Journal of Systems Architecture 167 (2025) 103468

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

FPGA-based CNN accelerator using Convolutional Processing Element to reduce idle states

Mohammad Dehnavi [a,*], Aran Ghasemi [a], Bijan Alizadeh [b]

[a] Department of Electrical Engineering, Kermanshah University of Technology, Kermanshah, Iran
[b] School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran

ARTICLE INFO

Keywords:
Convolutional neural network
Object detection
YOLOv3-Tiny
FPGA

ABSTRACT

Object detection has been a significant challenge in machine vision systems from the past to the present. Various hardware-based accelerators have been utilized to enhance speed efficiency. The primary objective of most of these accelerators is to minimize idle states in DSP blocks. In this paper, a new architecture based on Convolutional Processing Elements (CPEs) is proposed, wherein weights are stored, circularly shifted in an internal CPE buffer and used to generate output feature maps. In this way, the idle states of DSPs are reduced by increasing data reuse in CPEs and decreasing external memory accesses. The number of CPEs used to accelerate a CNN depends on the required speed and available hardware resources; configurations of 16, 32, 64, 128, and 256 CPEs can be utilized to accelerate a desired Convolutional Neural Network (CNN). To demonstrate the effectiveness of the proposed architecture, it is applied to the YOLOv3-Tiny object detection CNN. Experimental results show that our proposed architecture with 128 CPE cores can operate at 62.8 frames per second on an FPGA Xilinx XCKU060 with a working frequency of 200 MHz, using 16-bit fixed-point representation. This approach results in only a 1% drop in mAP while utilizing 43.2K LUTs, 94.4K FFs, 26.73 Mbits of RAM, and 1364 DSPs. Furthermore, the number of external memory chips is reduced by 67% compared to the state-of-the-art systems.

1. Introduction

Object detection using convolutional neural networks has been widely employed in computer vision in recent years, owing to their strong ability to detect complex patterns in images and their advantage in accuracy compared to traditional methods [1,2]. CNN-based object detectors are generally divided into two categories: single-stage detectors and two-stage detectors. In single-stage object detection, models attempt to detect all objects in an image simultaneously using a single detection model. This type of object detection is usually performed with simpler architectures and requires less inference time. Models such as YOLO (You Only Look Once) [3] and SSD (Single Shot Detector) [4] are among the most well-known single-stage object detection algorithms. In the two-stage approach, the object detection process is divided into two stages. In the first stage, potential regions containing objects are extracted by an algorithm. In the second stage, convolutional neural networks operate on these potential regions to detect the objects. Due to its two-stage nature, this method has a slower speed compared to the single-stage method. Some of the most well-known two-stage object detection algorithms include R-CNN [5], Fast R-CNN [6], R-FCN [7], Faster R-CNN [8], and Mask R-CNN [9].

The YOLO algorithm is a single-stage object classification and detection algorithm in the field of computer vision, known for its relatively high speed and accuracy. Convolutional neural network algorithms inherently require significant computational costs, posing a challenge for their implementation on resource-limited hardware [10,11]. While it is feasible to implement these algorithms on CPUs, CPUs often lack sufficient speed for real-time execution. One solution to alleviate this issue is to use GPUs, as they are optimized for parallel computations and generally can execute deep neural networks faster. However, GPUs have high power consumption and are large in size, making them suboptimal choices for embedded systems. In recent years, the use of FPGAs for implementing deep neural networks has gained traction due to their low power consumption, parallelizability of operations, configurability, and small size.

This work presents a new architecture for implementing single stage CNNs, such as YOLOv3-Tiny object detection, with high accuracy on a low-cost FPGA using fixed-point computations. The main contributions of this work are as follows:

- A new architecture based on a new processing element, called the Convolutional Processing Element (CPE), is proposed, in which

* Corresponding author.
E-mail address: M.Dehnavi@kut.ac.ir (M. Dehnavi).

https://doi.org/10.1016/j.sysarc.2025.103468
Received 30 January 2025; Received in revised form 12 May 2025; Accepted 21 May 2025
Available online 6 June 2025

distributed memories are used to buffer weights and feature maps. This architecture benefits from extracting neighboring elements from the feature maps and performing a circular shift of weights to reduce access to external memory.
- Based on the proposed CPE, a new memory access method is proposed that reduces the idle states of CPEs, increases resource efficiency, and minimizes the number of external memory chips.

The rest of the paper is organized as follows: In Section 2 related work is discussed; In Section 3, the background on CNNs hardware accelerators is presented; In Section 4, the proposed hardware architecture is explained in detail; In Section 5 experimental results are reported; the last section concludes the paper.

2. Related work

As we know, convolutional neural networks require significant computations, often involving multiply-accumulate (MAC) operations. General systems commonly utilize GPU-based systems due to their ease of implementation. However, in many mobile systems with power consumption constraints and systems requiring very high speeds but limited size, the use of GPUs is not feasible. Therefore, in recent years, the implementation of CNN-based algorithms on FPGA and ASIC hardware has garnered attention due to their higher speed, lower power consumption, and smaller size requirements [12–15].

Various research efforts have been conducted in this area, each with its specific approach and design aimed at achieving the proposed objectives. Some approaches focus on achieving lower power consumption, smaller hardware resource utilization, and lower cost, while in other cases, computational accuracy has been of utmost importance. Existing methods can generally be categorized based on these two approaches.

In the first category, hardware optimization is achieved by reducing the number of bits in the weights of the target CNN, thereby reducing the computational burden, and utilizing shift operations instead of multiplication. This approach has led to the development of highly optimized hardware architectures in terms of hardware size, albeit at the cost of reduced accuracy. For instance, in [16], an architecture for MNIST networks was proposed and implemented in a binary format. However, due to the use of only one-bit precision, the system's accuracy reduced by about 20 percent. In [17], an in-memory bit-shifting technique for the quantized weights of CNNs is proposed to reduce memory accesses.

In [18], authors presented a network architecture based on ZYNQ MPSOC, employing mixed-precision weights with 5-bit and 8-bit parts, as well as 7-bit and 8-bit precision for the accumulation part. This reduction in bits has resulted in approximately a 10 percent decrease in accuracy compared to the baseline. However, the proposed architecture offers configurability, enabling simultaneous configuration with operation, albeit at the cost of considerable hardware utilization.

In [12], an architecture for optimizing access to external memory was proposed. Nevertheless, to reduce the computational load, 8-bit precision for weights and 16-bit precision for addition and other computations were used. Moreover, a high utilization of hardware resources is observed, with 80 GOPS being achieved. The authors of [14] presented an optimized architecture for computing CNN, utilizing the Winograd method instead of direct convolution calculation. In this architecture, the computational precision is limited to 8 bits. These two optimizations have resulted in a 3 percent reduction in accuracy compared to the baseline, while still providing an acceptable hardware volume.

In [19], authors proposed an architecture for CNN utilizing 1-bit weights and 2-bit computations. This approach has led to higher speed, lower hardware volume, less energy consumption, and reduced memory access. In their recent work [15], they increased the number of computational bits to 8 bits and used a DSP48 block for calculating two multiplications. These changes have led to a nearly 50% reduction in DSP48 block utilization and reduced memory access with a suitable architecture provided. Despite the increase in the number of bits in computations, a relatively high difference in accuracy compared to the sample with full accuracy is still observed. In [20], Zhang and colleagues presented an implementation of YOLOv3-Tiny with 8-bit accuracy. This implementation has improved upon previous systems by reducing access to external memory and utilizing relatively acceptable hardware. Only four DDR3 memory chips are used in their system, and its accuracy dropped by 2 percent. In [21], 4-bit precision has been used for weights and CNN computations, implementing multiplication operations implemented using LUTs. These methods generally exhibit acceptable speed, power consumption, and hardware volume. However, due to the approach of reducing the number of bits and using few-bit accuracy, system accuracy has been reduced.

