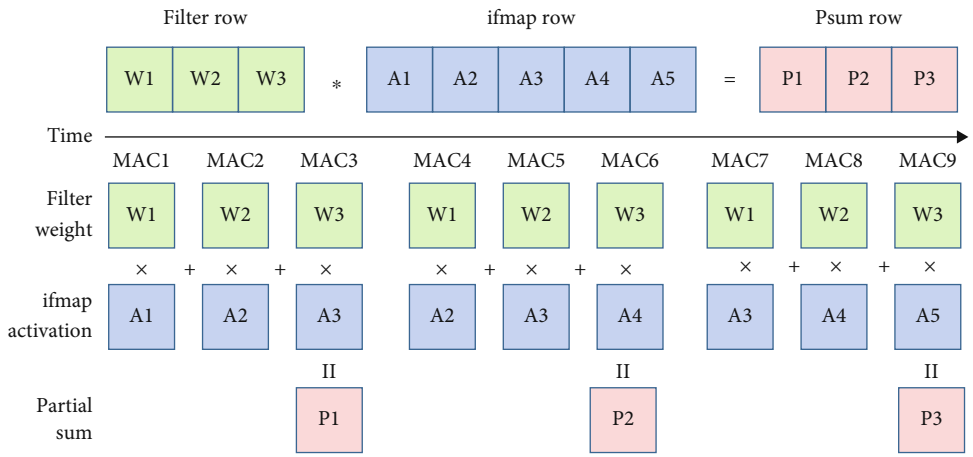

Figure 4: The schematic of the data flow of PE performing 1D convolution.

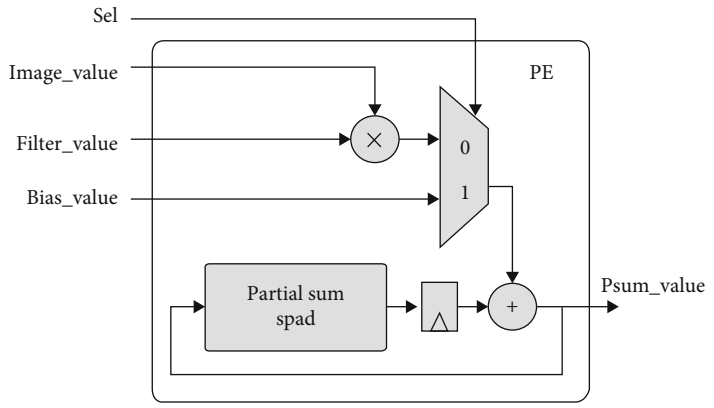

Figure 5: The schematic diagram of the PE unit.

data in bulk to the input buffer of the coprocessor through the AXI4-Stream Slave interface to provide feature map data, weight data, and bias data for the PE array

(2) The global controller communicates with external control signals through the AXI4-Lite bus interface. The global controller contains nine 32-bit instruction registers, which are used to store the parameters of

the convolutional layer as well as the enable signal and the end-of-computation signal

(3) GIB and GBB and tm LWBs all have an ID. After the data is input from the S_axis interface, only the buffer with the matching ID can receive the data

(4) GIB is distributing data to tc LIBs in the form of global broadcast, each LIB has its own ID, and only LIBs with matching IDs can receive data

(5) Each sp_pool can implement nonlinear activation, image stitching, and pooling. This design supports activation function ReLU and step size 2, size 2×2 maximum pooling

The FPGA development board used in this thesis is the ZYNQ-7000 ZC706 evaluation board, and the number of DSPs available and the size of on-chip storage is small compared to the mainstream chips. The amount of weight data in the fully connected layer is large, and there is no reusability. If the fully connected layer is executed in FPGA, only a small performance improvement can be obtained, but at the same time more computational and storage resources are occupied and the control logic is more complicated, which will directly affect the parallelism of the convolutional layer operation. Therefore, the coprocessor designed in this thesis is only responsible for the accelerated computation of the convolutional and pooling layers.

3. Design of Each Major Module in the Coprocessor

3.1. Convolutional Operation. Convolutional operations are the most central and computationally intensive operations in convolutional neural networks, so the efficiency of convolutional units directly affects the performance of the whole architecture. In fact, most of the hardware resources in the design solution are also allocated to the acceleration of convolutional operations. The basic operation of convolutional operation is matrix multiplication, and one convolutional operation consists of multiple multiplication and addition operations; in convolutional operation, the data is highly reusable, as shown in Figure 2.

3.1.1. Convolutional Reuse. A 3D convolutional kernel of size $k \times k \times C$ slides on a 3D feature of size fmap_size $\times$ fmap_size $\times$ C with a fixed step size. The same convolutional kernel convolves different receptive domains, and the filter weights are reused; the activations of two adjacent receptive domains overlap and are reused by different weights in one convolutional kernel.

3.1.2. fmap Reuse. M 3D convolutional kernels of size $k \times k \times C$ are slid over a 3D feature map of size fmap_size $\times$ fmap_size $\times C$ in a fixed step. When M convolution kernels convolve the same sensory domain, the activation values in that domain are reused.

