Journal of Real-Time Image Processing (2024) 21:64
https://doi.org/10.1007/s11554-024-01442-8

SURVEY

Survey of convolutional neural network accelerators on field-programmable gate array platforms: architectures and optimization techniques

Hyeonseok Hong[1] · Dahun Choi[1] · Namjoon Kim[1] · Haein Lee[1] · Beomjin Kang[1] · Huibeom Kang[1] · Hyun Kim[1]

Received: 23 November 2023 / Accepted: 20 February 2024
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2024

Abstract
With the recent advancements in high-performance computing, convolutional neural networks (CNNs) have achieved remarkable success in various vision tasks. However, along with improvements in model accuracy, the size and computational complexity of the models have significantly increased with the increasing number of parameters. Although graphics processing unit (GPU) platforms equipped with high-performance memory and specialized in parallel processing are commonly used for CNN processing, the significant power consumption presents challenges in their utilization on edge devices. To address these issues, research is underway to design CNN models using field-programmable gate arrays (FPGAs) as accelerators. FPGAs provide a high level of flexibility, allowing efficient optimization of convolution operations, which account for a significant portion of the CNN computations. Additionally, FPGAs are known for their low power consumption compared to GPUs, making them a promising energy-efficient platform. In this paper, we review and summarize various approaches and techniques related to the design of FPGA-based CNN accelerators. Specifically, to comprehensively study CNN accelerators, we investigate the advantages and disadvantages of various methods for optimizing CNN accelerators and previously designed efficient accelerator architectures. We expect this paper to serve as an important guideline for future hardware research in artificial intelligence.

Keywords Convolutional neural network · Accelerator · Field-programmable gate array (FPGA) · Design optimization · Data flow

1 Introduction

In recent years, the use of convolutional neural networks (CNNs) has significantly increased in various image processing fields, including classification [1, 2], object detection [3, 4], and segmentation [5, 6]. However, high-performance deep CNNs require substantial computational power, with operations such as matrix multiplication (MM) demanding on the order of 10–1000 giga-floating-point operations per second (GFLOPS) [1, 2, 7, 8]. Therefore, achieving high throughput and power efficiency in computing systems is crucial for the practical utilization of CNNs [9]. Traditionally, central processing

Hyeonseok Hong and Dahun Choi have contributed equally to this work.

✉ Hyun Kim
hyunkim@seoultech.ac.kr

Hyeonseok Hong
hs_hong@seoultech.ac.kr

Dahun Choi
dahun926@seoultech.ac.kr

Namjoon Kim
rlarla2626@seoultech.ac.kr

Haein Lee
haeinlee@seoultech.ac.kr

Beomjin Kang
beomjin@seoultech.ac.kr

Huibeom Kang
huibeom_k@seoultech.ac.kr

[1] Department of Electrical and Information Engineering, Research Center for Electrical and Information Technology, Seoul National University of Science and Technology, 232 Gongneung-ro, Nowon-gu, Seoul 01811, Korea

units (CPUs) have been limited by their computational throughput, which ranges from 10 to 100 GFLOPS, hampering their ability to completely support real-time CNN processing [10]. By contrast, GPUs, such as NVIDIA A100, exhibit a peak performance of 19.5 tera-FLOPS, enabling real-time operation of CNNs, thereby being extensively used in a wide range of applications [11]. However, GPUs face challenges such as significant power consumption (often exceeding 400 W), large and heavy form factors, and expensive prices. Consequently, they are primarily used in server environments and are impractical for deployment on edge devices [12]. Thus, there is a growing need for new hardware platforms to enable the practical deployment of CNN applications across diverse devices.

To address this issue, a CNN accelerator system utilizing field-programmable gate arrays (FPGAs) has been proposed, offering significant computing acceleration while maintaining high energy efficiency [13–19]. Unlike traditional instruction-based processors, such as CPUs and GPUs, FPGAs leverage register-transfer level (RTL) designs to enable flexible and reconfigurable designs. By implementing convolution (CONV) and MM operations, which constitute a significant portion of CNN operations, at the gate level and optimizing the computational flow for different models, FPGA systems with limited resources can be used to design various CNN models with low latency and high power efficiency. However, even when implementing the same CNN model, FPGA-based CNN accelerators exhibit varying hardware utilization and throughput depending on the FPGA platform and design methodology. Therefore, it is important to achieve optimal performance (i.e., throughput and power efficiency) within the constraints of limited resources, such as block RAM (BRAM), digital signal processing (DSP) blocks, lookup table (LUT), and Flip-Flop (FF). Therefore, various computation and memory optimization techniques and efficient architecture designs that consider FPGA resources and model structures are essential [20–22]. This paper summarizes the different architectures and optimization techniques used for designing FPGA-based CNN accelerators. In the Sect. 2, we provide a background on CNNs and FPGAs. In the Sect. 3, we provide a brief overview of the techniques required for CNN accelerator design. In the Sect. 4, we examine the optimization methods for maximizing hardware utilization and parallel computing. In the Sect. 5, we explore the architectures of various FPGA-based CNN accelerators. In the Sect. 6, different CNN models implemented on various FPGA chips are compared. Finally, in the Sect. 7, we provide a summary of this paper and discuss the prospects for CNN accelerators.

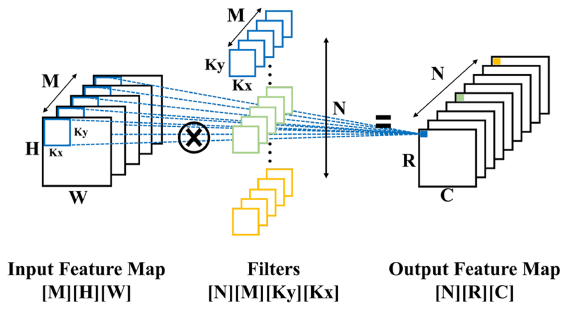
2 Preliminary

2.1 Convolution neural network

A CNN is a deep-learning model that is primarily used in image processing and pattern recognition [1, 2, 23–25]. It comprises multiple layers for feature extraction (e.g., CONV and fully connected (FC) layer) and nonlinear functions (e.g., activation and pooling). During the CONV operation, multiplication operations are performed as the filter slides across the input feature map with a specific stride, summing the products to generate the output feature map. This process allows the extraction of local information from the image. Iterative stacking of these layers allows extraction of low-level features, such as corners and edges, in the early layers, progressing toward more abstracted high-level features in the deeper layers. A CNN applies an activation function to each feature and reduces the resolution of the feature map through pooling. This process introduces nonlinearity into the CNN and facilitates efficient feature extraction from the CONV layers. The extracted features are subsequently passed through the FC layer, which is structured around interconnected input and output nodes that enable the calculation of the final output for classification and prediction. As a result, CNNs demonstrate superior accuracy in image pattern recognition compared to conventional machine learning techniques. Consequently, the CNN has been widely applied across various domains, such as classification [1, 2], object detection [3, 4], segmentation [5, 6], and super-resolution [26].

The CONV operation is crucial in CNNs because it generates an output feature map by performing multiply–accumulate (MAC) operations on an input feature map and weighted filter. Figure 1a illustrates a conceptual diagram of the CONV operation. Here, K_x and K_y represent the size of the filter, with M and N denoting the number of channels in the input and output feature maps, respectively; W, H, C, and R correspond to the width and height of the input and output feature maps, respectively. A filter of size $M * K_y * K_x$ is overlapped with the $M * H * W$ input feature map to obtain single-pixel output features through element-wise multiplication and addition. The filter moves based on stride size while performing the operation $C * R$ times and repeating this for N filters, thereby obtaining the final output feature map of dimension $N * C * R$. The CONV operation performed in one layer can be expressed in pseudocode, as shown in Fig. 1b, and the number of operations (i.e., $\mathrm{Oper}_{\mathrm{CONV}}$) is calculated, as follows:

$$\mathrm{Oper}_{\mathrm{CONV}} = K_x * K_y * M * C * R * N. \tag{1}$$



(a)

```
s = stride

for n in range(N): }  ——  Loop 4 :  Output Feature Map Channel
  for r in range(R):  }
    for c in range(C): }  ——  Loop 3 : Output Feature Map Resolution
      for m in range(M): }  ——  Loop 2 : Input Feature Map Channel
        for i in range(Ky): }
          for j in range(Kx) }  ——  Loop 1 : Kernel Size
            Output_Feature_Map[n][r][c] +=
            Input_Feature_Map[m][r * s + i][c * s + j] *
            Filter[n][m][i][j]
```

(b)

Fig. 1 Illustration of **a** convolution operation and parameter shape and **b** pseudo code

2.1.1 Convolution operation

2.1.2 Conventional general matrix multiply architecture

CONV operations, comprising six loops, may experience performance degradation due to branch instruction when executed in conventional computing environments such as CPUs and GPUs. To address this challenge, the general matrix multiplication (GEMM) method has been extensively adopted, whereby the 3D input feature map is elegantly reshaped into an $(N * K * K) \times (R * C)$ matrix, and the 4D filter is reshaped into an $M \times (N * K * K)$ matrix, subsequently implementing the CONV operation through MM. This methodology facilitates the high-weight reuse pattern in CONV operations, leading to a significant throughput enhancement. Moreover, it is sufficiently versatile to be applied to a wide range of model configurations, allowing the handling of highly complex tasks. Nevertheless, GEMM operations have their limitations. These include latency issues during the reshaping process, considerable memory overhead arising from substantial matrix sizes, and increased power consumption [27]. Consequently, there is a pressing need for novel computing platforms and strategic optimization formulations for CONV operations, particularly to augment the efficiency of CNN inference.