In the second category, the approach of articles has been based on relatively preserving the accuracy of the CNN, often considering 16 bits for computations and weights, and designing the CNN accordingly. For example, in [22], a precise architecture based on floating-point computations was proposed, consuming 2800 DSP blocks even for a small network. This architecture has been presented with a staged and hierarchical processing approach, mostly used in software system architectures. While this architecture has high accuracy, its hardware volume is often not acceptable for many applications. In [23], a hardware/software-based system was proposed where only the convolutional filter application part is implemented in hardware, and the rest is done in software. This article has used 18-bit precision for computations, providing sufficient accuracy but consumes high resources for storing and retrieving feature maps.

In [24], an architecture for implementing CNN on FPGA has been presented, optimizing access to external memory. However, the required internal resources have also increased. In [25,26], trained weights are stored in on-chip memory to reduce the need for external memory bandwidth. However, as the convolutional neural network grows larger, the weights also increase, making it impractical to store all network weights in on-chip memory.

Another approach, used in [27], stores intermediate results in on-chip memory and weights in external memory, yet this system also has memory limitations. In [28], the layers are executed sequentially, and the result of each layer is stored in DRAM. In this architecture, only after the current layer is fully ready, the next layer can begin its operation. This non-pipelined structure decreases processing speed. In [29], this issue has been addressed, allowing the start of the next layer operation without fully completing the current layer, improving processing speed. However, it requires 360K LUTs of FPGA resources.

Some implementations have utilized floating-point computations [22,30]. However recent research has shown that floating-point representation is not necessary [29]. Although floating-point computations lead to higher accuracy, they also require more hardware resources. Additionally, using fixed-point computations in systems with limited resources improves processing speed because they eliminate the need for complex arithmetic operations. This feature is crucial in applications that require fast processing.

In another approach, CNNs are optimized for hardware implementation. For example, SparkNOC, proposed in [31], enhances SparkNet for this purpose. Similarly, a framework introduced in [32] quantizes and maps CNNs from the Paddle-Lite library onto the FPGA platform. Table 1 summarizes the various hardware optimization approaches for CNN architectures, along with their strategies and the platforms used.

The main challenge that all of the above systems have aimed to address is reducing access to external memory for loading weights and feature maps and reducing the required hardware resource for convolution calculation. Next, a clear background of this challenge is provided, followed by a detailed explanation of a proposed system to address this challenge.

Table 1
Summary of hardware optimization approaches for cnn architectures.

Ref/FPGA	Approach	BP[a]	AD[b]
[16]/Stratix-V 5SGSD8	Binary networks with weights quantization	1	20%
[17]/ASIC	In-memory bit-shifting	(8,4)	–
[18]/ZCU102	Configurable architecture	(5,8)	10%
[12]/Zynq XC7Z035	Optimized external memory access	(8,16)	3%
[14]/ZCU104	Winograd method optimization	8	3%
[15]/KCU1500	Use DSP48 for two 8-bit multiplications, row-level parallelism	8	2%
[20]/Kintex XC7K325T	Reduced external memory access	8	2%
[21]/Zynq XCZU9EV	4-bit Quantization, LUT-based multiplication	4	3%
[23]/VC707	Hardware/software codesign	18	1%
[24]/Zynq XC7Z020	Reduced external memory access	–	–
[29]/Intel Arria 10	Reduce layers wait	–	–

[a] Bit precision.
[b] Accuracy drop.

3. Background

CNN systems involve a large number of multiplication and addition operations, leading to the generation of a significant amount of data that must be stored and retrieved. For instance, a small network like YOLOv3-Tiny has approximately 8 million parameters, and if implemented in a streaming mode, it requires 8 million addition and multiplication operators. However, FPGAs do not have sufficient resources to accommodate such a large number of arithmetic units. Therefore, a limited number of adders and multipliers are utilized for the implementation of these systems, and these resources are reused across different computation sections. This is due to the reduction in feature map dimensions in the final layers (caused by pooling layers) and the increase in processing frequency compared to the input video frequency. Consequently, most studies use external memory to process the network layer by layer [12,20,21], as shown in Fig. 1. In this architecture, the Processing Element (PE) consists a single multiplier, and the feature maps and filter weights for these PEs are cached in internal memory. In this architecture, convolution is performed first, and the resulting feature maps are stored in external memory, then retrieved to perform batch normalization and max-pooling. This requires two cycles of reading from and writing to external memory.

The main bottleneck of the existing architecture is the many idle states between operations that reduce overall system efficiency. Idle states refer to the percentage of idle time of the processing elements, which consist of LUTs, DSPs, and RAMs. In some cycles, processing elements stop computation and wait for new data, resulting in idle time for the system. The idle states are due to the following bottlenecks in the architectures:

- The first cause of the idle state is the external memory bandwidth during random access reads to SDRAMs and data reuse in PE units.
- The idle states arise when batch normalization and max-pooling are performed in this architecture. PE units are idle and wait for batch normalization and max-pooling to complete before starting a new convolutional layer.
- The idle states may occur in the convolution with a stride greater than one, during which some PE units are idle and wait for new valid data.
- The idle state may occur in convolutions of size 1×1, where some PE units remain unused.

In this paper, a new architecture is proposed to reduce idle states in the processing elements, improve data reuse, and decrease external memory bandwidth.

4. Proposed architecture

Convolutional Neural Networks (CNNs) typically consist of several main layers, including convolutional layers, max-pooling, batch normalization, and activation functions. Based on this structure, the overall

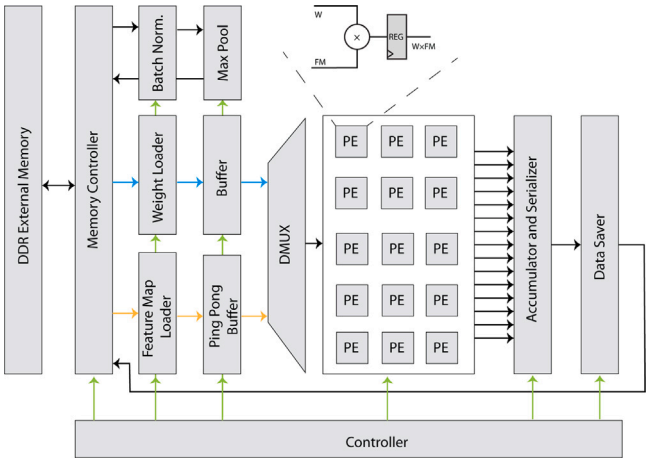

Fig. 1. The CNN architectures proposed by the state-of-the-arts [12,20,21].

design of the proposed architecture is shown in Fig. 2. The input image is received from the camera and stored in external memory. After receiving a new frame, computations begin. The feature maps are loaded by the Feature Map Loader, and the filter weights are loaded by the Weight Loader into CPEs. After applying each filter to the input feature map, results are accumulated, batch normalization is applied, and max-pooling is performed. Finally, the results are stored in external memory using a Data Saver. As shown in Fig. 2, multiple FIFOs are used for storing and retrieving data from external memory. These FIFOs serve as data caches, allowing external memory to store and retrieve data from different components in a sequential manner. The size of each FIFO is determined based on the amount of input and output data to ensure that no component experiences delays in storing or retrieving data. The details of this architecture are described in the following:

Convolutional Processing Element: In the proposed architecture, the Convolutional Processing Element (CPE) is used to apply each kernel. In each CPE, the convolution operation is performed for one 3×3 filter channel in the corresponding input channel. Here, 32 CPEs are used to apply 32 filter channels to the input FM simultaneously. The number of CPEs can be adjusted depending on the resources available in the FPGA and the required speed. All CPEs operate simultaneously and require both the filter weights and the input feature maps (FMs). The filter weights are provided by the Weight Loader, and the input image or FM is provided by the Feature Map Loader to the CPEs.