3.1.3. Filter Reuse. A 3D convolutional kernel of size $k \times k \times C$ slides over N 3D feature maps of size fmap_size $\times$ fmap_size $\times C$ with a fixed number of steps. The same convolutional kernel can convolve multiple sensory domains at the same location in the feature map, and the kernel weights are reused.

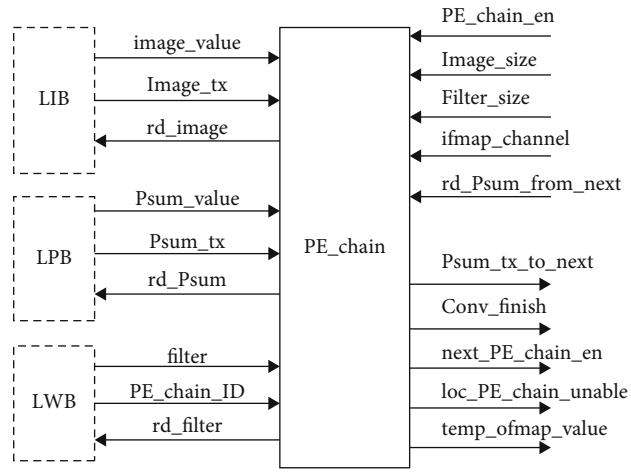

Figure 6: The schematic diagram of the I/O interface of the PE chain.

Although multiple multiplication and addition operations are required in a single iteration, the data demand is high, but in fact, since most of the data in the input feature subregion of the convolution operation can be reused, fully utilizing this feature can reduce the on-chip data cache capacity and off-chip storage bandwidth requirements.

3.2. Convolutional Cell Design. In this paper, in order to take full advantage of the data multiplexing form of the convolutional computation process and to fully consider the computational and storage resources of the FPGA, Figure 3 shows the schematic diagram of the proposed convolutional computation unit, which illustrates the data interaction between the computational unit and the memory.

The whole convolutional computation unit includes tm PE arrays, each PE array includes tc PE chains, each PE chain includes 3 PE, each PE contains a DSP, each PE can complete a one-dimensional convolution, each PE chain can complete a two-dimensional convolution, and multiple use of a PE chain or multiple PE chain parallel computation can achieve a three-dimensional convolution. As shown in Figure 3, first, there are tc LIBs corresponding to tc PE chains, and each LIB passes the image data to tm PE chains in the form of broadcast, realizing the feature map reuse form as shown in Figure 2. In addition, there are tm LWBs in the whole convolutional computation unit, and each LWB corresponds to one PE array, and the weights are passed to tc PE chains in the form of broadcast, realizing the convolutional reuse form shown in Figure 2. Finally, there are tm LPBs in the whole convolutional computation unit, each LPB stores the bias value (bias) from GBB for the first time and then stores the partial sum from PE chain; similarly, each LPB can output data to PE chain or to splice pool. Each LPB has two data input ports and two data output ports.

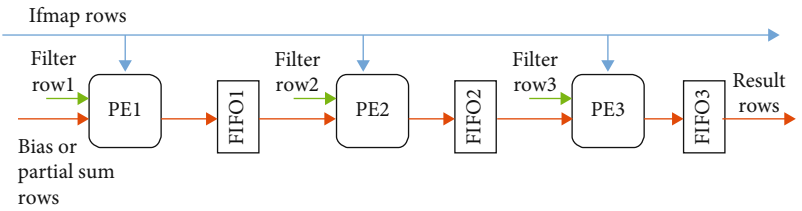

Figure 7: The schematic diagram of the proposed PE chain.

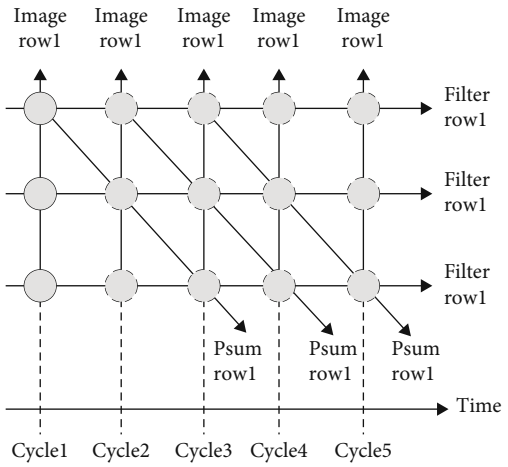

Figure 8: The space-time mapping (STM) of a PE chain.

3.3. PE. The most basic multiplication and addition unit PE in the convolution unit designed in this paper adopts the stream processing structure of RS, which can realize the one-dimensional convolution calculation. Assuming the size of the input feature map ifmap_size = 5 and the size of the convolution kernel $k = 3$, the data flow into PE is shown in Figure 4. The PE structure designed in this paper saves storage resources by reducing the Image Scratch Pad and Filter Scratch Pad compared to the PE structure of Eyeriss [10], which is certainly a good design in the case of storage constraints. Figure 5 shows the schematic diagram of the PE unit. Sel in the input signal is the selection control signal; Sel = 1 plus the bias value or the partial sum from the previous PE; Sel = 0 plus the product of the activation value and the weight value. This PE structure can be implemented in the FPGA with only one DSP example. In this paper, to avoid the logic overhead and control complexity associated with dynamic fixed-point quantization, both weights and activation values are quantized using a fixed-state 16-bit fixed-point quantization.