2.2 Field-programmable gate array

FPGAs are reconfigurable hardware devices suitable for digital circuit implementation. FPGAs employ programmable logic (PL) based on hardware description languages for circuit design and can be categorized into two types. The first type, peripheral component interconnect express (PCIe)-based FPGAs, are used in large-scale data centers without a processing system (PS) and excel in accelerating computation tasks with massive data processing requirements [28]. Notable examples are Xilinx Alveo [29] and Intel PAC chip [30]. By contrast, SoC FPGAs, which have an integrated PS, allow designs within the same development environment. Owing to their reasonable pricing, low power consumption, and compact size, they are extensively used in mobile and edge devices for image processing, signal processing, and deep-learning applications [31]. This paper focuses on the CNN accelerator implemented using SoC FPGAs. In SoC FPGAs, the PL enables users to achieve their desired hardware design by implementing anything from simple logic to complex function modules at the gate level. Meanwhile, PS handles complex operations using processing cores, facilitates data communication between off-chip memory and PL through memory interfaces, and enables interconnection with peripherals (i.e., sensors and displays), allowing the construction of complex system-on-chip circuits [21]. The design of CNN accelerators on FPGAs offers several advantages over other hardware devices (e.g., CPU and GPU) in terms of design flexibility, power efficiency, and latency [32]. First, the flexible and configurable nature of FPGAs enables the optimized design of various CNN architectures and layer operators. This advantage allows flexible adaptation to new models or algorithm changes. Second, FPGAs achieve high power efficiency by performing only the optimal calculations required for CNNs. This is particularly beneficial in power-constrained environments, such as mobile devices and edge computing [12]. Third, FPGAs offer very low latency by enabling the direct connection of peripherals to logic through parallel processing and I/O interfaces. This advantage is particularly relevant in fields where low latency is essential, such as autonomous vehicles. FPGAs consist of resources such as PS, arithmetic units (LUTs, FFs, and DSP blocks), and on- and off-chip memory (dynamic random-access memory). This section elucidates the functions of each of these resources in accelerator design. Table 1 lists the specifications of FPGA chips commonly employed in CNN accelerator research. Notably,

Table 1 Hardware resources in various FPGAs

FPGA Chip	Processor core	DSPs	LUTs	FFs	On-chip Memory (Mb)
Intel Stratix V 5SGXA7	Nios® II processor	768	469,440	938,880	50.0
Intel Stratix 10 GX 10 M	Quard-core ARM Cortex-A53 MPCore with CoreSight	3456	693,2160	13,864,320	259.0
Intel Arria 10 GX 1150	Dual-core ARM Cortex-A9 MPCore with CoreSight	1518	854,400	1,708,800	54.3
Xilinx Zynq-7000 XC7Z020	Dual-core ARM Cortex-A9 MPCore with CoreSight	220	53,200	106,400	4.9
Xilinx Zynq-7000 XC7Z045	Dual-core ARM Cortex-A9 MPCore with CoreSight	900	218,600	437,200	19.2
Xilinx Zynq Ultrascale + XCZU2EG	Quard-core ARM Cortex-A53 MPCore with CoreSight	240	47,232	94,464	5.3
Xilinx Zynq Ultrascale + XCZU7EV	Quard-core ARM Cortex-A53 MPCore with CoreSight	1728	230,400	460,800	BRAM 11.0, URAM 27.0
Xilinx Zynq Ultrascale + XCZU9EG	Quard-core ARM Cortex-A53 MPCore with CoreSight	2520	274,080	548,160	32.1
Xilinx Virtex-7 XC7VX485T	MicroBlaze^{TM} processor	2800	303,600	607,200	37.0
Xilinx Virtex-7 XC7VX690T	MicroBlaze^{TM} processor	3600	433200	866400	53.0
Xilinx Virtex Ultrascale + XCVU9P	Quard-core ARM Cortex-A53 MPCore with CoreSight	6840	1,182,240	2,364,480	BRAM 75.9 URAM 270.0

FPGA boards used in previous papers for which experimental results have been presented, were included in this paper.

2.2.1 Processing system

The processor core embedded in the PS of FPGAs operates separately from the PL. Considering the high performance and cost of the processor core, it is used to handle the complex operations in CNN implementations that are challenging or resource-intensive to implement in PL, such as softmax and non-maximum suppression. Such data are communicated between the off-chip memory and PL via direct memory access (DMA).

2.2.2 Arithmetic units

Arithmetic units within FPGAs are hardware blocks for performing various arithmetic operations. FPGAs include large-scale arithmetic blocks with low complexity. Typically, arithmetic operations can be performed using LUTs, which provide output values corresponding to the input values and FFs, which store data temporarily. FPGAs contain a significant number of LUTs and FFs, allowing the implementation of complex logic operations through their interconnection. This enables the parallel processing of multiple operations, thereby increasing hardware utilization. DSP slices are specialized hardware blocks within an FPGA that enable the fast execution of signal processing algorithms at high speeds. The DSP slices contain components such as multipliers, accumulators, and registers. These elements accelerate the MAC calculations, enabling high-performance computations. Owing to their high computational throughput and superior operational efficiency, DSP slices are primarily used to implement processing elements (PEs) for the most critical CONV operations in CNN accelerators [33].

2.2.3 On-chip memory

On-chip memory in FPGAs, known as BRAM, stores feature maps, filter weights, and other data, enabling efficient data flow design by synchronously delivering data to the PE. High-end FPGAs feature an on-chip memory called ultra RAM (URAM). With a capacity of 288 Kb per slice, URAM offers nine times the storage capacity of standard BRAM, which holds 32 Kb per slice. This makes URAM particularly suitable for data-intensive applications. Although on-chip memory provides fast access, it has a significant size limitation, as shown in Table 1. Therefore, in CNN accelerators, there is a limitation that all parameters (e.g., filters, and feature maps) cannot be stored using on-chip memory, and most designs eventually require off-chip memory access.

2.2.4 Off-chip memory

Off-chip memory is typically implemented using dynamic random-access memory (DRAM), which offers a relatively large capacity and is peripherally connected to the FPGA chip. In general, off-chip memory is used to store all the necessary filters for the CNN as well as the output feature maps and input images resulting from the CONV operations within the PL. However, communication between the PL and off-chip memory is typically limited to the PS and DMA, resulting in relatively high latency and power consumption. Therefore, most CNN accelerator designs aim to minimize off-chip memory access [13, 34, 35].

3 Brief overview of CNN accelerator design

This paper primarily focuses on FPGA accelerator design, examining two key aspects for effective implementation (see Fig. 2). The first is the optimization technique. At the hardware level, ongoing research aims to maximize the utilization of parallel processing techniques, such as unrolling and batching, to alleviate computational bottlenecks and improve performance. Additionally, studies have sought to minimize DRAM access costs through double buffering, tiling, distributed BRAM, and memory hierarchy utilization. Compression methods, including quantization, pruning, and winograd, have also been explored to efficiently utilize limited resources by reducing MAC operation costs. The effective integration of these techniques is crucial for developing FPGA-based CNN accelerators that reduce computational complexity and minimize memory access for enhanced performance. The second key factor is the hardware architectural design, which is critical in designing an efficient implementation of CNN characteristics. The performance of an accelerator can be improved based on the design of the processing unit (PU). Research on fusing operations between adjacent layers aims to increase operational efficiency and reduce memory access costs. Additionally, various architectures can be designed, such as systolic arrays (SAs), leveraging a pipelined approach with multiple PE and multi-PU to prevent the idle state of PEs. Finally, given the operational characteristics of CNNs, notably, a technology that divides and processes calculations between the CPU and FPGA, known as CPU-FPGA collaborative computing, higher performance can be achieved compared to a single computing platform.

4 Optimization techniques of CNN accelerators for FPGA implementation

To implement CNN models efficiently on FPGA systems with limited memory capacity and available resources, employing various optimization techniques is crucial. These techniques generally include computational optimization [15, 16, 36–38], memory optimization [34,

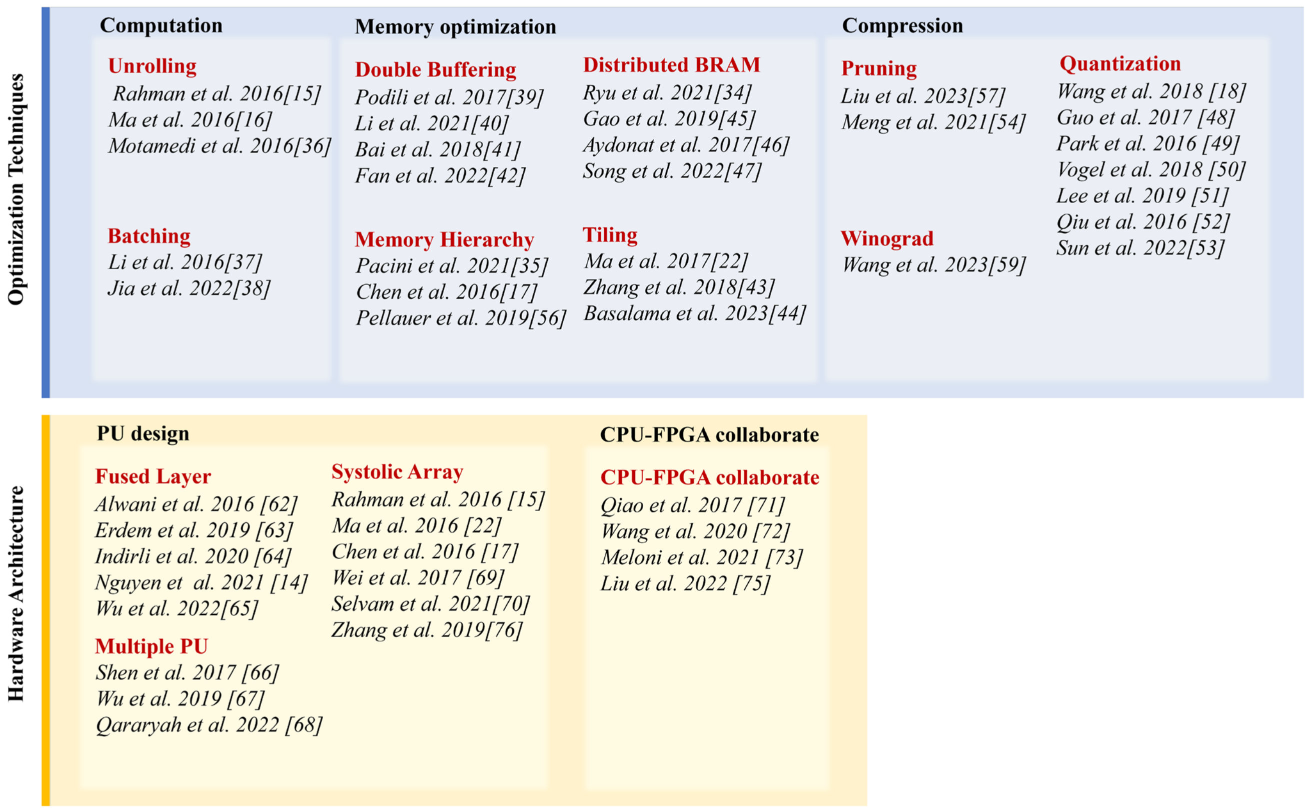


Fig. 2 Classification of various approaches for CNN accelerator design

39–48], and compression [18, 49–55]. Integrating these techniques effectively is essential in developing FPGA-based CNN accelerators, as they contribute to optimizing its performance by reducing computational complexity and minimizing memory accesses.

4.1 Computational optimization

Representative computational optimization techniques that improve the performance and maximize the efficiency of CNN accelerators on FPGAs include unrolling to improve parallel processing and tiling to reduce memory access costs. Repetitive calculations can be executed at once through unrolling, and overall system performance can be improved by minimizing memory access through tiling.