As mentioned before, one of the contributions of this work is to reduce the idle states for loading data and utilize DSP resources more effectively to increase system efficiency. For this purpose, the proposed structure for a CPE is illustrated in Fig. 3(a). In this architecture, a 3×3 neighborhood from a channel of the input feature map (FM) is extracted using the Neighborhood Extractor (NE) structure, and the

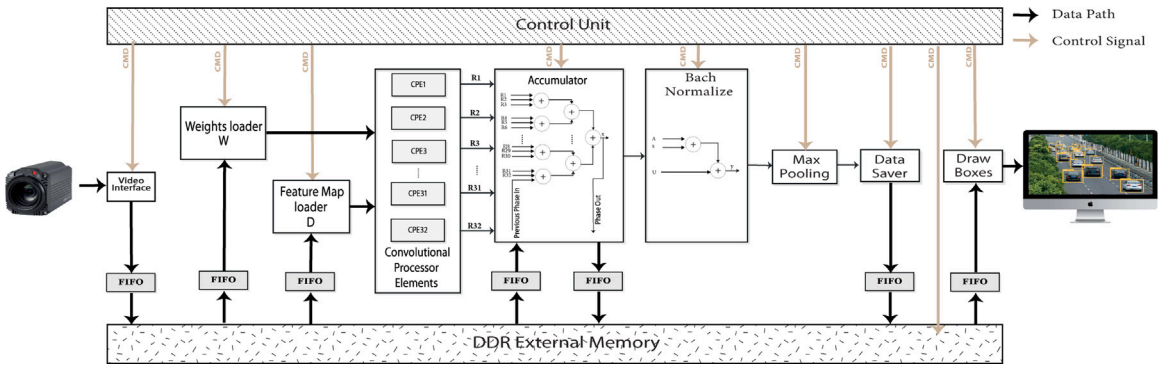

Fig. 2. Overall proposed architecture.

necessary weights are stored in the BRAM memory within the CPE. In this case, each extracted neighborhood will remain unchanged until all weights are applied, which is achieved by halting input to the NE. The line buffers of the NE reuse data in subsequent lines and pixels. The weights' line buffers reuse data for different filters and avoid reading new data.

After applying all weights, a new neighborhood is obtained by loading a new input to the NE, and the above process is repeated until all weights are applied to the input FM. In this structure, all weights are first loaded into the internal memory block once, and then in each clock cycle, a filter is applied to the neighborhood extracted in each CPE. After multiplying the filter weights with the input, the results are aggregated together to obtain the output of a channel (Fig. 3(a)).

If we assume that in a layer, 64 filters with 16 channels are to be applied, these 64 filters are divided into two phases with 32 filters each. CPEs one to sixteen are used for the first phase, and CPEs seventeen to thirty-two are used for the second phase. For the first phase, the filter loader loads all the weights of the first channels of these 32 filters into the Block RAMs of CPE1. The remaining weights are loaded in a similar manner into the other CPEs. For instance, the weights of the sixteenth channels of these 32 filters are sent to CPE16. For the second phase, the same mechanism is used. Since one filter is applied to the extracted neighborhood at each clock cycle, 32 clock cycles are required to apply all the filters.

According to Fig. 3(a), nine multiplications and eight additions are required to apply one 3×3 kernel. The architecture in Fig. 3(b) minimizes the number of DSP48 units required for these computations that only use ten DSP48. As observed, the registers within the DSP48 units are programmed in a way that allows the multiplication results to be aggregated within the DSP48 units, reducing their number to a minimum. In this architecture, ten DSP48 units are required in each CPE.

Some of the kernels have a 1×1 dimension, which is used in some layers. For these kernels, several multiplexers are used, as shown in Fig. 3(a). The datapath of input data for this purpose is shown in red in this architecture. As shown, in 1×1 mode, FMs are fed to CPE registers using I1 to I9 and the related multiplexers. In this mode, the neighborhood extractor is not used. The number of CPEs can vary according to the system's required speed, the speed and quantity of available external SDRAM memory, and the available hardware resources, ranging from 16, 32, 64, to 128 units.

To clarify it, let us consider data-flow of the proposed hardware as shown in Fig. 4. As you can see in this figure, the feature map (FM) in external memory is divided into two or more groups. Each group read in raster scan format, and then fed into the input FIFO. A demultiplexer then distributes the data among different CPEs, each responsible for the computations of a single channel. Each CPE performs a complete 3×3 convolution using 10 DSP48 units, consisting of multipliers and adders. In these CPEs, only two rows of the input FM are buffered, and the results are generated and stored in external memory in raster scan format without data reordering.

In [20], many lines are buffered, and a large multiplexer is used for mapping these data to multipliers (here PE). The input FM is divided into smaller groups with dimensions N, R, C, and they perform parallelism in rows and depth. These groups need to have some overlaps due to data dependency. This type of grouping causes address discontinuities and also requires a wider multiplexer. As row-level parallelism increases, address discontinuities increase, and a wider multiplexer is also needed. Therefore, these problems cause an increase in external memory bandwidth and the number of LUT resources used.

In the proposed architecture, however, the buffers are embedded in CPEs, and this structure reduces the need for multiplexers, with parallelism performed only at the channel (R) level. In this manner, data is read from and written to external memory as a stream with minimal address discontinuity. As shown in Fig. 4, max-pool are executed in the datapath without any storage or retrieval of data from external memory. As mentioned, the number of these groups depends on the number of available CPEs.

Accumulator: After applying various filter channels, the output of the channels needs to be aggregated to compute the final result. When 32 CPE units are used, and considering tree-based accumulation, it is feasible to use the outputs in each section and also to aggregate three values in each DSP48 block. Aggregation of three values is performed in the first stage of the tree adder, and in the next stages, each DSP48 block is used to aggregate two values. Thus, the number of DSP48 blocks used in this section will be 23. In some layers where the number of filters exceeds the number of CPE units, computations are performed in multiple phases, and normalization and pooling layers are not applied until all computations are completed. In these cases, the output results of the ACCUMULATOR are aggregated with the previous phase and stored in external memory. The last phase is an exception and is not stored in external memory. In the last phase, batch normalization and max-pooling are applied and then stored in external memory with the FM saver for processing the next layer.

Batch normalization: The normalization process is straightforward and simplified as Eq. (1):

$$y = Ax + u \tag{1}$$

where A and u are two constants [20]. This equation is straightforward and can be implemented with a single DSP48.

Max pooling: In the process of selecting the maximum value, the data pattern must remain unchanged for storage in memory compared to the read pattern. This is a straightforward block, and its implementation requires only three BRAMs.