3.4. PE Chain. PE chain is the core of the entire convolutional computation unit. Without considering parallel computation, a PE chain can do all the computations of the convolutional layer in the neural network independently.

Figure 6 shows the schematic diagram of the I/O interface of the PE chain. When the PE chain detects an enable signal, it sends an activation request signal to the LIB, a partial sum signal to the LPB, and a weight request signal to the LWB; the LIB receives the request signal and sends a feature map value; the LPB sends a partial sum; the LWB sends a convolution kernel. A channel counter is designed in PE chain. When a 2D feature map value is computed, the counter is incremented by one until the counter equals the computed channel, and all input feature map channels are computed. This marks the completion of the computation of all channels in a 3D tile (a large feature map divided into several smaller feature maps, each of which called a tile).

PE chain computational logic: if the size of the convolution kernel is k, then a PE chain consists of k PEs connected sequentially and expanded in space-time, which is a two-dimensional systolic array structure. In this design, only 3 × 3 convolutional kernels are supported, so a PE chain consists of 3 PEs, as shown in Figure 7, and each two PEs are connected by a FIFO (the last PE chain in a single PE array as shown in Figure 3 contains only two FIFOs, and the last FIFO is replaced by an LPB).

A PE chain can implement a 2D convolution, and a PE chain can fully implement a 3D convolution calculation without considering the computational speed (all 2D convolutions are executed serially by a PE chain). Assuming an input feature map of size ifmap_size = 5 and a convolution kernel of size $k = 3$, the space-time mapping (STM) of a PE chain is shown in Figure 8.

Each solid circle in the graph represents one PE, three PEs in total, and the computation process of each PE is expanded along the time axis to form a two-dimensional pulsating array structure. Each row of the feature map is passed to each PE simultaneously in the form of a broadcast. The data flow of Psum_row1 in the space-time mapping diagram of PE chain is shown in Figure 9, where the data flow of each PE is shown in Figure 4. The first row of the convolution kernel is given to PE1, the second row to PE2, and the third row to PE3, with three rows of input simultaneously, which requires that one convolution kernel must be taken from the LWB at a time. In each cycle, the PE computes a one-dimensional convolution, and the partial sum is temporarily stored in the FIFO. When the previous PE finishes computing a row, the next PE is opened, and the partial sum in the previous FIFO is read as bias into the current PE.

3.5. PE Array. PE array is an array of PE chain, and the whole convolutional computation unit is implemented by multiple PE arrays, utilizing two data reuse modes, such as convolutional reuse and feature map reuse shown in Figure 2, to achieve a high degree of parallelism in convolutional computation. Due to the limited on-chip storage, the input feature map must be segmented, and the segmentation strategy is related to the overall structure of the PE array, which also

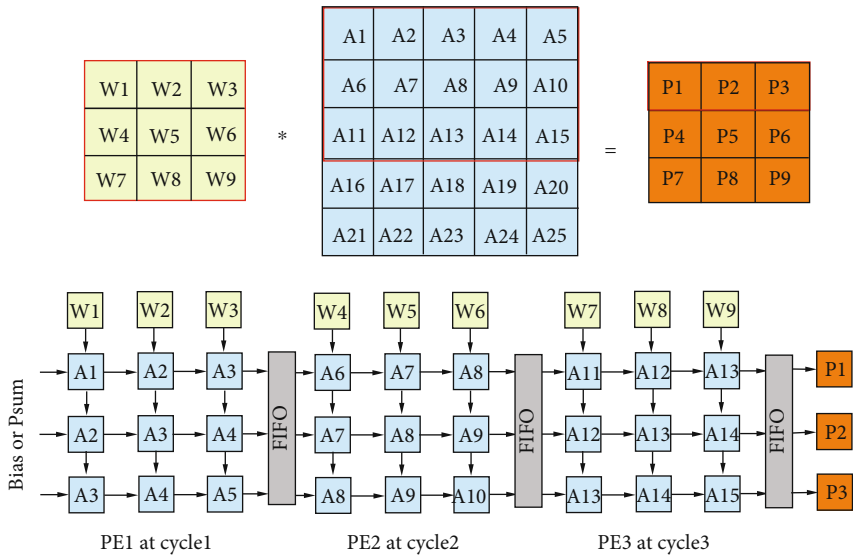

Figure 9: The data flow of Psum_row1 in the space-time mapping diagram of PE chain.

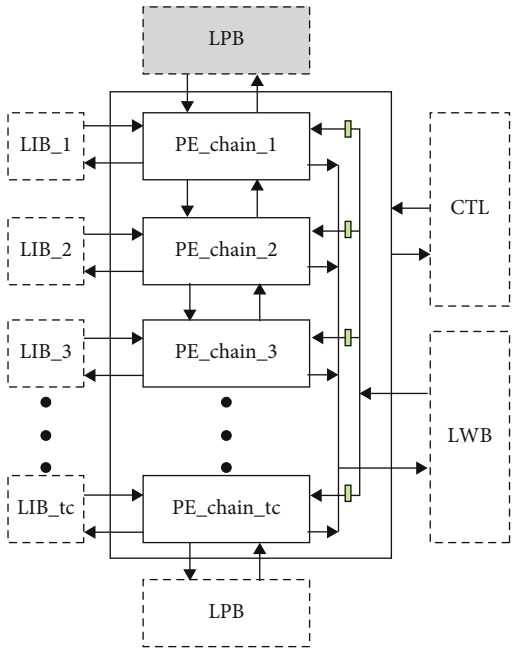

Figure 10: The overall structure of a single PE array.