4.1.1 Unrolling

Unrolling is a technique used to reduce the number of iterations in a loop. It is an optimization technique aimed at improving performance by minimizing the overhead caused by the large number of iterations involved in the CONV operation. In FPGAs, it is possible to reduce computation costs more effectively by utilizing the parallelized hardware structure to perform multiple computations simultaneously. As depicted in Fig. 3, the iteration of the CONV layer consists of four levels: kernel loop (P_x, P_y), input channel loop (P_m), intra-channel loop (P_w, P_h), and output channel loop (P_n). By applying the unrolling technique to the kernel loop, multiple kernel elements can be processed simultaneously, and by applying the unrolling technique to the input channel loop, CONV operations can be performed at multiple feature map locations simultaneously. Moreover, unrolling the intra-channel loop enables concurrent operations on multiple input channels, while unrolling the output channel loop allows simultaneous operations on multiple output channels. Unrolling these loops at different levels in CONV operations effectively reduces overhead. For instance, unrolling the intra-channel loop minimizes memory access by leveraging weight reuse and facilitates parallel processing of multiple multiplication operations. Rahman et al. [15] proposed a new architecture called the input-recycling CONV array of neurons, which optimizes memory and computational resources. However, this architecture suffered from high computational complexity as a disadvantage. To address this issue, the internal CONV operation was optimized by unrolling the intra and output channels, thereby reducing the complexity of the architecture. Ma et al. [16] designed a CONV and pooling layer to parallelly output adjacent feature maps, which led to an issue of overlapping data and computations in the normalization module. The output

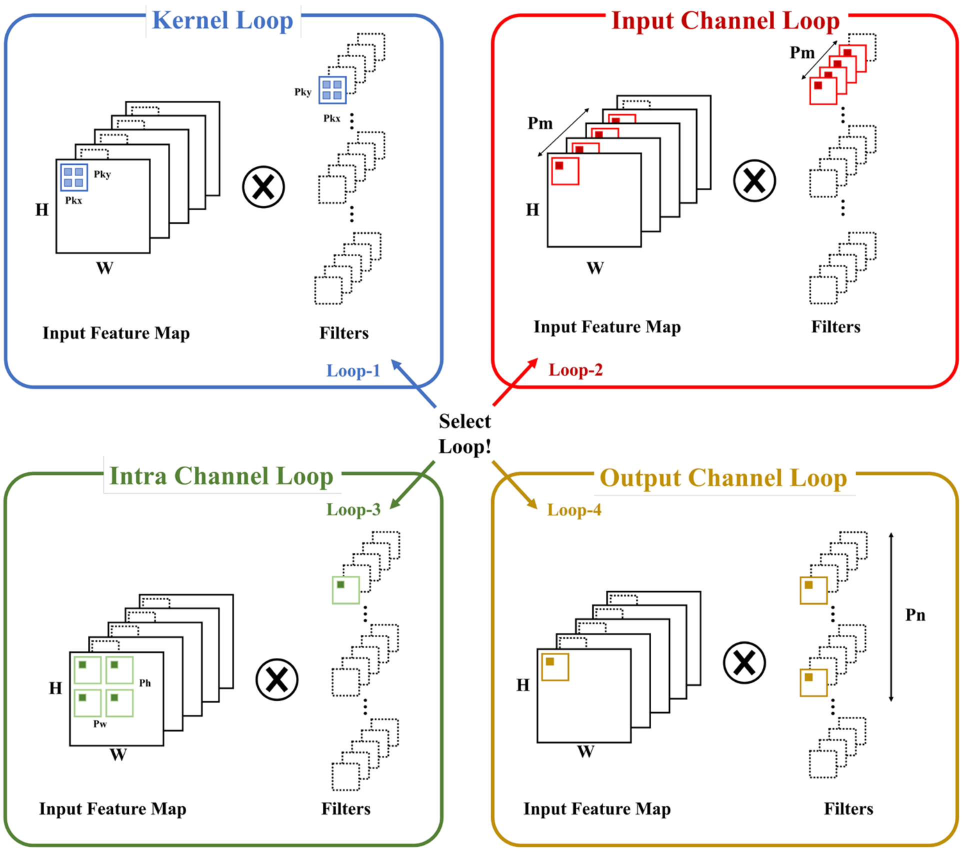


Fig. 3 Four levels of convolution loop unrolling from [43]

channel loop was unrolled to avoid data overlap. In addition, the design incorporated the reuse of intermediate pixels to save memory. Motamedi et al. [36] proposed the parallel convolution engine (PCE) that leverages parallelism by unrolling the kernel, input, and output loops. They achieved approximately a 1.9% speed improvement by unrolling the kernel loop and combining multiplication and addition units to exploit parallelism on the same FPGA device. It is, therefore, crucial to choose the appropriate unrolling strategy for each loop level, considering the specific requirements and constraints of the hardware.

4.1.2 Batching

Batching, which processes multiple input images simultaneously to enhance the throughput, can be applied to operations that compose CNNs, such as CONV and FC layers. In Fig. 4, we can see the application of batching during FC operations. Using the same weight addressing for multiple input data operations, it maximizes weight data reuse, converting vector multiplication into MM, achieving N times the throughput. Li et al. [37] *implemented batching to diminish the weight load required for FC operations thereby reducing the necessity for off-chip memory access. Their approach effectively maximizes the reuse of FC weights, allowing for computations on multiple input data with minimal DRAM access. Furthermore, the study introduced a novel technique to counterbalance the increased output buffer consumption caused by larger batch sizes in successive FC layer operations, by shifting the weight window addressing pattern from a vertical (Fig.* 4b) *to a horizontal orientation (Fig.* 4c), *thus requiring less space for temporary results. Remarkably, this technique achieved a performance efficiency of 391 frames per second (FPS) on Xilinx XC7VX690T, demonstrating its significant impact on computational efficiency.* Jia et al. [38] proposed an accelerator with three levels (i.e., core, graph, and batch) of scalability for various CNN model operations. They processed multiple input images synchronously in the operation unit graph. Leveraging the characteristic that weights are shared for each input, they boosted throughput using just one weight buffer and proposed a scalable design capable of 1–8 batch operations. As a result, through the application of various parallelization techniques, the authors achieved 8× throughput. In conclusion, although batching can significantly increase throughput through parallel operations, the resource consumption such as PE or on-chip memory storage required to process the increased operations due to batch size must be considered.

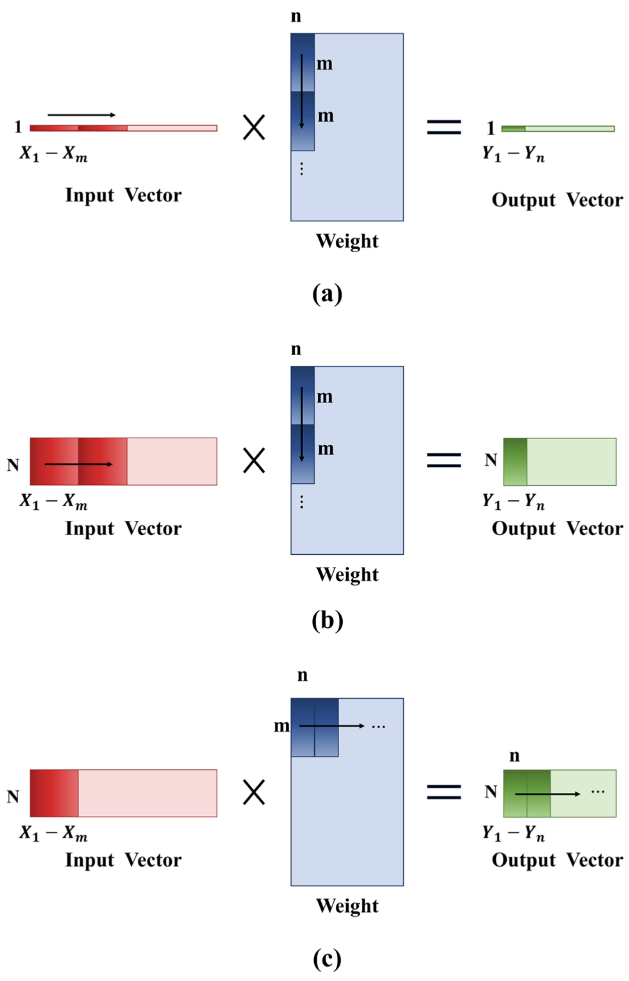


Fig. 4 Illustration of FC operation from [37]. **a** FC operation single image processing, **b** batching FC vertical weight addressing, **c** horizontal weight addressing

4.2 Memory optimization

In addition to Subsection *Computational Optimization*, harmonious design between memory, which stores operands (e.g., CONV filter, FC weight, and feature map), and PE is necessary to maximize operational efficiency and implement low-power systems. As introduced in Subsection *Field-programmable gate array*, the memory used in FPGA systems can be largely divided into on-chip memory (e.g., SRAM, BRAM, cache, and buffer) and off-chip memory (e.g., DRAM). Despite the advantages of BRAM fast access time and high scalability, it is impossible to store all massive operands, such as those of ResNet-101 [1] (i.e., 44 M weight parameters, 170 MB) and VGG-16 [7] (138 M weight parameters, 527 MB), due to the limited capacity of BRAM. Therefore, the parameters of CNN models must be stored in DRAM, which provides a larger capacity, and data must be loaded to the PL via an interface (i.e., AXI) and DMA.

However, the energy consumed in DRAM access (i.e., 640 pJ) is approximately 130× greater than that in SRAM access (i.e., 5 pJ) [56], and the throughput of accelerators is negatively impacted due to bandwidth limitations and long latency. Therefore, this subsection explains memory optimization methods that reduce off-chip memory access and maximize the utilization of capacity-limited on-chip memory to provide an efficient data flow of operations for the accelerator.

4.2.1 Double buffering

In CNN accelerators, the on-chip memory not only exchanges data with the off-chip memory but also delivers operands connected to the PE and stores the results of MAC operations. However, a single buffer cannot perform both functions at once, and during communication with the off-chip memory, the PE becomes idle. This significantly lowers PE utilization and increases latency. Double buffering is a method used to address this issue, using two buffers to simultaneously process data delivery to the PE and data loading from the off-chip memory. When double buffering is applied, one buffer communicates with the PE and is used for the current layer operation, while the other buffer loads the data necessary for the next layer operation from the off-chip memory. The roles of the two buffers switch every time a layer is completed, which is referred to as "ping-pong". The advantage of double buffering is that it can hide latency caused by data transfer, as it can load the necessary data during the layer operation time. Podili et al. [39] applied the double buffering technique to the kernel buffer by connecting one input buffer and several kernel buffers to the PE. When implementing the VGG-16, they reduced latency by hiding the data refill time through at least 196 data reuses. Li et al. [40] proposed a block CONV to minimize off-chip memory access, allowing multiple tiles to be loaded iteratively and written to the main memory by partitioning the feature map into 27×48 small tiles and then applying double buffering. Bai et al. [41] applied double buffering to the weight buffer to reduce the latency of filter loading, which varies for the three types of CONVs (i.e., standard, depth-wise, point-wise) that compose MobileNetV2. They were able to implement an accelerator with a weight buffer size of only 36Kb, with 3.7% usage of Intel Arria 10 GX 1150. Fan et al. [42] not only used the traditional classification loss of cross entropy but also introduced latency and energy as losses in their network architecture search approach. This allowed them to find the optimal compressed network within limited resources. By applying a ping-pong mechanism to both the input feature map and weight buffer, they managed to hide latency due to off-chip memory access, achieving a performance of 319 FPS on Intel Arria 10 GX 1150. However, compared to a single buffer, double buffering has the drawback of resulting in structural changes that complicate the memory controller and generally increase on-chip memory usage.