5. External memory interface

The way data is stored and retrieved plays a significant role in the efficiency of the proposed architecture. This section discusses the

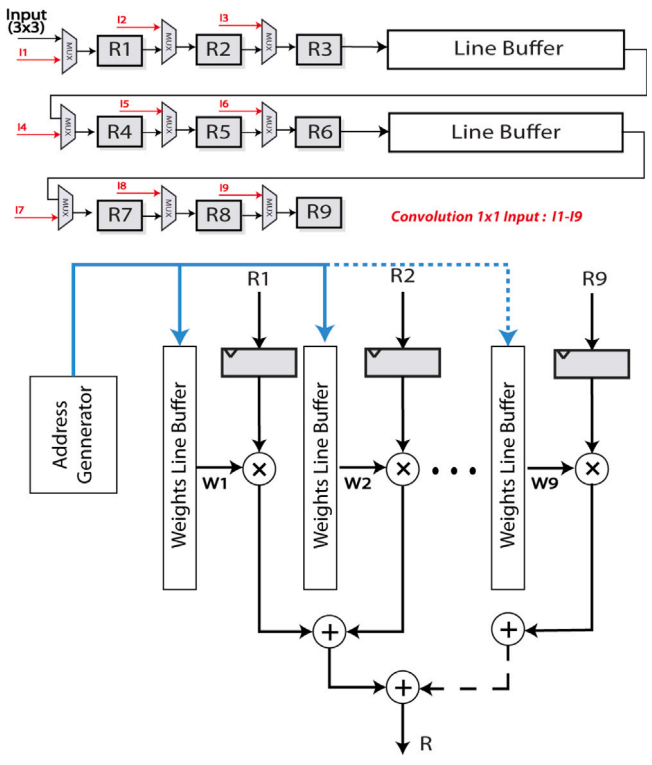

Fig. 3. The proposed architecture (a) a CPE architecture (b) multiply-accumulation (MAC) operations Using the AMD DSP48 in CPE architecture.

method of storing and retrieving data from external memory for different layers. Additionally, the approach to applying filters in this context is presented. Table 2 lists the layers of the YOLOv3-Tiny network, along with the number of filters and channels for each layer. As observed, most layers follow a similar pattern. In this section, a representative layer from each pattern is selected, and the method of data retrieval and storage in memory is explained. This approach can be readily extended to other layers.

Layer One Operations: In this layer, 16 filters are applied to the input RGB image, which has dimensions of 416 × 416 × 3. Since the input image has 3 channels, the filters in this layer also have 3 channels. Thus, there are 48 filter channels that must be applied to the input image. This layer is an exception in the proposed structure because the number of filter channels is odd. In this layer, only CPEs 1 to 24 are used to apply the 16 filters. Each CPE process two filters. The method of applying the filters is illustrated in Fig. 5. After applying the filters, the results are accumulated in the Accumulator, and every two clock cycles, the results for one pixel from the feature map (FM) in each channel are produced. In this figure, the input image is read in raster scan mode, and the neighborhoods are formed in CPE buffers.

Layer four Operations: In this layer, 128 filters with 64 channels are applied. Unlike previous layers, the number of input FM channels is greater than the number of CPEs. The input FM channels are divided into different groups (phases). In each phase, the input FM with a specific number of channels is read from memory, filters are applied, and the results are stored in external memory. Layer four of YOLOv3-Tiny has 64 channels, and 128 Filters are applied. This layer is used as an example in this section, with its operations shown in Figure 6. As shown, this layer is divided into two groups and completed in two phases. In the first phase, 32 channels of filters and 32 channels of input FM are convolved and then stored in SDRAM. In the second phase, the remaining 32 channels of the filters and FM are convolved and added to the result of the previous phase, which is read from external memory, in the Accumulator.

The result, after batch normalization and max-pooling, is stored in external memory. In this layer, after extracting a 3 × 3 neighborhood in the CPE with cyclic shift, 128 filters are applied. Then, with the arrival of a new input, a new neighborhood becomes available, and the process of shifting and convolving the filters is repeated. With this data-flow, the addresses of reads and writes are contiguous (see Fig. 6).

6. SDRAM memory bandwidth

A key factor in the efficiency of the proposed system is the external memory bandwidth. Insufficient bandwidth slows down the system, while excessive bandwidth results in unnecessary resource usage. In this section, the worst-case external memory bandwidth is estimated based on the specifications of the proposed architecture. The external memory utilized is of the SDRAM type, which boasts high sequential read and write speeds but exhibits low speed in random read and write operations. The proposed architecture operates in a manner that receives data sequentially and transfers the results to the output for writing to memory while preserving the data pattern. In layers with one phase, data is read in a completely sequential manner. However, in layers that necessitate two phases for the complete execution of convolution, a gap exists between read addresses, and no-gap exists between write addresses. Moreover, the results from the previous phase must be both written and read. In layers encompassing four phases, there are three gaps between read addresses. The same process is applied to layers with additional phases. Let us assume a layer possesses F filters with CH channels. We can now calculate the required number of SDRAMs as follows:

According to the proposed architecture, for a 3 × 3 kernel, every F clock cycles necessitate the presence of one data at the input of each CPE. Given that there are N_C CPE cores, $\frac{N_C}{F}$ data must be loaded into these CPEs. Following processing, one FM is produced at the output with each clock cycle, necessitating storage in external memory. In the most demanding scenario, the result from a previous phase is read for accumulation with that of the current phase. Considering our FMs are 16 bits each, the required number of bits per clock cycle (N_{bc}) in the most critical phase is calculated as Eq. (2):

$$N_{bc} = 16 \times (1 + 1 + \frac{N_C}{F}) \tag{2}$$

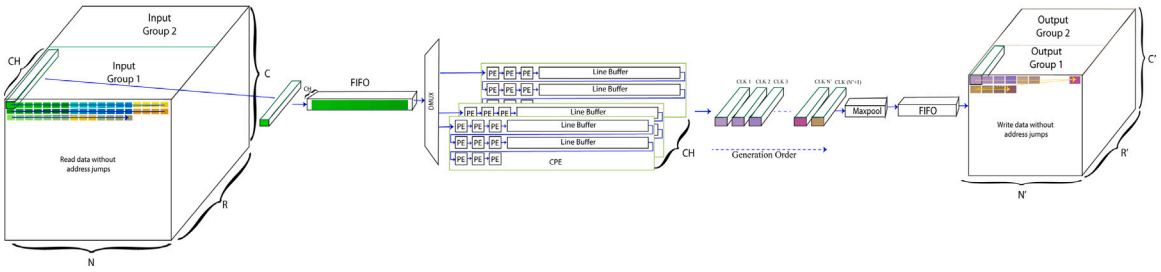

Fig. 4. The data-flow of the proposed architecture.

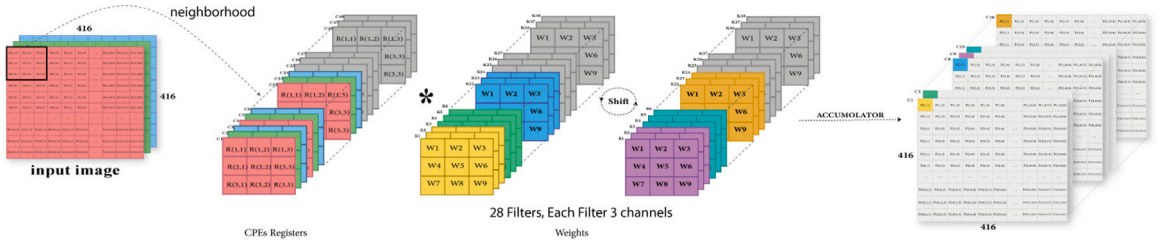

Fig. 5. The read and write data from external memory and perform convolution on them for layer one.

Table 2
The YOLOv3-Tiny convolution layers details and number of phase and shift according to 32 CPEs.