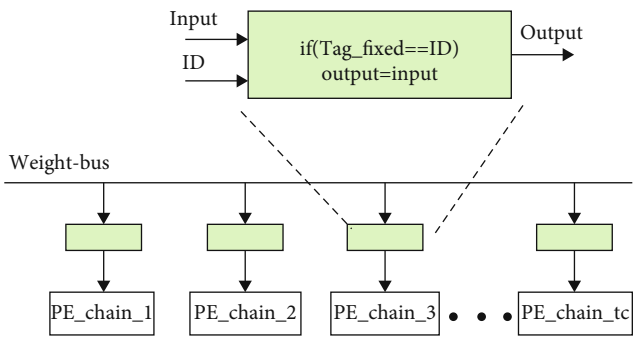

Figure 11: The schematic diagram of the proposed PE chain.

leads to the decision of computation modes for multiple PE arrays.

The overall structure of a single PE array is shown in Figure 10. The detailed I/O interfaces are omitted, and the directions indicated by the arrows in the figure only represent data interaction relationships.

A single PE array contains tc PE chains, and tc PE chains are connected in turn. When PE array receives the signal from each PE chain, it can easily control the opening and closing of each PE chain. The nth PE chain sends a signal to the nth-1st PE chain to request a partial sum. After the nth-1st PE chain receives the requested signal, it sends a signal to the next PE chain and at the same time reads its own FIFO3 and sends a partial sum. The tcth PE chain differs from the previous PE chain in that it contains only two FIFOs, and the third FIFO is replaced by the LPB, and the first PE chain sends a request signal to the LPB.

Data interaction between PE array and each buffer: each PE chain has its corresponding local image buffer (LIB), and all PE chains share a local weight buffer (LWB), as shown in Figure 11, and the LWB sends the weights to PE array in the form of broadcast, and only the PE chain with matching ID can receive them. chain can only receive them. The start of PE array is controlled by three signals, the signal from GIB and the signals from LPB and LWB, respectively.

3.6. Image Segmentation Strategy. For a PE array with a PE chain in each PE array, the entire convolutional computation unit has only one PE chain (with 3 DSPs), and all local caches alone require at least 53 Mb of storage space to satisfy the computation of all convolutional layers. For 4 PE arrays, each PE array contains 64 PE chains, i.e., the entire convolutional computation unit uses 256 PE chains (with 768 DSPs), and the local cache alone requires at least 255 Mb of storage space. The on-chip storage of FPGA chip resources is only 19.1 M, while the number of DSP is 900. Obviously, if the feature map is not divided, the on-chip storage is not enough and the computational unit cannot be fully utilized. In order

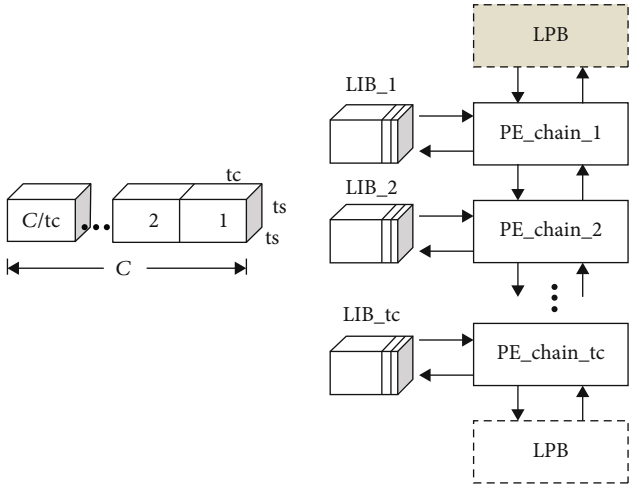

Figure 12: Correspondence of a 3D tile to a PE array.

to balance the allocation of on-chip storage and computational units, this paper adopts the strategy of splitting the feature map size. The smaller the size of the split feature map, the smaller the local storage occupation, and the larger the global cache can be allocated with limited storage resources, reducing the delay of data loading during each round of computation of computation units. Of course the trouble brought is the increase in the number of rounds of computation needed and the increase in the number of data loading from the global buffer. So the size of the partition needs to be adapted to the architecture of the hardware, taking into account both the parallelism of multiple PE arrays and the time delay of opening multiple PE chains in each PE array in turn.

In the hardware architecture design of this thesis, for the convenience of control, so that the size of each tile is equal, C should be divisible by tc. A tile $(\text{ts} \times \text{ts} \times C)$ of depth C is decomposed into C/tc groups according to successive groups of tc 2D tile, and the 2D tile in each group is stored in the corresponding LIBs, which correspond to PE array, as shown in Figure 12. Each LIB stores 2D tile.