4.2.2 Tiling

In the FPGA implementation of CNN accelerators, the CONV operation, which accounts for the majority of computations, is performed through MAC operations within the PE. Typically, the PE receives the feature input and weights as operands from on-chip memory (i.e., BRAM). However, there are capacity limitations regarding holding feature maps and weights in BRAM, especially as their sizes increase proportionally with the complexity of the model. Consequently, the implementation of an accelerator for large CNN models relies on loading feature maps and weights from off-chip memory (i.e., DRAM). However, fetching large amounts of data from DRAM can have a significant negative impact on layer and network latency [13]. To address this issue, tiling is employed to fetch the required feature data from DRAM in block-sized units, known as tiles. These tiles are then stored in BRAM, allowing for maximum reuse of fetched data and enabling the computation of CONV operations in complex models with limited on-chip memory resources. Tiling divides the input feature map into T tiles, reducing the on-chip memory requirement to $1/T$ of the original size. This enables efficient BRAM utilization and facilitates the implementation of complex CNN models on FPGA platforms with limited resources. Ma et al. [43] defined the latency associated with tiling, including the intra-tiling loop (on-chip memory access) and inter-tiling loop (off-chip memory access). They analyzed each tiling technique to explore the trade-off between on-chip memory size and external memory access. Then, they proposed a design methodology to identify the optimal on-chip memory size and latency. As a result, they implemented VGG on Intel Arria 10 GX 1150 achieving a throughput of 645 giga operations per second (GOPS). Zhang et al. [44] addressed the issue of frequent DRAM access caused by the row-major representation of feature maps in tiling-applied RTL designs using high-level synthesis. They proposed cube index transformation and DRAM layout techniques to maximize the utilization of DRAM memory burst length and bit width. Basalama et al. [45] proposed a technique called "dynamic tiling", in which a different tiling factor is assigned to each layer of the network. This approach enables data feeding between line buffers and results in a 1.7× performance improvement in terms of latency compared to traditional uniform tiling methods. Indeed, tiling has some limitations that need to be considered. One limitation is the increase in design complexity due to changing access patterns in on-chip memory. This can introduce additional challenges in the design process. Additionally, the frequent off-chip memory communication in tiling-based designs can result in power inefficiency, which can be a significant drawback from a power-efficiency perspective. Therefore, it is crucial to carefully analyze and address these challenges when employing tiling techniques so that the overall design trade-offs can be optimized and the desired performance and efficiency goals can be achieved.

4.2.3 Distributed BRAM

Distributed BRAM is a method of maximizing PE utilization by placing dedicated BRAM at the PE. Figure 5a, b show an overview of centralized BRAM and distributed BRAM. In centralized BRAM, because all PEs share a global buffer and data path, a data bottleneck occurs when the data demand of the PE increases. This results in a significant decrease in PE utilization due to the increased number of unused PEs. In contrast, distributed BRAM maintains high PE utilization and increases processing speed through operation parallelization by placing a dedicated buffer for each PE, allowing all PEs to be used simultaneously for operations. Ryu et al. [34] placed a PE and SRAM buffer for each channel and performed separable channel-wise CONV operations [23], allowing for various MobileNet [23, 24] designs through channel stationary techniques. Moreover, by storing all weights and feature maps used in each channel in distributed SRAM, they implemented depth-wise separable CONV operations within MobileNet without accessing off-chip memory. Gao et al. [46] proposed a tile architecture, which combines multiple small PE arrays, and implemented coarse-grained parallelism through data sharing by placing a buffer for each tile. Aydonat et al. [47] proposed stream buffer arrays, which supply the input feature map needed for CONV layer execution to PEs and store the CONV output in the buffer. By storing filters in each PE cache, they minimized idle computations in the PE. Song et al. [48] proposed a kernel pruning method called hardware-oriented regular pruning, utilizing the finite impulse response (FIR) filter, and implemented a double CONV PE module. By allowing each PE to process two 1D kernel weights and an input feature map, they designed an efficient pruning-aware accelerator. As a result, they achieved a significant 5.83× weight compression at VGG and reduced the MAC usage by 1.46×, while still attaining a high throughput of 110 FPS on Xilinx XCVU9P. However, when using distributed BRAM, each PE can only use a small buffer size due to the limited on-chip memory resources within the FPGA. Moreover, data sharing between PEs can lead to BRAM access conflicts.

4.2.4 Memory hierarchy Utilization

Memory hierarchy aims to overcome the limited capacity of on-chip memory and resolve the high latency and power consumption issues of off-chip memory by operating a combination of various levels of memory (i.e., DRAM, cache, buffer, etc.) (see Fig. 5c). Pacini et al. [35] proposed a method to increase power efficiency by minimizing off-chip access and drastically reducing on-chip memory usage. Instead of connecting the on-chip buffer used in the PE and the off-chip memory, they implemented L1 and L2 cache to store the feature map repeatedly used in operations in the L1 cache, thereby maximizing data reuse. Chen et al. [17] established a four-level storage hierarchy (i.e., off-chip DRAM, global buffer, FIFO array, and register file) and analyzed the energy cost arising from data movement in each storage. Pellauer et al. [57] proposed a "buffet" memory, which borrows the explicit decoupled data orchestration taxonomy to solve the inefficiency of

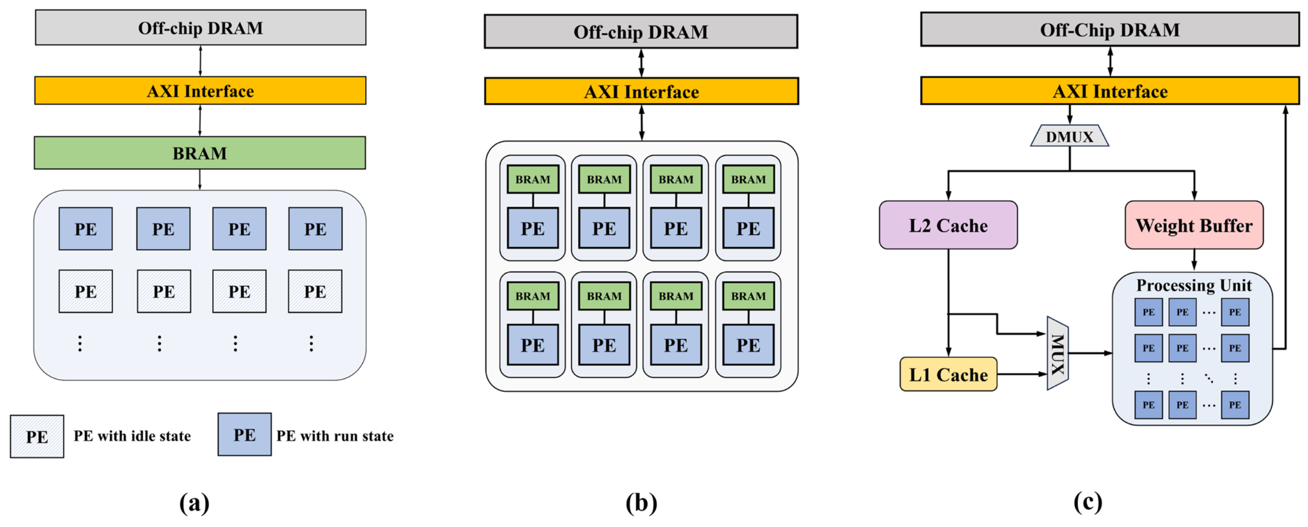


Fig. 5 Various CNN accelerator memory system from [37]. **a** Centralized BRAM, **b** distributed BRAM, **c** hierarchical cache

implicit data orchestration inherent in traditional caches and the inflexibility in data reuse found in FIFO. The authors established a memory hierarchy by dividing the buffet into three levels and applying it to the tiled-GEMM operation accelerator, demonstrating energy efficiency that is 1.53× and 5.39× greater compared to traditional double-buffered scratchpads and cache, respectively. However, implementing a memory hierarchy system in FPGA-based CNN accelerators presents challenges compared to designs using simple on-chip memory.

4.3 Compression

The compression technique is employed in CNN accelerators to decrease memory bandwidth and computational demands. Notably, traditional models such as ResNet-18 and ResNet-50 [1] possess substantial model sizes (i.e., ResNet-18: 46.8 MB, ResNet-50: 97.5 MB). Developing CNN accelerators for these models requires significant computational resources and frequent DRAM access, resulting in heightened latency and increased power consumption. CNN compression techniques are essential for addressing these challenges. In this section, we explain quantization, which is one of the most widely utilized compression methods in CNN accelerator design and provide a brief overview of other compression techniques, such as pruning [58, 59] and Winograd [60].

4.3.1 Quantization

Quantization is a compression technique that converts 32-bit floating-point data (FP32) into low-precision integer or fixed-point formats [61]. Some information is inevitably lost when converting 32-bit data into low-precision format. However, accuracy can be recovered through quantization-aware training, achieving even higher accuracy than that of the baseline through recent quantization research [54]. By performing INT8 operations instead of the traditional FP32 operations during CONV in the MAC unit of CNN accelerators, an approximately 4× faster inference speed can be achieved while reducing energy consumption by approximately 18.5× [62]. In addition, this technique effectively reduces the size and memory footprint of the model, making it highly efficient in resource-constrained environments [53].