Layer	Filter	Input	Output	Weights Shift in CPEs	Number of phase
Conv1_3 × 3	16	416 × 416 × 3	416 × 416 × 16	2	1
Conv2_3 × 3	32	208 × 208 × 16	208 × 208 × 32	16	1
Conv3_3 × 3	64	104 × 104 × 32	104 × 104 × 64	64	1
Conv4_3 × 3	128	52 × 52 × 64	52 × 52 × 128	128	2
Conv5_3 × 3	256	26 × 26 × 128	26 × 26 × 256	256	4
Conv6_3 × 3	512	13 × 13 × 256	13 × 13 × 512	512	8
Conv7_3 × 3	1024	13 × 13 × 512	13 × 13 × 1024	1024	16
Conv8_1 × 1	256	13 × 13 × 1024	13 × 13 × 256	256	4
Conv9_1 × 1	128	13 × 13 × 256	13 × 13 × 128	128	1
Upsample	–	13 × 13 × 128	26 × 26 × 128	–	–
Concat	–	26 × 26 × 256 and 26 × 26 × 128	26 × 26 × 384	–	–
Conv10_3 × 3	512	13 × 13 × 256	13 × 13 × 512	512	8
Conv11_3 × 3	256	26 × 26 × 384	26 × 26 × 256	256	12
Conv12_1 × 1	255	13 × 13 × 512	13 × 13 × 255	255	2
Conv13_1 × 1	255	26 × 26 × 256	26 × 26 × 255	255	1

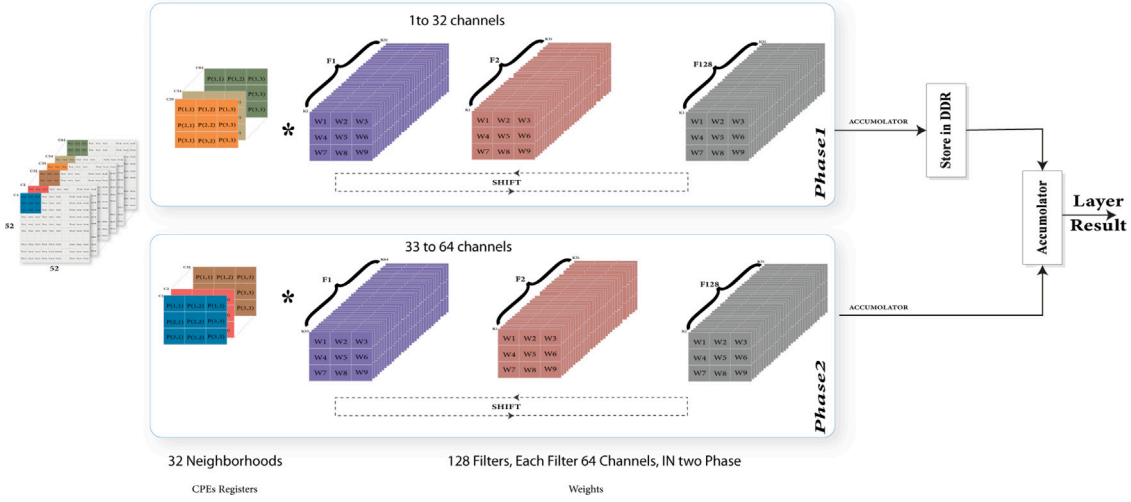

Fig. 6. The read and write data from external memory and perform convolution on them for layer four.

Now, let us consider that we have an SDRAM of DDR3 type with a 16-bit bus width, and its operating frequency is 800 MHz. In this case, an SDRAM needs to provide 1600×16 bits of data per clock if a sequential sequence of data is read. If there is a data discontinuity, we can consider that in the most critical case, the delay in reading each data is equal to the address discontinuity. This discontinuity occurs only for loading data to the CPE cores. Therefore, Eq. (2) can be rewritten as follows:

$$N_{bc} = 16 \times (1 + 1 + \frac{N_C \times J}{F}) \tag{3}$$

where J is the number of address jumps between reads. J depends on the number of channels and the number of cores and can be formulated as follows:

$$J = \frac{CH}{N_C} \tag{4}$$

Replace it in Eq. (3) and have

$$N_{bc} = 16 \times (1 + 1 + \frac{CH}{F}) \tag{5}$$

According to the above equations, the required number of SDRAMs (N_S) is also calculated as follows:

$$N_S = ceil(\frac{N_{bc} \times f_c}{1600 \times 16}) \tag{6}$$

where f_c is CPE core frequency. By simplifying this equation we have:

$$N_S = ceil(\frac{16 \times (2 + \frac{CH}{F}) \times f_c}{1600 \times 16}) \tag{7}$$

Eq. (7) is calculated for 3×3 convolutions, but the critical external memory bandwidth is needed when 1×1 kernels are applied. As shown in Fig. 3(a), eight channels of input FM are loaded to one CPE for 1×1 kernels, which increases the data requirement per clock cycle by eight times. As a result, CH is replaced with $CH \times 8$ in Eq. (7), and we obtain Eq. (8).

$$N_S = ceil(\frac{16 \times (2 + \frac{CH \times 8}{F}) \times f_c}{1600 \times 16}) \tag{8}$$

Eq. (8) shows that the number of external SDRAMs depends on the number of bits used in computations, the number of channels, and their operating frequency. The number of CPE cores, denoted by N_C, does not affect the number of required SDRAMs for continuous data processing. This equation shows the worst case of DDR chips used in the proposed hardware architecture, and in practice, fewer than this number may be required.

The required memory bandwidth is calculated for the loading of feature maps as described above. For weights, similar calculations and considerations are performed to increase DSP efficiency. The weights are written to external memory and read in streaming mode as required. Therefore, the required cycle is the same as the number of weights. As known, for the YOLOv3-Tiny network, about 10M weights need to be loaded onto the system. To avoid reduced DSP efficiency, when layer n is running, the weights of layer $n + 1$ or the next phase are loaded simultaneously. One SDRAM memory is needed for loading weights simultaneously with layers running.

7. Experimental result

In order to completely evaluate the proposed architecture, it was implemented on the FPGA hardware platform for different configurations, but with 16-bit precision. The YOLOv3-Tiny network was selected as the preferred object detection network that was implemented on hardware and evaluated; however, the proposed system can also be used for other similar convolutional neural networks. This system was implemented on the XC7K325T chips using Vivado 2020.2 software with N_C set to 32, and two DDR3 external chips were utilized. For other configurations, another chip with more resources was used to evaluate the proposed system. The Xilinx memory interface generator (MIG) is used for interfacing with external memory. The proposed system can operate at 200 MHz.

The resources used by the proposed system for different configurations and all modules are shown in Table 3. As shown, the resources used for CFG1, in which N_C is equal to 32, are 9.45 Mb of RAM, 370 DSP blocks, 27.2K flip-flops, and 16.5K LUTs. The resource utilization for other configurations is also shown. It must be mentioned that the reported resources for CPE are for a single CPE. Therefore, most of the DSP blocks are used to implement multiply-accumulate (MAC) operations on CPEs, and most of the RAM is used in the weights buffers of the CPEs as well. Some modules are not affected by the quantity of CPEs, and their resources are fixed regardless of changes in the number of CPEs. In the DDR Controller module, 117 BRAMs are used for the implementation of FIFOs that cache data for storage and retrieval from external memory.

The required time for loading weights onto CPE distributed memories for different configurations is listed in Table 4. In CFG1, nine weights of one CPE are read in two transactions, but in CFG2 and CFG3, the nine weights are read in one transaction. In this table, the number of clock cycles for loading feature maps and performing convolution is listed. As mentioned, the weights are read from external memory and stored in internal CPE memories simultaneously with the convolution of the previous layer. Therefore, the critical time for the system is the convolution time of the layers. The convolution times of layers depend on the number of CPEs, and with an increase in CPEs, both the layer time and the total time will decrease.