The actual simulation verifies that after PE chain is enabled, it needs to request feature map data, weight data, and partial sum data and then turn on the first PE, and the process takes five clock cycles. According to the calculation logic of PE chain, the previous PE needs to calculate one row of feature map data before the next PE can be turned on, and the process takes $3(\text{ts} - k + 1)$ clock cycles. The PE3 calculation of the previous PE chain produces the first partial sum to start the next PE chain, and the actual simulation verifies that the process takes seven clock cycles. It takes $\text{tc} \times [5 + 6(\text{ts} - k + 1) + 7]$ clock cycles from the first PE chain first enable to the second enable, while it takes $5 + \text{ts} \times 3(\text{ts} - k + 1)$ clock cycles for the first PE chain to finish computing the first 2D tile. tc and ts are bounded as follows:

$$5 + \text{ts} \times (\text{ts} - k + 1) < \text{tc} \times [5 + 6(\text{ts} - k + 1) + 7]. \quad (1)$$

When $k = 3$, the inequality is rewritten as follows:

$$0 < 6\text{tc} \times \text{ts} - 3\text{ts}^2 + 6\text{ts} - 5. \quad (2)$$

The constructor is as follows:

$$f(\text{ts}, \text{tc})6\text{tc} \times \text{ts} - 3\text{ts}^2 + 6\text{ts} - 5. \quad (3)$$

In order to minimize the interval time of PE chain, $f(\text{ts}, \text{tc})$ should be taken as the minimum value. To facilitate control, tc should be equal to 2^n, considering that there are only 900 DSPs in the FPGA and each PE chain contains 3 DSPs, the relationship between the number of DSPs used. When $\text{tc} = 4$ and $\text{ts} = 9$, the function $f(\text{ts}, \text{tc})$ is greater than 0 and smallest. The theoretical interval of the PE is 22 clock cycles.

The design of this thesis uses 64 PE arrays for parallel computation. When $\text{tc} = 4$, $\text{tm} = 64$, $\text{ts} = 9$, the local buffer occupies at most 5.5 Mb. This paper uses distributed RAM instantiation for memories with larger bit width (48 bits) or smaller space occupancy, such as FIFO, LPB, and LOB.

3.7. Computational Decision for Multiple PE Chains. The convolutional computation unit uses 64 PE arrays, and each PE array contains 4 PE chains. The computation model used in this paper is as follows: the feature map values in each LIB are shared by 64 PE arrays, and each PE array has its own separate LWB, as shown in Figure 13.

The previous PE chain passes the computed partial sum to the next PE chain. Each tc 2D tile is computed as a small cycle (called a 2D_tile_cycle), and each 3D tile takes C/tc small cycles to compute, so the number of clocks ($\text{time}_{\text{tile}}$) needed to finish computing a 3D tile is as follows:

$$\text{time}_{\text{tile}} = \text{tc} \times [5 + 6(\text{ts} - \text{k} + 1) + 7] \times \frac{c}{\text{tc}}. \quad (4)$$

Each 3D tile is a large cycle (called a 3D_tile_cycle), and 64 PE arrays produce 64 output feature maps at the same time. The PE array designed in this paper contains 4 PE chains, and each PE chain corresponds to a channel of the input feature map, and when there are only three channels of the input feature map, the value stored in the fourth LIB is all 0 for the convenience of control.

This computational model has a great improvement for the utilization of computational units. According to the parameters of the neural network model of VGG16, the minimum number of output channels is 64, so it can ensure that all 64 PE arrays can be effectively computed when computing each layer of convolution. For neural networks with less than 64 output channels, we can flexibly control the number of PE array through the controller parameter configuration to reduce unnecessary power consumption.

3.8. Splice Pool. The overall structure of splice pool is shown in Figure 14 (the clock signal and reset signal are omitted in the figure) and contains an internal Pool Buffer and a specially designed comparator. After PE array calculates a 3D tile, it sends a signal to splice pool, and splice pool detects

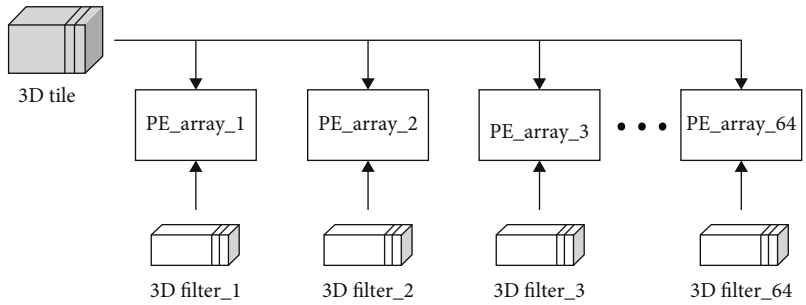

Figure 13: Convolutional unit computation model.

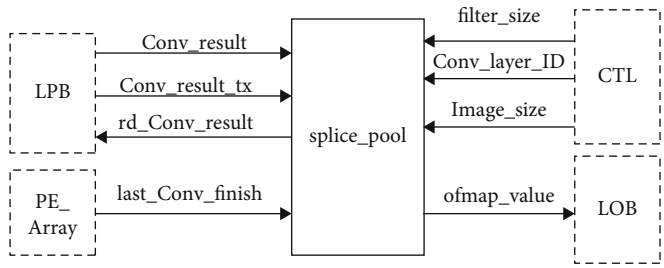

Figure 14: The overall structure of splice pool.