Quantization can be classified based on the data format used in MAC operations. Guo et al. [49] presented a linear quantization technique in the most widely used INT format. By converting the weight and activation values of each layer to integers, this technique offers advantages such as reduced computational complexity compared to floating-point formats, improved computational speed, and simplified hardware implementation. Park et al. [50] quantized the data format to fixed-point numbers, thereby enabling hardware-friendly operations through a combination of integers and shift operations (fraction bits), while also providing an advantage in terms of accuracy by expressing a wider range than the integer format. Vogel et al. [51] performed the CONV operation in logarithmic form by taking logarithms of weights and activations. Although this method involves nonuniform quantization intervals, it can significantly reduce hardware resource usage and power consumption compared to other data formats by allowing operations in addition to multiplication. However, quantization using this method requires a significant number of LUTs, and the quantizer has higher latency than other techniques. Wang et al. [18] significantly reduced memory consumption and computation on Zynq-7000 XC7Z045 through a low-precision CNN model. However, to ensure high network accuracy, the first and last layers had high precision, and binary and ternary quantization were applied to the middle layers. The experimental results showed a decrease of 2.6% in accuracy compared to the baseline when all middle layers were quantized as ternary in AlexNet, and a decrease of 0.7% was obtained compared to the baseline when all layers were quantized as 8-bit. Lee et al. [52] used the stochastic computing (SC) technique to effectively quantify CNN because the quantization performance can vary greatly depending on the dataset, training method, and network. However, SC has the problem of high latency in CNN accelerator design and poor high-precision quantization. To solve the high latency caused by SC, this research applied logarithmic quantization to reduce resource and computational costs. Compared with linear quantization, the latency of the operation was reduced through lower-precision quantization, and the computational cost was effectively reduced by addition instead of multiplication MAC operations. Qiu et al. [53] observed variations in the quantization range of weights and activations across different layers. To address this issue, they analyzed the distribution of each layer and determined the configurations that minimized quantization errors. Specifically, they allocated eight or four bits to different layers. Experimental results on VGG16-SVD demonstrated that the proposed approach achieved an accuracy similar to that of the baseline, while providing an approximately 1.4× faster processing speed than the CPU. Because each data format has different advantages and disadvantages, a quantization strategy should be selected considering the data format that is most suitable for the specific environment. Sun et al. [54] presented an efficient FPGA acceleration method for CNNs with intra-layer and mixed-precision quantization. They addressed the issue of irregular distribution of 8-bit filters throughout the entire layer on FPGA due to mixed-precision quantization, where bit operations are processed separately. This technique divides the filters in each layer into several weighted tiles, each containing a certain number of filters. For each weight tile, the filter ordering is reorganized such that the first tiled filters are preserved as 8-bit quantized filters, and the remaining filters in the tile are quantized into 4-bit. This reduces the indexing overhead and improves computation throughput. Through this method, a throughput improvement of approximately 39% was achieved on Xilinx Zynq-7000 XC7Z020.

4.3.2 Other compression methods

Pruning is a prominent CNN compression technique that effectively reduces model size and computational workload by removing redundant weights, particularly in accelerators. However, because many of the weights become zero, it can be even more effective in reducing computational cost in conjunction with the zero-skipping technique [58]. The zero-skipping technique skips computations when the data are zero, thereby avoiding unnecessary operations. When used alongside weight pruning, which sets many weights to zero, this approach further optimizes computational efficiency. Algorithms have also been designed to optimize CONV operations, which account for the majority of computations in CNNs. One notable example is the Winograd algorithm [60], which is a mathematical technique used in CNNs to optimize and enhance CONV operations, primarily by utilizing MM. This method aims to minimize multiplication and memory requirements, thereby providing fast computation and energy efficiency and plays a crucial role in achieving high performance, particularly on FPGA and other hardware accelerators. Most accelerator compression studies apply a single compression technique. However, some studies have designed accelerators by integrating two compression techniques. Meng et al. [55] designed an accelerator that exclusively utilizes on-chip memory without external memory access, in contrast to conventional CNN accelerators that access external memory to store the parameters. To achieve optimization using only on-chip memory, a method that combines quantization and structured sparsity was proposed. First, the weights are divided into small groups. Within each group, the smallest weights are set to zero, thus sparsing that particular group. To achieve high element-wise sparsity, the importance of the weights is evaluated by considering the relative magnitudes of all the surviving weights within the same layer. Based on the importance scores assigned, weights are globally pruned, starting from those with the smallest scores, thereby gradually increasing sparsity. Next, quantization is applied to approximate the weights using small integers. This approach simultaneously performs structured pruning and quantization, leveraging the advantages of both techniques. They, thus, achieved a significantly more compressed accelerator design than previous CNN accelerators, resulting in a 2.34× higher GOPS in ImageNet classification.

5 Hardware architecture of CNN accelerators on FPGA platforms

In this section, we introduce various architectures (i.e., fused-layer architecture, multi-PU architecture, systolic array (SA) architecture, and CPU-FPGA collaborative computing architecture) to enhance the efficiency of the CNN accelerator. First, we introduce the fused-layer architecture [63–66], which fuses and processes operations of adjacent layers, and the multi-PU architecture [14, 67–69], which uses multiple processors to efficiently handle operations of various shapes. After that, we elaborate on the SA architecture [15–17, 70, 71], which uses a PE grid in a pipeline manner, and the CPU-FPGA collaborative computing architecture [72–75], which is designed to efficiently handle tasks specialized for both CPU and FPGA.

5.1 Fused-layer architecture

The design of existing CNN accelerators in FPGAs primarily optimize and evaluate parallelism, reusing the data in a single CONV layer [76]. However, this method requires access to off-chip memory to store the intermediate data between layers, resulting in high power consumption and long latency. To solve this problem, the fused-layer method performs operations by fusing adjacent layers. As shown in Fig. 6, the fused-layer method can efficiently reduce DRAM access by fusing connected Layer1 and Layer2 without processing each CONV individually. The fused-layer method is an approach that leverages the locality of convolution operations. To achieve this, Layer1 is designed to compute the outputs in the order required for the subsequent Layer2 operations, whereby Layer 1 performs operations on the input feature map tiles. The intermediate feature map produced by Layer1 is not stored in off-chip memory. Instead, it is directly utilized as the input feature map for Layer 2. Consequently, the fused-layer method reduces DRAM access by transmitting intermediate feature maps to the subsequent layer without storage. It also optimizes memory utilization by promptly discarding intermediate data after use in the next layer. The fused-layer method minimizes overall data transfer overhead, thereby contributing to a more streamlined and resource-efficient deep learning model. Alwani et al. [63] proposed a method for manipulating on-chip data inflow and a technique for fusing the processing of the subsequent CONV layers using a pyramid-shaped multi-layer sliding window. This fusion layer enables the on-chip caching of intermediate data, which can effectively reduce data transfer from off-chip memory. The fused-layer design using

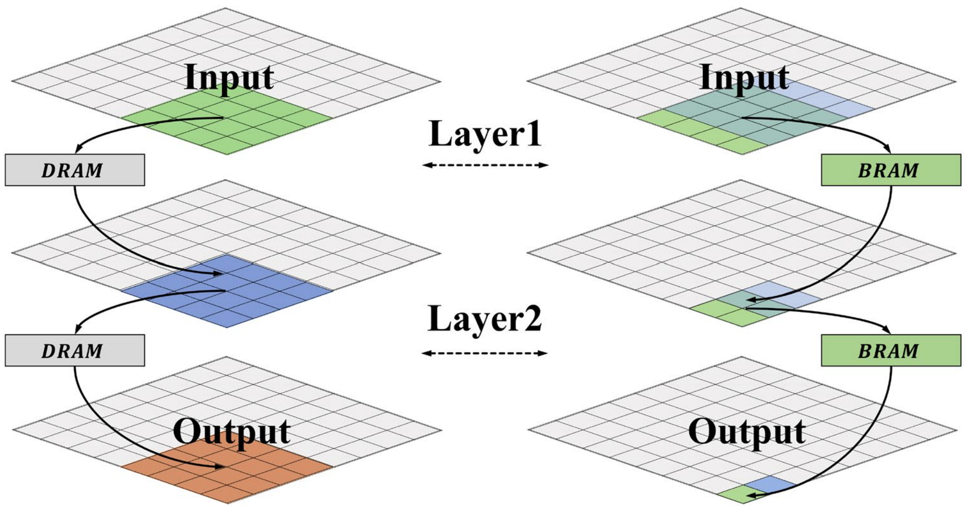


Fig. 6 An example of a fused-layer in two convolution layers from [63]

VGG-16 on the Xilinx Virtex-7 FPGA dramatically reduced off-chip usage from 77 MB to just 3.6 MB. Erdem et al. [64] optimally fused just two CONV layers. In other words, unlike a full pyramid, which fuses until one output pixel is generated, they used a tile-based truncated pyramid. This method can achieve a more efficient trade-off in memory usage by adjusting the tile size. Specifically, the tile-based truncated pyramid design, employing only two fused layers and VGG-16 on the Xilinx Zynq-7000 XC7Z020 FPGA, enhanced the computation to communication ratio (CCR) from 49.4% to 81.6%. As a result, unlike the full pyramid approach, which requires storing massive amounts of intermediate data in the on-chip memory, the tile-based truncated pyramid design is efficient, even on FPGAs with only 4.9 Mb BRAM size. Indirli et al. [65] proposed a configurable design of a fused-layer accelerator that can accelerate more than two layers simultaneously. The proposed design uses half-precision and output tiling to reduce memory usage. Output tiling partitions the output feature map into square tiles, enabling parallel computation of the divided tiles. This method allows more data to be processed simultaneously. Specifically, when using VGG-16 on the Xilinx XCZU15EG FPGA, this method achieves 42× speedup and reduces transfers from external memory by 95x compared to a single-layer design. However, if the same optimization scheme is applied to all layers, the different feature maps and parameter sizes used in each layer of CNNs, present a limitation in efficient acceleration as CNNs process operations of different shapes. To overcome this limitation, Nguyen et al. [14] designed different mixed-precision and layer-specific architectures for each layer to reduce the DRAM access caused in transmitting feature maps. They also proposed streaming CONV, which allows the simultaneous computation of consecutive CONVs. As a result, they reduced the model size by a factor of 22.66× for YOLOv3 and 28.93 × for Tiny-YOLOv2. They also achieved 1.88 tera operations per second (TOPS) on the Xilinx Virtex-7 XC7VX485T. Wu et al. [66] proposed an architecture that combines CONV and shortcut layers. This architecture places a shortcut operation before the CONV operation and utilizes the input features from the previous layer stored in DRAM, which effectively reduces unnecessary DRAM accesses in the shortcut operation, thereby achieving a bandwidth reduction of approximately 38.03% on the Xilinx XCZU9EG.