In this system, fixed-point calculation and 16-bit precision are used for feature maps, 16 bits are used for filter weights, while the original YOLOV3-Tiny network is implemented using floating-point precision in software. This necessary optimization decreases the mean Average Precision (mAP) of the YOLOV3-Tiny network by 1%, but it is inevitable due to the high resource requirements of hardware implementation for these networks.

In Table 5, a comprehensive comparison between the proposed method and state-of-the-art methods is presented. For a fair comparison, only methods that have implemented YOLOV3-Tiny are included in this comparison, and the proposed system with various configurations is presented. The main parameters in this table are FPGA resources, accuracy drop according to mAP, Giga Operations Per Second (GOPS), power consumption, the number of DDR chips (N_S), and system performance. We can calculate the number of DDR chips according to Eq. (8). The worst case corresponds to layer $Conv8\_1 \times 1$, where CH is 1024 and F is 256. Therefore, N_S is equal to 5 for DDR3 and 3 for DDR4 technology. In practical tests, the required number of DDR chips is obtained to be less than the worst case, as shown in Table 5.

There are many parameters in Table 5 for comparison, which makes a fair comparison challenging. Some papers consider GOPS as a figure of merit; however, GOPS depends on FPGA technology, occupied resources, and available resources. Others consider DSP usage (DSP efficiency) as a figure of merit, but some papers use LUTs for computation, which decreases DSP usage. It should be noted that frequency depends on technology, GOPS/DSP depends on LUT resources and other resources rather than DSP alone, power consumption depends on technology and resource usage, and FPGA occupied resources depend on precision. We need to define a new figure of merit that considers all parameters and does not depend on available FPGA resources and FPGA technology. The proposed figure of merit, known as system performance (SP), is defined as Eq. (9).

$$SP = \frac{GOPS}{MD \times f \times (0.49L + RM + D + 0.49FF)} \tag{9}$$

where MD represents the drop in mAP, expressed as a percentage. f denotes the frequency of the system. L, RM, D, and FF represent the percentage usage of LUTs, RAMs, DSPs, and Flip-Flops, respectively. These resource percentages are based on the available resources of an XC7K325T for all systems listed in Table 5. In this equation, a weight of

Table 3
Resource utilization of proposed system for 16-bit precision and different configuration.

Module	CFG1 (32 CPEs)				CFG2 (64 CPEs)				CFG3 (128 CPEs)			
	LUT	FF	RAM (18 KB)	DSP	LUT	FF	RAM (18 KB)	DSP	LUT	FF	RAM (18 KB)	DSP
DDR controller	1845	972	117	0	2200	1350	117	0	1845	972	117	0
Weight loader	242	368	0	0	242	368	0	0	242	368	0	0
FM loader	910	882	0	0	2120	1730	0	0	3011	2753	0	0
CPE	232	672	10	10	232	672	10	10	232	672	10	10
Accumulator	1166	381	0	17	2452	783	0	34	3537	1254	0	51
Batch norm.	1283	951	36	25	1283	951	36	25	1283	951	36	25
MaxPool	3192	1512	52	8	3192	1512	52	8	3192	1512	52	8
DataSaver	412	575	0	0	412	575	0	0	412	575	0	0
Total system	16,474	27,145	525	370	26,749	50,277	845	707	43,218	94,401	1485	1364

Table 4
The YOLOv3-Tiny layers processing time using the proposed system.

Layer	Filter	Filters weights clock cycles	CFG1 (32 CPEs)	CFG2 (64 CPEs)	CFG3 (128 CPEs)
			Feature maps clock cycles	Feature maps clock cycles	Feature maps clock cycles
Conv1_3 × 3	16	48	349,448	174,724	174,724
Conv2_3 × 3	32	512	705,600	352,800	176,400
Conv3_3 × 3	64	2048	719,104	359,552	179,776
Conv4_3 × 3	128	8192	746,496	373,248	186,624
Conv5_3 × 3	256	32,768	802,816	401,408	200,704
Conv6_3 × 3	512	131,072	921,600	460,800	230,400
Conv7_3 × 3	1024	524,288	3,686,400	1,843,200	921,600
Conv8_1 × 1	256	29,128	173,056	86,528	43,264
Conv9_1 × 1	128	3640	21,632	21,632	21,632
Upsample	–	0	0	0	0
Concat	–	0	0	0	0
Conv10_3 × 3	512	131,072	921,600	460,800	230,400
Conv11_3 × 3	256	98,304	2,408,448	1,204,224	602,112
Conv12_1 × 1	255	14,507	86,190	43,095	43,095
Conv13_1 × 1	255	7254	172,380	172,380	172,380
Total		982,833	11,714,770	6,054,391	3,183,111

Table 5
Comparison of the proposed system with the state-of-the-arts.

	[20]	[20]	[14]	[13]	[12]	[33]	This work	
	8-bit	16-bit					CFG2	CFG3
Year	2023	2023	2022	2023	2021	2022	2024	2024
Network	YOLOv3-Tiny	YOLOv3-Tiny	YOLOv3-Tiny	YOLOv3-Tiny	YOLOv3-Tiny	YOLOv2-Tiny	YOLOv3-Tiny	YOLOv3-Tiny
Dataset	VOC	VOC	ILSVRC2015	COCO2024	VOC	COCO2024	VOC	VOC
Precision (bit)	8	16	8	8	8	16	16	16
Clk (MHz)	200	200	200	150	100	200	200	200
Platform	XC7K325T	XKU060	XC7K325T	XCVU9P	XC7Z035	XC7Z045	XC7K410T	XCKU060
BRAM (18 KB)	553	1106	326	4073	924	**226**	845	1485
DSP48	687	1382	146	**96**	924	448	707	1364
LUT(k)	136.5	273	123	132	88k	99.4	**26.7**	43.2
mAP drop (%)	2	**1**	3	2	1.5	2	**1**	**1**
GOPS	401	386	348	351	80.7	138.8	216	**408**
Power (W)	12.3	22.3	**3.3**	5.52	–	4.55	4.5	8.1
DDR Chips	4-DDR3	8-DDR4	–	–	–	–	2-DDR3	3-DDR4
SP[a]	54.8	52.3	57.97	23.8	23.29	35.9	57.66	**59.82**

[a] System performance.

0.49 is added for Flip-flops and LUTs. This weight arises from the fact that each 16-bit multiplication requires 337 LUTs. If we use all the LUTs of the XC7K325T as multipliers, approximately 604 multipliers would be available. This number of multipliers corresponds to about 72% of the available DSP48s. If these LUTs are used as memory, they can also provide approximately 25% of the block memories of the XC7K325T. Therefore, the average of these percentages is used as the weight for LUTs. For Flip-flop usage, based on the results of various works, each LUT is typically accompanied by two Flip-flops. As a result, the weight used for LUTs is also applied to Flip-flops. As shown in the Table 5, the proposed architecture demonstrates good system performance across different configurations. Furthermore, one of the main resources is the number of required external memory chips, which affects the power consumption of the final system.

The authors in [20] implement two optimizations to enhance the performance of their architecture. First, they use 8-bit precision for both feature maps and weights, employing a single DSP48 to multiply one weight with two feature maps. Second, they utilize an array of processing elements (PEs) fed by switchable line buffers that store feature maps and weights. This architecture requires numerous external memory chips to reduce idle states and load the necessary data into the line buffers. Additionally, the max-pooling layer and batch normalization have separate data paths from the convolution data path. During this process, the convolution data path remains idle, waiting for the completion of max-pooling and batch normalization, which reduces system performance. For a fair comparison, this architecture was extended to 16-bit precision, and the results are presented in Table 5. As shown, this architecture requires a significant number of LUTs.