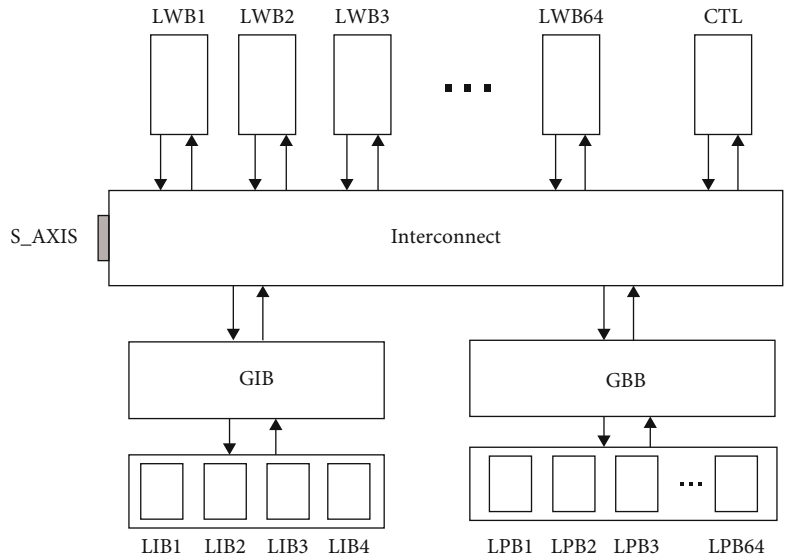

Figure 15: Data interaction of input buffers with the bus.

Table 1: Individual input buffer IDs.

Memory	GBB	LWB1	…	LWB64	GIB
ID	1	2		65	66

the signal and starts to enable it, delays one clock cycle to send a signal to request the convolution result, and after LPB receives the request signal, it sends a transmission signal and delays one clock cycle to send the convolution result. When an output tile is stored, splice pool stops enabling and enters the state of waiting for the next enable. When four output tiles are spliced, the comparator is started and the ready pooling signal is sent to the LOB, and then, the pooling result is output every four clock cycles.

3.9. Memory System. The memory system mainly consists of input buffers and output buffers, both of which are connected to the DMA controller through the AXI4-Stream bus, and interacts with DDR3 by DMA. This avoids frequent data exchange with DDR3 memory and reduces access to memory power. The memory capacity configuration in the coprocessor is combined with the features of Block RAM and Distributed RAM (DistRAM) in FPGAs. The memory capacity

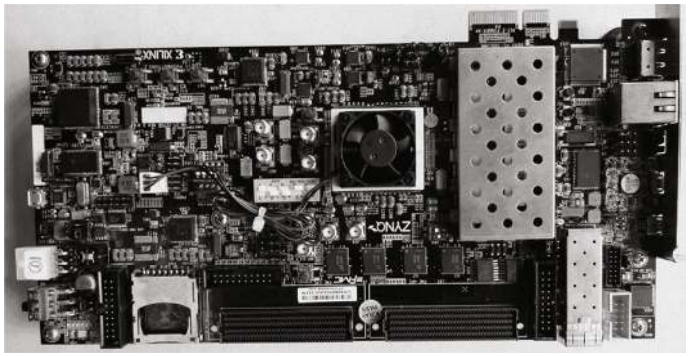

Figure 16: The ZC706 evaluation board.

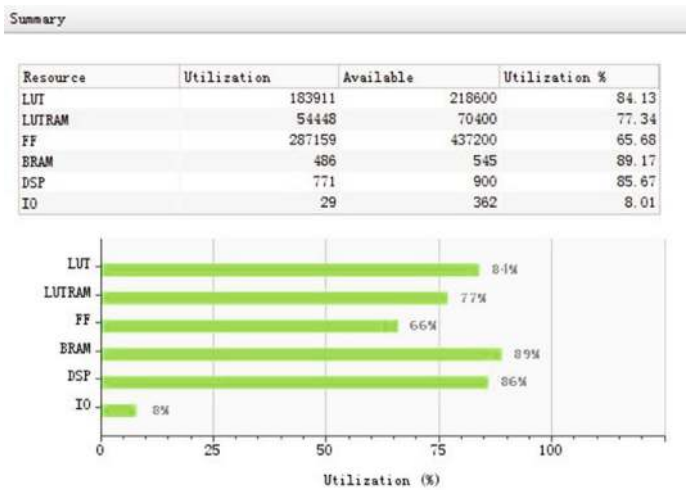

Figure 17: The resource utilization report.

Table 2: The performance statistics of the coprocessor computing VGG16.

Layer	Number of operations (Ops)	Number of clocks (clk_num)	Performance (GOP/s)	Power (W)
Model	9437184	21669	87.10	9.21
Conv1	176518656	471803	74.83	9.21
Conv2	3765731328	33739520	22.32	9.23
Conv3	1916338176	4287402	89.39	9.24
Conv4	3832676352	17419712	44.00	9.23
Conv5	1984167936	2440995	162.57	9.22
Conv6	3968335872	9996224	79.40	9.23
Conv7	3968335872	9996224	79.40	9.25
Conv8	2123366400	3468295	122.44	9.28
Conv9	4246732800	7790023	109.03	9.25
Conv10	4246732800	7790023	109.03	9.26
Conv11	1207959552	3621895	66.70	9.23
Conv12	1207959552	3621895	66.70	9.24
Conv13	1207959552	3621895	66.70	9.23

Table 3: The computational performance comparison results for different platforms.