5.2 Multi-PU architecture

In conventional CNN accelerator approaches, a single PU is used to serially process one CONV layer at a time. However, single-PU accelerators using the same processing structure cannot efficiently handle CNNs with various shapes (e.g., channel) and types (e.g., depth-wise CONV). Therefore, recent approaches have used multiple processors to process operations of various shapes. Figure 7 presents an example calculation for eight CONV layers of different shapes. The *x*- and *y*-axes represent time and PE utilization, respectively. In Fig. 7a, the use of a single PU is optimized only for specific CONVs, resulting in idle PEs and other CONV operations. Meanwhile, in Fig. 7b, the use of two PUs enables the parallel processing of operations with different shapes. Therefore, in terms of PE utilization and operation time, it is more efficient than a single-processor structure. Shen et al. [67] divided available hardware resources into smaller processors to efficiently handle CONV layers with different shapes. Such accelerator designs using multiple processors can process adjacent CONV in a pipeline manner, thus achieving high computational efficiency and 1.51× throughput for AlexNet on Xilinx Virtex-7 XC7VX690T. Wu et al. [68] proposed an accelerator targeting MobileNet, consisting of various types of CONVs (e.g., depth-wise, point-wise). They raised the issue of many PEs not being utilized when calculating the depth-wise CONV layers in

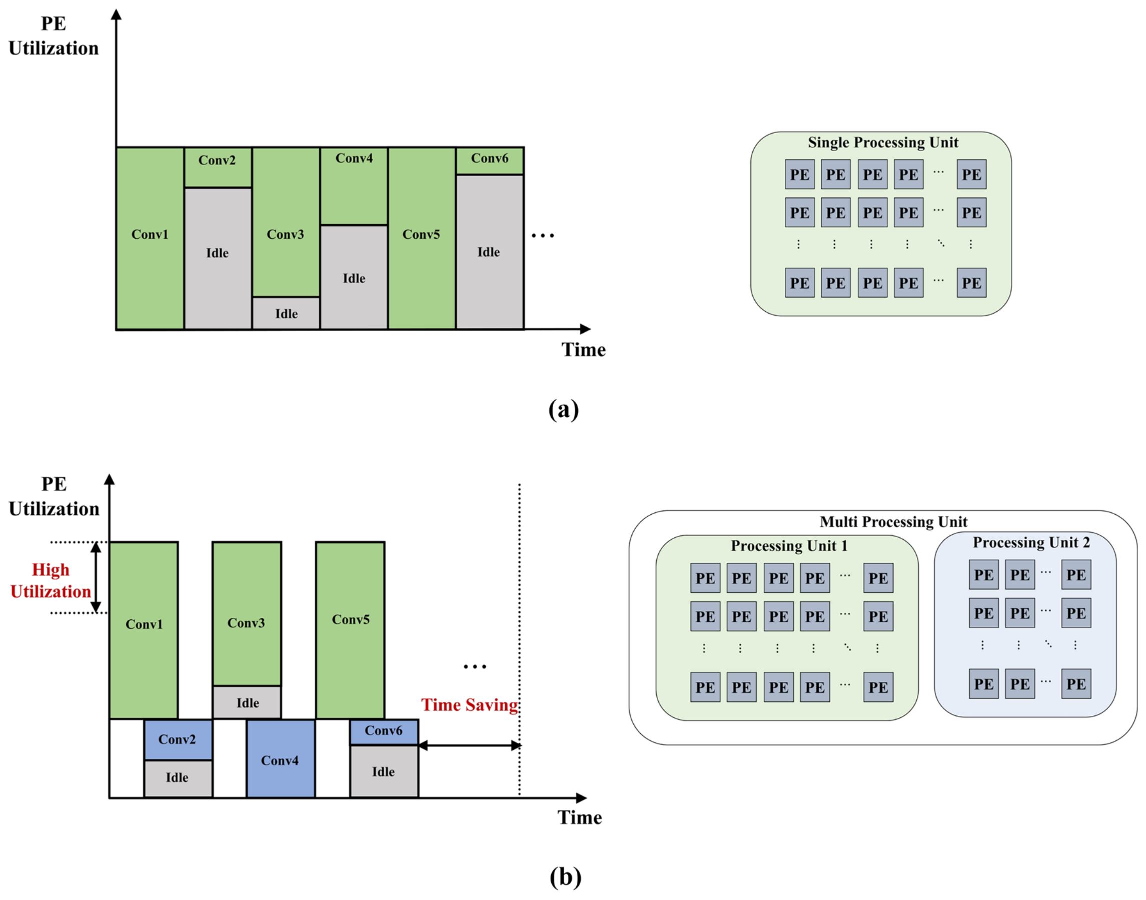


Fig. 7 An example of a network pipeline according to the processor for eight convolution layers from [68]. **a** Single convolution processor and **b** multi-convolution processors

a single-processor design. As a solution, they proposed the MobileNet accelerator that uses separate processors specifically designed for standard and depth-wise CONV layers instead of using a general CONV processor. Consequently, the proposed method achieved speedups of 8.4× and 33.6× over the CPU on Xilinx XCZU2EG and XCZU9EG, respectively. Qararyah et al. [69] proposed a hybrid architecture, whereby a single PE handles a single layer for the initial layers that exhibit greater heterogeneity; whereas in the remaining layers, a single PE handles multiple layers. As a result, the proposed accelerator architecture achieved 1.7× and 4.1× throughput improvements compared to the single PE accelerator structures for the MobileNetv1 model on Xilinx XCZU2EG and XCZU9EG, respectively. However, this architecture has several limitations. First, as the operations vary between layers in the model, the required number of PEs increases, causing significant resource consumption. Second, a significant memory bandwidth is required to drive multiple PEs simultaneously. Third, as the number of PEs increases, the overhead of the control logic grows. Therefore, in cases where the model consists of many layers that perform similar CONV operations, designing one PE to process multiple layers is efficient, as shown in Fig. 6a.

5.3 Systolic array architecture

The SA is a parallel-computing structure with multiple deeply pipelined PEs [15]. Each PE processes input data and forwards output data to the next PE. This structure has advantages in parallel computation and is often used in CNN accelerator implementations. Figure 8 shows the structure of a 2D SA and a schematic of the single PE used to implement a CNN accelerator. A 2D SA consists of multiple densely pipelined PEs, each composed of a buffer and accumulator. In each cycle, for one PE (x, y), the input data are passed to PE $(x + 1, y)$ and weight data are passed to PE $(x, y + 1)$. Moreover, each PE accumulates the product of the input and weight data passed from the adjacent PE (OUT_{xy}), and output data move outside through the PE array. This SA architecture can achieve high frequency by solving the timing issues encountered in massive parallelization through local interconnects and data transfers shifted between PEs. However, the mapping of a CNN model onto the SA is not

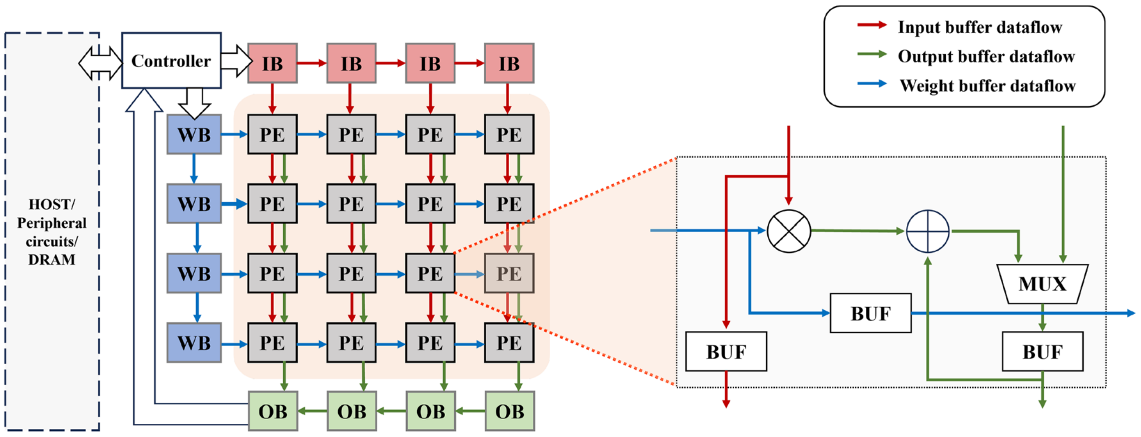


Fig. 8 Systolic array architecture

straightforward. Wei et al. [70] mapped a CNN model onto the SA through the following three stages: (1) find a feasible mapping; (2) select a PE array shape; and (3) determine the data reuse strategy. In the first stage, they found a feasible mapping for 3D CONV operations on 2D SA. In the second stage, they selected the PE array shape that influences the number of DSPs, clock frequency, and DSP utilization by determining the size of each dimension. Finally, by selecting an appropriate tiling size that allowed extensive data reuse, they found the SA configuration that yielded optimum throughput and hardware utilization. Utilizing the SA design, a comparable performance enhancement of 1.8× GOPS was achieved on the same Intel Arria 10 GT 1150 FPGA, in contrast to a previous VGG accelerator [43] that uses a comparable number of DSPs and LUTs. Chen et al. [17] introduced a new data flow called row stationary (RS) to minimize energy consumption due to data movement in SA. The RS approach decomposes a 2D CONV into multiple one-dimensional (1D) CONVs for processing. This maximizes the reuse of filters and feature maps, minimizing the cost of accumulating partial sums. Unlike traditional data flow methods, RS is flexible and can be adapted to a variety of CNN architectures. RS maximizes energy efficiency by fully utilizing the local storage of PE, direct communication between PEs, and spatial parallelism. In the experiments using the CNN configuration of AlexNet, the proposed RS data flow showed 1.4–2.5× greater energy efficiency in CONV operations and 1.3× greater efficiency in FC operations than conventional data flows. Zhang et al. [77] proposed a 2D SA design to improve frequency. The authors indicated that the critical path caused by the formation of a long DSP chain within the PE of the existing accelerator, is detrimental to the frequency. To solve this problem, they designed the SA accelerator that divides the data path of the DSP accumulation chain into multiple segments, selecting the sum of one segmentation to output through a multiplexer (MUX) in each cycle, thereby achieving a 1.29× higher frequency and 1,495 GOPS performance on the Xilinx KCU1500 platform for the VGG16 network. The specialized design of SA focusing on data reusability, can achieve efficient data processing and minimize data movement, resulting in high-throughput and energy-efficient characteristics. However, a significant drawback of SA is its poor hardware utilization performance in nonrepetitive operations. To address this issue, Selvam et al. [71] proposed a method called fully separable convolution (FuSeConv), which transforms the less data-reusable depth-wise separable CONV into a 1D CONV. FuSeConv decomposes the traditional CONV filter ($K * K * C$) into two groups of 1D filters ($K * 1 * C/D$ and $1 * K * C/D$), and partially sums them, thereby enabling mapping onto a 2D SA structure. Consequently, applying a depth-wise separable CONV to a 64×64 PE array, they achieved a 3× to 7× speedup on the MobileNet family (MobileNetV1-3, MnasNet).