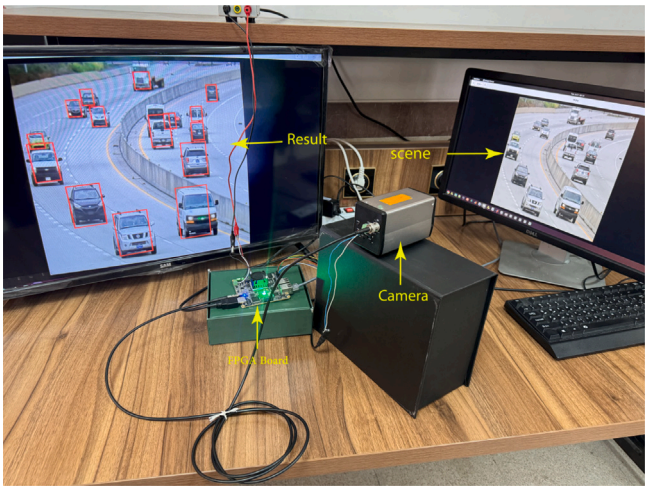

Fig. 7. Practical test of the proposed system.

Another challenge is its high power consumption, primarily caused by the extensive use of external memory.

In [14], the Winograd algorithm is utilized to compute convolutions more efficiently, but this optimization effects the system's accuracy. Such a reduction in accuracy is unacceptable for many applications. As shown in Table 5, the mAP drop of this system is 3%. The system's performance is around 58, which is considered good but lower than that of the proposed system. In terms of power consumption, this system performs best due to its lower resource usage.

The authors in [13] propose hardware where all feature maps are stored in internal memory. This architecture has good DSP48 resources but uses 4073 BRAMs and 132K LUTs, and the high resource usage negatively impacts system performance. The system uses a relatively small number of DSP48s but a high number of other resources.

In [12], a 32-array of processing elements uses single instruction multiple data processing. This architecture uses a large volume of FPGA resources but achieves a low GOPS, low operation frequency, and poor system performance. A downside of this system is its high resource consumption.

In [33], a CNN accelerator is proposed that is flexible for different kernel sizes, but it uses many LUT resources and has lower GOPS and system performance. However, its power consumption and FPGA resource usage are acceptable. Despite using 16-bit precision, the system experiences a 2% mAP drop due to some optimizations in the algorithm.

The proposed system outperforms the state-of-the-art in certain parameters. As shown in Table 5, the proposed system achieves the best GOPS and accuracy while using only three DDR4 chips. This reduced number of DDR4 chips positively affects power consumption, and the proposed system delivers better GOPS per watt compared to the state-of-the-art systems.

For a comprehensive evaluation of the proposed system and architecture, practical testing was conducted. In this setup, YOLOv3-Tiny was trained on various car datasets, and the trained weights were deployed onto the FPGA board. Live video input was captured using an SDI port and a Full HD SDI camera, while the results of object detection were displayed on a monitor via an HDMI port. The test setup is shown in Fig. 7. As shown, the system's accuracy in practical tests closely matches that of the original network, with results being very close to the ground truth.

8. Conclusion

In this paper, a convolutional neural network accelerator for implementation on low-cost FPGAs has been designed in which Convolutional Processing Elements (CPEs) are utilized as the main processors for convolutions. By using these CPEs, data read and write operations in external memory chips are performed sequentially, and the number of external memory accesses is decreased. As a result, the number of external memory chips is reduced, and idle states of the Processing Elements are minimized. A new figure of merit is proposed in which all aspects of an accelerator are considered. Experimental results confirm the performance of the proposed architecture in terms of resource efficiency, network accuracy, and system speed.

In future work, we plan to compress weights and feature maps to reduce internal memory usage and decrease external memory bandwidth. Furthermore, we aim to design an architecture that can work efficiently for both single-stage and multi-stage networks.

CRediT authorship contribution statement

Mohammad Dehnavi: Supervision, Project administration, Methodology, Conceptualization. **Aran Ghasemi:** Software. **Bijan Alizadeh:** Writing – review & editing, Supervision, Formal analysis.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G.V. Hernandez, L. Krpalkova, D. Riordan, J. Walsh, Deep learning vs. Traditional computer vision, in: Advances in Computer Vision, Springer International Publishing, 2019, pp. 128–144, http://dx.doi.org/10.1007/978-3-030-17795-9_10.

[2] M. Caro, H. Tabani, J. Abella, F. Moll, E. Morancho, R. Canal, J. Altet, A. Calomarde, F.J. Cazorla, A. Rubio, P. Fontova, J. Fornt, An automotive case study on the limits of approximation for object detection, J. Syst. Archit. 138 (2023) 102872, http://dx.doi.org/10.1016/j.sysarc.2023.102872, URL https://www.sciencedirect.com/science/article/pii/S1383762123000516.

[3] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2016, pp. 779–788, http://dx.doi.org/10.1109/CVPR.2016.91.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, A.C. Berg, SSD: Single shot MultiBox detector, in: Lecture Notes in Computer Science, Springer International Publishing, 2016, pp. 21–37, http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[5] R. Girshick, J. Donahue, T. Darrell, et al., Rich feature hierarchies for accurate object detection and semantic segmentation, 580–587. URL http://www.cv-foundation.org/openaccess/content_cvpr_2014/papers/Girshick_Rich_Feature_Hierarchies_2014_CVPR_paper.pdf.

[6] R. Girshick, Fast R-CNN, in: Proceedings of the IEEE International Conference on Computer Vision, ICCV, 2015.

[7] J. Dai, Y. Li, K. He, et al., R-FCN: Object detection via region-based fully convolutional networks, Adv. Neural Inf. Process. Syst. 29, URL https://proceedings.neurips.cc/paper_files/paper/2016/file/577ef1154f3240ad5b9b413aa7346a1e-Paper.pdf.

[8] S. Ren, K. He, R. Girshick, et al., Faster R-CNN: Towards real-time object detection with region proposal networks, Adv. Neural Inf. Process. Syst. 28, URL https://proceedings.neurips.cc/paper_files/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf.

[9] K. He, G. Gkioxari, P. Dollar, et al., Mask R-CNN, 2961–2969. URL http://openaccess.thecvf.com/content_ICCV_2017/papers/He_Mask_R-CNN_ICCV_2017_paper.pdf.

[10] Y. Yan, Y. Ling, K. Huang, G. Chen, An efficient real-time accelerator for high-accuracy DNN-based optical flow estimation in FPGA, J. Syst. Archit. 136 (2023) 102818, http://dx.doi.org/10.1016/j.sysarc.2022.102818, URL https://www.sciencedirect.com/science/article/pii/S1383762122003034.

[11] J. Gutiérrez-Zaballa, K. Basterretxea, J. Echanobe, M.V. Martínez, U. Martinez-Corral, Ó. Mata-Carballeira, I. del Campo, On-chip hyperspectral image segmentation with fully convolutional networks for scene understanding in autonomous driving, J. Syst. Archit. 139 (2023) 102878, http://dx.doi.org/10.1016/j.sysarc.2023.102878, URL https://www.sciencedirect.com/science/article/pii/S1383762123000577.

[12] V. Jain, N. Jadhav, M. Verhelst, Enabling real-time object detection on low cost FPGAs, J. Real-Time Image Process. 19 (1) (2021) 217–229, http://dx.doi.org/10.1007/s11554-021-01177-w.