Layer group	This work (GOP/s) VGG16	[15]	[16]	[17]
Conv_1	23.04	46.42	1578.8	45.15
Conv_2	52.97	80.44	1675.5	69.60
Conv_3	88.45	151.57	2177.1	103.51
Conv_4	111.47	147.91	2791.6	86.04
Conv_5	66.70	186.99	1003.5	36.27

configuration method in this design is flexible based on performance requirements and resource redundancy.

The input buffers include GIB, LIB, LWB, GBB, and LPB. The data interaction relationship is as follows Figure 15. Since all data needs to be input from the axis_S port, each memory connected to the interconnect logic is assigned an ID for efficient distribution, and only the memory with the matching ID can receive the current data. Table 1 shows the individual input buffer IDs. Considering that DDR reads and writes are usually in burst mode, the bandwidth utilization is higher when reading or writing batches of data sequentially and lower when random address reads and random address writes alternate frequently. Therefore, the data

Table 4: The computational performance of the coprocessor of this design compared with other FPGA processor.

Parameters	[18]	[13]	[19]	[17]	This work
Platform	Virtex5SX240t	ZynqXC7Z045	Virtex7VX485t	ZynqXC7Z045	ZynqXC7Z045
Frequency (MHz)	120	150	100	150	200
Quantification	48 bits	16 bits	32 bits	16 bits	16 bits
Power	14	8	18.61	9.63	9.30
Performance (GOP/s)	16	23.18	61.62	187.80	62.54

is stored in DDR in the order of the first layer (bias–filter–feature map), the second layer (bias–filter–feature map), and so on. Therefore, the order of ID change is 1,2,3 .....66, when data is loaded from off-chip memory into the input buffer before each layer of convolutional computation, and the order of ID change after that varies according to the configuration parameters of the controller.

The output buffer consists of tm local output buffer (LOB) with one LOB connected after each pooling unit. 64 LOBs are stored at the same time, and a signal is sent to the controller, and the host reads the output feature values sequentially through the M_axis port.

3.10. Global Controller. The global controller mainly consists of a lookup table and a register configuration unit, which completes the control of each module of the whole coprocessor. The configuration of the registers is done through the AXI4-Lite interface configuration, including nine 32-bit registers, each with corresponding address and description. The global controller starts the coprocessor when it detects an enable signal. In order to use as much DSP as possible for convolution calculation instead of for address or other intermediate parameters, this thesis unifies the parameters that may be used by each module in the controller for calculation and then sends them to each module on demand.

4. FPGA Hardware Verification

The coprocessor designed in this thesis focuses on the accelerated computation of the convolutional layer in the VGG16 neural network model. The coprocessor is verified in hardware based on the ZC706 evaluation board kit, as shown in Figure 16. The evaluation board contains a configurable dual-core ARM Cortex A9 processor and a 28 nm XC7Z045 FFG900-2-based FPGA chip with 54650 Slice (each Slice contains 4 LUTs and 8 flip-flops), 900 DSP cells, and 545 Block RAM cells.

The proposed coprocessor implements parallel computation of 64 PE arrays with the coprocessor resources as shown in Figure 17. 768 DSPs are used for the convolutional computation unit, 2 DSPs for the controller, and 1 DSP for the GIB. In the process of implementation and layout wiring, the "default" implementation strategy is selected, and the clock frequency can reach 200 MHz.

At the clock frequency of 200 MHz, the peak computation capacity of the coprocessor designed in this paper is 316.0 GOP/s. Due to the long loading time of the data, especially in the case of a relatively large number of 3D tiles, the interval time of PE array accumulates relatively large, resulting in a relatively low average computation capacity. Table 2 shows the performance statistics of the coprocessor computing VGG16. In the verification scheme, the ROM of the off-chip memory needs to be instantiated using Block RAM. Therefore, the verification can only have the storage capacity of GIB, so the convolutional layers with more parameter values and activation values cannot be verified. We can extrapolate the convolutional computation time and data loading time based on the verified convolutional layers and then infer the computational performance of the convolutional layers that cannot be verified directly. According to the actual test results in Table 2, it can be found that the total computational power consumption increases slightly with the number of accesses when all DSPs are involved in the computation. The main reason is that the DDR is not accessed during the test, and all data interactions take place between on-chip memory and registers and before registers and DSPs. The difference in the number of accesses results in a slight difference in the total power consumption.

Table 3 shows the computational performance comparison results for different platforms. The CPU is Intel Xeon E5-2690 CPU@2.90GH, the GPU is Nvidia K40 GPU, and the mGPU is Nvidia TK1 Mobile GPU development kit. The convolutional layers between the two pooling layers are called a conv group, and there are five groups. The weights of this design are not processed by any compression, while the paper [12] uses SVD compression, and the weight values are reduced to 36% of the original; nevertheless, the computational performance of the coprocessor designed in this paper is at the same level as the mGPU.