5.4 CPU-FPGA collaborative computing architecture

The CPU-FPGA collaborative computing architecture leverages the collaboration between a CPU and an FPGA to provide high-performance computing. This architecture offers higher power efficiency and throughput than a single computing platform. The CPU is responsible for handling software-oriented flexible algorithms with low parallelization, executing general instruction-based tasks, whereas FPGA accelerates specific tasks with high hardware parallelism. When designing an accelerator, considering the roles of the CPU and FPGA is essential in enabling efficient distribution of computational tasks, to achieve high-performance computing and flexible programming. Qiao et al. [72] proposed a technique that accelerates CNNs by allocating specialized computations to FPGA and CPU. They designed the MM accelerator on FPGA to accelerate the CONV and FC layer operations, with the remaining tasks performed on the CPU. In addition, they addressed the latency caused by frequent data copying, by employing a virtual memory approach to allow the CPU and FPGA to operate in the same memory space using DMA to transfer data to on-chip memory. Comparing their approach to those implemented on general-purpose devices, they achieved a performance improvement of 3.54× compared to using the Intel Xeon X5675 CPU, and an energy efficiency improvement of 4.7× compared to using the Nvidia K20 GPGPU. Wang et al. [73] proposed a design approach for the YOLOv2 network, whereby non-performance-critical layers, such as max-pooling and concatenation, are assigned to the CPU, dedicating all FPGA hardware resources to accelerating the CONV layers. They introduced interbatch layer-wise pipelining, which enables the CPU and FPGA to concurrently process operations from different layers when multiple input images are present, achieving a throughput improvement of 1.17× compared with the model in which max-pooling was implemented on Intel Arria 10 GX 1150 FPGA. Meloni et al. [74] introduced a design approach to accelerate operations such as CONV and pooling by implementing a convolution engine (CE) in the PL of the FPGA. They efficiently handled off-chip memory communication, specifically, the data marshaling layer and fully connected layer operations, by leveraging the SIMD vectorization capabilities of the ARM Cortex-A9 NEON vector unit in the PS of the FPGA. This approach enabled them to handle layers that cannot be easily implemented in PL. Liu et al. [75] proposed a PS-PL co-design structure, which comprises a PS that manages the model parameters and configuration and a PL that handles layer operations. The PS conveys the configuration and parameters to the PL through the AXI high-performance (HP) and AI general performance (GP) interfaces, respectively. On the PL side, the transferred parameters are used to process compute-intensive operations in the CONV, pooling, and FC layers for acceleration. Notably, considering the massive FC layer operation, they suggested a method for determining whether to compute using PS or PL by identifying trade-offs to prevent transmission overhead. Through this design technique, inferences for various networks (e.g., AlexNet, VGG, and MobileNet) could be made. In their implementation, they achieved 206 GOPS for VGG network on Xilinx Zynq-7000 XC7Z045 FPGA.

6 Performance comparison

A hardware performance comparison of various CNN accelerators is presented in Table 2. All accelerators are organized based on the implemented CNN models, documenting the data formats of the weights and activations that constitute these models. GOPS was used as a unit to measure throughput, which represents the computational capacity of a CNN accelerator per second. The throughput measurement, denoted as (conv), focuses specifically on the CONV layers rather than the entire network. The inference speed was based on FPS, which calculates the number of images that can be processed per second. All the resources (DSPs, LUTs, FFs, BRAMs) indicated HW utilization within the implemented FPGA chip. Notably, some studies such as those of Zhang et al. [44] and Song et al. [48] have implemented the VGG model utilizing a considerable amount of hardware resources, including DSP and LUT, to achieve high throughput performance, whereas Guo et al. [49] demonstrated the implementation of the same model with significantly fewer resources. These approaches highlight the efficiency and flexibility of utilizing hardware resources and demonstrate the practical applicability of accelerator designs. The performance difference (e.g., GOPS) between the studies of Zhang et al. [76] and Liu et al. [79] clearly demonstrates the advantages of applying quantization. Liu et al. [79] quantized weight and activation to 8-bit and 16-bit fixed-point formats, respectively, achieving 222.1 GOPS. In contrast, Zhang et al. [76] did not apply quantization (floating point 32-bit), obtaining a throughput of 61.6 GOPS, which indicates a decrease of approximately 72% compared to [79]. Additionally, the method of Zhang et al. [76] excelled by 152% in resource efficiency (GOPS/Slice) compared to that of Liu et al. [79]. Zhang et al. [76] applied various techniques such as tiling, double buffering, and unrolling to minimize memory access and computation costs but did not achieve high throughput. Throughput can be improved by employing quantization and PU design approaches, such as multi-PU and SA configurations. Qiu et al. [53] and Liu et al. [75] utilized the same fixed-point format to explore the impact of bits allocated to the network ([53]: 16-bit, [75]: 8-bit) on GOPS. Research [75] applying lower-precision quantization, demonstrated an improvement of approximately 9% in GOPS compared with [53]. The presence or absence of quantization can significantly affect accelerator throughput, and the allocation of bits can be a key factor in enhancing throughput. However, it is important to note that low-precision quantization may lead to a decrease in accuracy; thus, in designing an accelerator the trade-off between accuracy and throughput should be considered. Sun et al. [54] demonstrated the advantages of versatility by performing three different network inferences on a single FPGA platform to explore the relationships between network components. Additionally, to efficiently perform mixed-precision computing, 4-bit and 8-bit data were reordered. If 8-bit operations can be supported by reusing 4-bit operators, a higher throughput can be achieved.

Two studies by Nguyen et al. [13, 14] reduced DRAM access with aggressive mixed precision and layer fusing for all CONV operations, thereby demonstrating 32.5× and

Table 2 Performance and resource utilization of CNN accelerator designs

Previous research	Model	Data format (w/a)	Frequency (MHz)	Power (W)	Throughput (GOPS)	Speed (FPS)	DSPs (The number of usage)	LUTs (The number of usage)	FlipFlops (The number of usage)	BRAM (Mb)	FPGA chip
[72]	AlexNet [2]	Float 32	150	14.4	77.8	–	836 (92.8%)	183,190 (83.8%)	21,8296 (49.9%)	14.5 (73.8%)	Xilinx Zynq-7000 XC7Z045
[47]	AlexNet [2]	Fixed 16	303	–	1382.0	–	1476 (97.2%)	503,808 (59.0%)	100,7616 (59.0%)	49.7 (91.6%)	Intel Arria 10 GX 1150
[76]	AlexNet [2]	Float 32	100	18.6	61.6 (conv)	46.3 (conv)	2240 (80.0%)	186,251 (61.4%)	205,704 (33.9%)	20 (53.9%)	Xilinx Virtex-7 XC7VX485T
[78]	AlexNet [2]	Fixed 8	300	17.7	290.4 (conv)	9.7	696 (40.3%)	101,953 (44.3%)	127,577 (27.7%)	7.1 (63.6%)	Xilinx Zynq Ultrascale+ XCZU7EV
[79]	AlexNet [2]	Fixed 8/16	100	24.8	222.1	–	1436 (39.9%)	115,036 (26.6%)	174,412 (20.1%)	21.1 (39.7%)	Xilinx Virtex-7 XC7VX690T
[37]	AlexNet [2]	Fixed 16	156	30.2	565.9	391	2144 (59.6%)	273,805 (63.2%)	262,703 (30.3%)	34.4 (65.1%)	Xilinx Virtex-7 XC7VX690T
[67]	AlexNet [2]	Float 32	100	7.6	85.2 (conv)	64.0 (conv)	2443 (87.2%)	176,876 (58.3%)	270,991 (44.6%)	29.2 (39.4%)	Xilinx Virtex-7 XC7VX485T
	AlexNet [2]	float 32	100	10.2	113.9 (conv)	85.6 (conv)	3177 (88.3%)	236,877 (54.7%)	348,049 (40.2%)	25.8 (48.8%)	Xilinx Virtex-7 XC7VX690T
[70]	AlexNet [2]	Float 32	239.6	–	360.4	246.9	1290 (85.0%)	700,000 (82.0%)	1,400,000 (81.9%)	47.2 (86.0%)	Intel Arria 10 GT 1150
	VGG-16 [7]	Fixed 8/16	231.85	–	1171.3	37.2	1500 (98.8%)	626,000 (73.0%)	1,252,000 (73.3%)	40.1 (61.0%)	Intel Arria 10 GT 1150
[53]	VGG-16 [7]	Fixed 16	150	9.6	187.8 (conv)	4.5	780 (89.2%)	182,616 (83.5%)	127653 (29.2%)	17.5 (86.7%)	Xilinx Zynq-7000 XC7Z045
[49]	VGG-16 [7]	Fixed 8	214	3.5	84.3 (conv)	2.5	190 (86.4%)	29,867 (56.1%)	35,489 (33.4%)	3.1 (61.1%)	Xilinx Zynq-7000 XC7Z020
[39]	VGG-16 [7]	Fixed 32	200	8.04	229.2 (conv)	7.0	256 (100.0%)	392,740 (83.7%)	785,480 (83.7%)	8.4 (16.8%)	Intel Stratix V 5SGXA7
[80]	VGG-16 [7]	Float 32	200	28.8	160.0	–	1600 (63.5%)	141,394 (51.6%)	140,506 (51.6%)	16.2 (50.4%)	Xilinx Zynq Ultrascale+ XCZU9EG
[44]	VGG-16 [7]	Fixed 16	150	26	354.0	15.3	2833 (78.7%)	346,560 (79.9%)	311,904 (36.0%)	44.9 (84.9%)	Xilinx Virtex-7 XC7VX690T
[43]	VGG-16 [7]	Fixed 8/16	150	–	645.3	20.8	1518 (100.0%)	322,000 (37.7%)	644,000 (37.7%)	38 (70.0%)	Intel Arria 10 GX 1150
[48]	VGG-16 [7]	Fixed 8	100	–	-	110.6	4096 (59.9%)	922,000 (78.0%)	165,000 (7.0%)	26.2 (72.6%)	Xilinx Virtex UltraScale+ XCVU9P

Table 2 (continued)