[13] S. Ki, J. Park, H. Kim, Dedicated FPGA implementation of the Gaussian TinyYOLOv3 accelerator, IEEE Trans. Circuits Syst. II: Express Briefs 70 (10) (2023) 3882–3886, http://dx.doi.org/10.1109/TCSII.2023.3289514.
[14] S. Li, Q. Wang, J. Jiang, W. Sheng, N. Jing, Z. Mao, An efficient CNN accelerator using inter-frame data reuse of videos on FPGAs, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 30 (11) (2022) 1587–1600, http://dx.doi.org/10.1109/TVLSI.2022.3151788.
[15] D.T. Nguyen, H. Je, T.N. Nguyen, S. Ryu, K. Lee, H.-J. Lee, ShortcutFusion: From tensorflow to FPGA-based accelerator with a reuse-aware memory allocation for shortcut data, IEEE Trans. Circuits Syst. I. Regul. Pap. 69 (6) (2022) 2477–2489, http://dx.doi.org/10.1109/TCSI.2022.3153288.
[16] S. Liang, S. Yin, L. Liu, W. Luk, S. Wei, FP-BNN: Binarized neural network on FPGA, Neurocomputing 275 (2018) 1072–1086, http://dx.doi.org/10.1016/j.neucom.2017.09.046.
[17] B. Khabbazan, M. Sabri, M. Riera, A. González, An energy-efficient near-data processing accelerator for DNNs to optimize memory accesses, J. Syst. Archit. 159 (2025) 103320, http://dx.doi.org/10.1016/j.sysarc.2024.103320, URL https://www.sciencedirect.com/science/article/pii/S1383762124002571.
[18] L. Chang, S. Zhang, H. Du, Y. Chen, S. Wang, A reconfigurable neural network processor with tile-grained multicore pipeline for object detection on FPGA, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 29 (11) (2021) 1967–1980, http://dx.doi.org/10.1109/TVLSI.2021.3109580.
[19] D.T. Nguyen, T.N. Nguyen, H. Kim, H.-J. Lee, A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 27 (8) (2019) 1861–1873, http://dx.doi.org/10.1109/TVLSI.2019.2905242.
[20] D. Zhang, A. Wang, R. Mo, D. Wang, End-to-end acceleration of the YOLO object detection framework on FPGA-only devices, Neural Comput. Appl. 36 (3) (2023) 1067–1089, http://dx.doi.org/10.1007/s00521-023-09078-8.
[21] B. Zhao, Y. Wang, H. Zhang, J. Zhang, Y. Chen, Y. Yang, 4-bit CNN quantization method with compact LUT-based multiplier implementation on FPGA, IEEE Trans. Instrum. Meas. 72 (2023) 1–10, http://dx.doi.org/10.1109/TIM.2023.3324357.
[22] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, 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 Arrays, FPGA '15, ACM, 2015, http://dx.doi.org/10.1145/2684746.2689060.
[23] A. Ahmad, M.A. Pasha, G.J. Raza, Accelerating tiny YOLOv3 using FPGA-based hardware/software co-design, in: 2020 IEEE International Symposium on Circuits and Systems, ISCAS, 2020, pp. 1–5, http://dx.doi.org/10.1109/ISCAS45731.2020.9180843.
[24] G. Dinelli, G. Meoni, E. Rapuano, T. Pacini, L. Fanucci, MEM-OPT: A scheduling and data re-use system to optimize on-chip memory usage for CNNs on-board FPGAs, IEEE J. Emerg. Sel. Top. Circuits Syst. 10 (3) (2020) 335–347, http://dx.doi.org/10.1109/JETCAS.2020.3015294.
[25] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, O. Temam, ShiDianNao: shifting vision processing closer to the sensor, in: Proceedings of the 42nd Annual International Symposium on Computer Architecture, ISCA '15, ACM, 2015, http://dx.doi.org/10.1145/2749469.2750389.
[26] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, O. Temam, DaDianNao: A machine-learning supercomputer, in: 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014, pp. 609–622, http://dx.doi.org/10.1109/MICRO.2014.58.
[27] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, L. Wang, A high performance FPGA-based accelerator for large-scale convolutional neural networks, in: 2016 26th International Conference on Field Programmable Logic and Applications, FPL, 2016, pp. 1–9, http://dx.doi.org/10.1109/FPL.2016.7577308.
[28] P.F. Felzenszwalb, R.B. Girshick, McAllester, Object detection with discriminatively trained part-based models, IEEE Trans. Pattern Anal. Mach. Intell. 32 (9) (2010) 1627–1645, http://dx.doi.org/10.1109/TPAMI.2009.167.
[29] S. Li, Y. Luo, K. Sun, N. Yadav, K.K. Choi, A novel FPGA accelerator design for real-time and ultra-low power deep convolutional neural networks compared with titan x GPU, IEEE Access 8 (2020) 105455–105471, http://dx.doi.org/10.1109/ACCESS.2020.3000009.
[30] M. Alwani, H. Chen, M. Ferdman, P. Milder, Fused-layer CNN accelerators, in: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, 2016, pp. 1–12, http://dx.doi.org/10.1109/MICRO.2016.7783725.
[31] M. Xia, Z. Huang, L. Tian, H. Wang, V. Chang, Y. Zhu, S. Feng, SparkNoC: An energy-efficiency FPGA-based accelerator using optimized lightweight CNN for edge computing, J. Syst. Archit. 115 (2021) 101991, http://dx.doi.org/10.1016/j.sysarc.2021.101991, URL https://www.sciencedirect.com/science/article/pii/S1383762121000138.
[32] Y. Wan, X. Xie, L. Yi, B. Jiang, J. Chen, Y. Jiang, Pflow: An end-to-end heterogeneous acceleration framework for CNN inference on FPGAs, J. Syst. Archit. 150 (2024) 103113, http://dx.doi.org/10.1016/j.sysarc.2024.103113, URL https://www.sciencedirect.com/science/article/pii/S138376212400050X.
[33] H. Hongmin, L. Xueming, Q. Yadong, H. Xianghong, X. Xiaoming, An efficient parallel architecture for convolutional neural networks accelerator on FPGAs, in: Proceedings of the 6th International Conference on High Performance Compilation, Computing and Communications, in: HP3C'22, ACM, 2022, http://dx.doi.org/10.1145/3546000.3546010.

Mohammad Dehnavi received the B.Sc. degree in electrical engineering from Razi University, Kermanshah, Iran, in 2009, and the M.Sc. and the Ph.D. degree in electronics from the Shahid Beheshti University, Tehran, Iran, in 2011 and 2019, respectively. He is currently an assistant professor with the Department of Electrical Engineering, Kermanshah University of Technology, Iran. His research interests include video processing accelerators, embedded systems, hardware implementation of image and signal processing systems, and convolutional neural networks.

Aran Ghasemi obtained his B.Sc. degree in Electronics Engineering from Kermanshah University of Technology, Iran, in 2019. He is presently involved in Digital Electronic Systems research at the Electrical and Computer Engineering laboratory, Kermanshah University of Technology, Iran. His primary research interests include video processing accelerators, hardware implementation of image and signal processing systems, and convolutional neural networks.

Bijan Alizadeh received his Ph.D. degree in electrical and computer engineering from the University of Tehran, Iran, in 2004. Prior to joining the University of Tehran in 2011, He was with the School of Electrical Engineering, Sharif University of Technology, Iran, as an Assistant Professor from 2005 to 2007 and VLSI Design and Education Center (VDEC), The University of Tokyo, Japan, as a Research Associate from 2007 to 2010. In 2011, he joined the School of Electrical and Computer Engineering at the University of Tehran as an assistant professor, where he has been an associate professor since August 2017. He has authored or co-authored over 140 publications in international scientific journals and conferences. He has been engaged in the research and development of VLSI systems, FPGA-based reconfigurable computing, formal verification and debug, hardware security, and high-level synthesis.