Table 4 shows the computational performance of the coprocessor of this design compared with other FPGA processor. The coprocessor designed in this paper mainly adopts two solutions to achieve higher clock frequency. First, reduce the fan-out, the signal fan-out from LIB broadcast output to 64 PE arrays is 64, and the design reduces the fan-out to 8 by adding intermediate flip-flops. Second, reduce the multiplication calculation except convolution calculation, put all the parameters involved in the controller to calculate uniformly, and then distribute to each module as needed. The computational performance of this design scheme is relatively high compared to the paper [14–16] and much worse compared to the paper [13]. On the one hand, it is because the paper [13] uses the SVD compression algorithm to reduce the amount of weight value to 36% of the original, and on the other hand, this design scheme in the cache read and write processes, we often start the computation process or the transfer operation between memories only after all the read and write processes are completed in order to

simplify the logic, which will cause the computation unit to hang, and the waiting time overhead at this time will become larger as the 2D_tile. The waiting time overhead becomes larger as the number of 2D_tile increases.

5. Conclusion

In this paper, an FPGA-based architecture for convolutional neural network coprocessor is proposed and fully validated, starting from the optimization methods of convolutional neural networks at the algorithmic and hardware acceleration levels. A one-dimensional convolutional computation unit PE with row stationary (RS) streaming mode is proposed to maximize the energy efficiency of convolutional computation by using RF-level data reuse and partial and local accumulation; a three-dimensional convolutional computation unit PE chain with pulsating array structure is proposed with the MIT Eyeriss structure. Each PE chain consists of three PEs connected in sequence, which can achieve the functions that require nine PEs in the Eyeriss structure. Multiple PE chains form a PE array in the form of pipeline processing, and the coprocessor can flexibly control the number of PE array openings according to the number of output channels of the convolutional layer to reduce invalid computation and reduce power consumption. A picture segmentation method is proposed that is compatible with the hardware architecture, and this segmentation method can make the PE chain run with the shortest interval between two runs. The performance evaluation results show a peak computational performance of 316.0 GOP/s and an average computational performance of 62.54 GOP/s at a clock frequency of 200 MHz, with a power consumption of around 9.25 W.

Data Availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Special Fund for the Shenzhen (China) Science and Technology Research and Development Fund (JCYJ20180503182125190) and Shenzhen (China) Science and Technology Research and Development Fund (JCYJ20200109120404043).

References

[1] J. Misra and I. Saha, "Artificial neural networks in hardware: a survey of two decades of progress[J]," *Neurocomputing*, vol. 74, no. 1-3, pp. 239–255, 2010.

[2] F. M. Dias, A. Antunes, and A. M. Mota, "Artificial neural networks: a review of commercial hardware[J]," *Engineering Applications of Artificial Intelligence*, vol. 17, no. 8, pp. 945–952, 2004.

[3] Y.-H. Chen, C.-P. Fan, and R. C.-H. Chang, "Prototype of low complexity CNN hardware accelerator with FPGA-based PYNQ platform for dual-mode biometrics recognition," in *2020 International SoC Design Conference (ISOCC)*, pp. 189-190, Yeosu, Korea (South), 2020.

[4] M. Holler, S. Tam, H. Castro, and R. Benson, "An electrically trainable artificial neural network (ETANN) with 10240 floating gate synapses," in *International Joint Conference on Neural Networks*, pp. 191–196, Washington, DC, USA, 1989.

[5] H. Srivastava and K. Sarawadekar, "A depthwise separable convolution architecture for CNN accelerator," in *2020 IEEE Applied Signal Processing Conference (ASPCON),*, pp. 1–5, Kolkata, India, 2020.

[6] N. Suda, V. Chandra, G. Dasika et al., "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 16–25, New York, NY, USA, February 2016.

[7] J. Cong and B. Xiao, *Minimizing computation in convolutional neural networks[C]//International Conference on Artificial Neural Networks*, Springer, Cham, 2014.

[8] H. Kim and K. Choi, "Low power FPGA-SoC design techniques for CNN-based object detection accelerator," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pp. 1130–1134, New York, NY, USA, 2019.

[9] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: a tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[10] V. Sze, Y. H. Chen, J. Emer, A. Suleiman, and Z. Zhang, "Hardware for machine learning: challenges and opportunities," 2016, http://arxiv.org/abs/1612.07625.

[11] M. Sankaradas, V. Jakkula, S. Cadambi et al., "A massively parallel coprocessor for convolutional neural networks," in *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, pp. 53–60, Boston, MA, USA, 2009.

[12] D. T. Kwadjo, J. M. Mbongue, and C. Bobda, "Performance exploration on pre-implemented CNN hardware accelerator on FPGA," in *2020 International Conference on Field-Programmable Technology (ICFPT)*, pp. 298-299, Maui, HI, USA, 2020.

[13] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 682–687, Columbus, OH, USA, 2014.

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

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, http://arxiv.org/abs/1409.1556.

[17] C. Szegedy, W. Liu, Y. Jia et al., "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, Boston, MA, USA, 2015.

[18] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 247–257, 2010.

[19] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Array*, pp. 161–170, New York, NY, USA, 2015.