Previous research	Model	Data format (w/a)	Frequency (MHz)	Power (W)	Throughput (GOPS)	Speed (FPS)	DSPs (The number of usage)	LUTs (The number of usage)	FlipFlops (The number of usage)	BRAM (Mb)	FPGA chip
[75]	VGG-16 [7]	Fixed 8	150	6.2	206.0	6.8	787 (87.4%)	154,000 (70.5%)	128,000 (29.3%)	18.9 (98.5%)	Xilinx Zynq-7000 XC7Z045
	MobileNetV1 [23]	fixed 8	150	6.2	13.0	5.5	787 (87.4%)	154,000 (70.5%)	128000 (29.3%)	18.9 (98.5%)	Xilinx Zynq-7000 XC7Z045
[55]	MobileNetV1 [23]	Fixed 4	133	30.4	3013.0	2648.0	1730 (100.0%)	2,671,800 (77.1%)	5,343,600 (77.1%)	27.8 (21.0%)	Intel Stratix 10 GX 10 M
[68]	MobileNetV1-SSD [81]	Fixed 8	430	–	-	31.0	212 (88.3%)	31,198 (66.1%)	46,809 (49.6%)	5.1 (96.7%)	Xilinx Zynq Ultrascale+ XCZU2EG
	MobileNetV1-SSD [81]	Fixed 8	333	–	-	124.3	2070 (82.1%)	161,944 (59.1%)	301,416 (55.0%)	27.1 (84.5%)	Xilinx Zynq Ultrascale + XCZU9EG
	MobileNetV2 [24]	Fixed 8	430	–	-	205.3	212 (88.3%)	31,198 (66.1%)	46,809 (49.6%)	5.1 (96.7%)	Xilinx Zynq Ultrascale+ XCZU2EG
	MobileNetV2 [24]	Fixed 8	333	–	-	809.8	2070 (82.1%)	161,944 (59.1%)	301,416 (55.0%)	27.1 (84.5%)	Xilinx Zynq Ultrascale + XCZU9EG
[82]	MobileNetV2 [24]	Fixed16	150	3.11	10.9	17	206 (93.6%)	41,622 (78.2%)	47,331 (44.5%)	4.5 (91.4%)	Xilinx Zynq-7000 XC7Z020
[54]	MobileNetV2 [24]	Mixed precision	100	3.0	29.3	49.2	214 (97.0%)	39,100 (74.0%)	–	4.4 (90.0%)	Xilinx Zynq-7000 XC7Z020
	ResNet-18 [1]	Mixed precision	100	3.0	46.8	12.9	214 (97.0%)	39,100 (74.0%)	–	4.4 (90.0%)	Xilinx Zynq-7000 XC7Z020
	ResNet-50 [1]	Mixed precision	100	3.0	63.6	7.8	214 (97.0%)	39,100 (74.0%)	–	4.4 (90.0%)	Xilinx Zynq-7000 XC7Z020
[67]	SqueezeNet [25]	Fixed 16	170	7.2	909.7 (conv)	1173.0 (conv)	2880 (80.0%)	643,908 (92.9%)	–	22.9 (43.2%)	Xilinx Virtex-7 XC7VX690T
[35]	CloudScout [83]	Fixed 16	115.4	4.5	–	6.9	1158 (67.0%)	59,195 (25.7%)	20,771 (4.5%)	1.4 (12.5%)	Xilinx Zynq Ultrascale + XCZU7EV
[40]	VDSR [26]	Fixed 4/8	200	–	-	–	265 (73.6%)	69,316 (98.2%)	4912 (3.5%)	4.78 (61.1%)	Xilinx Zynq UltraScale + XCZU3EG
[66]	YOLO-like	Fixed 8	100	–	14.3	–	277 (11.0%)	65,780 (24.0%)	–	5.7 (17.6%)	Xilinx Zynq Ultrascale+ XCZU9EG
[13]	tiny YOLOv2	Mixed precision	200	8.7	464.7	66.6	1026 (36.6%)	86,000 (28.3%)	60,000 (9.9%)	18.5 (49.8%)	Xilinx Virtex-7 XC7VX485T

Table 2 (continued)

Previous research	Model	Data format (w/a)	Frequency (MHz)	Power (W)	Throughput (GOPS)	Speed (FPS)	DSPs (The number of usage)	LUTs (The number of usage)	FlipFlops (The number of usage)	BRAM (Mb)	FPGA chip
[14]	sim YOLOv2	Mixed precision	200	–	1877.0	109.3	829 (29.6%)	245,300 (80.8%)	–	22.4 (60.4%)	Xilinx Virtex-7 XC7VX485T

131.3× higher throughput than the YOLO accelerator of Wu et al. [66]. Meng et al. [55] proposed an accelerator that exclusively leverages on-chip memory, eliminating DRAM access by applying high element-wise sparsity and low-precision quantization. Although this approach may introduce some loss in accuracy compared to other accelerators, it significantly reduces latency and achieves high throughput. Pacini et al. [35] significantly reduced the usage of filter and feature map buffers through a hierarchical cache system, implementing an accelerator with only 1.4 Mb of memory usage. Although the hierarchical cache system significantly reduces memory usage, power consumption is greater compared to other accelerators because of the extensive use of LUTs and registers. Sun et al. [54] significantly reduced resource utilization and power consumption by integrating a mixed-precision quantization technique. Zhang et al. [44] despite utilizing memory optimization techniques such as tiling, were unable to prevent frequent off-chip memory access and a significant BRAM consumption of 44.9 Mb. To address these issues, we believe that employing a hierarchical memory structure or fused-layer techniques can substantially reduce off-chip memory access and enhance BRAM utilization. Wei et al. [70] implemented AlexNet and VGG through an SA design, achieving high frequency and high throughput. Compared with the CONV inference of AlexNet designed by Zhang et al. [76], they achieved 2.4× higher operating frequency and 5.9× higher throughput. Compared with the VGG designed by Ma et al. [43], with the same bit precision [70], obtained 1.8× superior GOPS. Nevertheless, notable limitations are the high computational load and significantly increased hardware resource consumption, with an average of 3.38× and 1.45× compared to [43, 76], respectively. Liu et al. [75] proposed a flexible inference of VGG and MobileNet using the same hardware implementation by effectively coordinating the PS. Meanwhile, Wu et al. [68] demonstrated the advantages of reusability by implementing various forms of MobileNet inference using a MobileNet-dedicated accelerator with a multi-PU design. These studies accelerate computations significantly, achieving performance gains of 147.2× and 16.5×, respectively, compared to [54, 75]. They also achieved significant reductions in hardware usage, including over threefold decrease in DSP and BRAM utilization and more than a five-fold reduction in LUTs, while exhibiting a remarkable 37.3× increase in FPS when implemented on the ZU2EG platform, surpassing the performance demonstrated by Liu et al. [75]. Thus, depending on the specific CNN and FPGA resource specifications, various network compression and optimization techniques can be applied for computation and memory. In conclusion, by selecting the appropriate PE and data flow that match the characteristics of CNNs, the desired CNN accelerator can be designed.

7 Conclusion

This paper investigates and analyzes previous studies with regard to architecture and optimization techniques of FPGA-based CNN accelerators. By employing optimization techniques such as parallel computing, memory access optimization, and reduction of computational workload, the overall processing speed can be improved, thereby minimizing the computational cost and maximizing the performance of the accelerator. Furthermore, through the design of accelerator architectures, data flow, memory structure, and computational types can be optimized, to reduce computational workload and enhance parallel processing.

The current research accomplishments on FPGA-based CNN accelerators have been instrumental in facilitating the commercial deployment of CNN inference, characterized by low power consumption and high throughput within embedded devices. However, CNN accelerators, with inherent features for dedicated network optimization, exhibit notably reduced compatibility compared to GPUs. Therefore, it is imperative that future research be steered toward the development of versatile accelerators capable of operating a variety of CNNs within a singular FPGA platform, and trainable accelerators that encompass back-propagation operations. In addition, it has become increasingly important to advance vision transformer (ViT) accelerator research, particularly for optimizing operations to facilitate multi-head self-attention (MSA) and FC in ViT.

Acknowledgements This research was partly supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2023-RS-2022-00156295) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation) and the National R &D Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2022M3I7A1078936).

Authors' contributions In the research project, Hyeonseok Hong and Dahun Choi equally contributed to conceptualization, drafting, data curation, analysis, visualization, and validation. Namjoon Kim, Haein Lee, Beomjin Kang, and Huibeom Kang were involved in data curation and draft preparation, with Kim and Lee also contributing to visualization. Hyun Kim oversaw the project, managed administration, secured funding, provided resources, and reviewed and edited the manuscript.

Data availability Data availability is not applicable to this article as no new data were created or analysed in this study.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
2. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (2017)
3. Choi, J., Chun, D., Kim, H., Lee, H.-J.: Gaussian yolov3: an accurate and fast object detector using localization uncertainty for autonomous driving. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 502–511 (2019)
4. Redmon, J., Farhadi, A.: Yolov3: An Incremental Improvement. arXiv preprint arXiv:1804.02767 (2018)
5. Bolya, D., Zhou, C., Xiao, F., Lee, Y.J.: Yolact: real-time instance segmentation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 9157–9166 (2019)
6. Lee, S.I., Kim, H.: Gaussianmask: uncertainty-aware instance segmentation based on Gaussian modeling. In: Proceedings of the 26th International Conference on Pattern Recognition (ICPR 2022) (2022)
7. Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556 (2014)
8. Kim, N.J., Kim, H.: FP-AGL: filter pruning with adaptive gradient learning for accelerating deep convolutional neural networks. IEEE Trans Multimed. **25**, 5279–5290 (2023)
9. Chun, D., Choi, J., Lee, H.-J., Kim, H.: CP-CNN: computational parallelization of CNN-based object detectors in heterogeneous embedded systems for autonomous driving. IEEE Access **11**, 52812–52823 (2023)
10. Guo, K., Zeng, S., Yu, J., Wang, Y., Yang, H.: A Survey of FPGA-Based Neural Network Inference Accelerator. arXiv preprint arXiv:1712.08934 (2018)
11. Choquette, J., Gandhi, W., Giroux, O., Stam, N., Krashinsky, R.: Nvidia a100 tensor core GPU: performance and innovation. IEEE Micro **41**(2), 29–35 (2021). https://doi.org/10.1109/MM.2021.3061394
12. Kim, H.: Review of optimal convolutional neural network accelerator platforms for mobile devices. J. Comput. Sci. Eng. **16**(2), 113–119 (2022)
13. Nguyen, D.T., Nguyen, T.N., Kim, H., Lee, H.-J.: A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection. IEEE Trans. Very Large Scale Integr. Syst. **27**(8) (2019) 1861–1873
14. Nguyen, D.T., Kim, H., Lee, H.-J.: Layer-specific optimization for mixed data flow with mixed precision in FPGA design for CNN-based object detectors. IEEE Trans. Circuits Syst. Video Technol. **31**(6), 2450–2464 (2021)
15. Rahman, A., Lee, J., Choi, K., Efficient FPGA acceleration of convolutional neural networks using logical-3d compute array. In: 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1393–1398. IEEE (2016)
16. Ma, Y., Suda, N., Cao, Y., Seo, J.-S., Vrudhula, S., Scalable and modularized RTL compilation of convolutional neural networks onto FPGA. In: 2016 26th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–8. IEEE (2016)
17. Chen, Y.-H., Emer, J., Sze, V.: Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks. ACM SIGARCH Comput. Archit. News **44**(3), 367–379 (2016)
18. Wang, J., Lou, Q., Zhang, X., Zhu, C., Lin, Y., Chen, D., Design flow of accelerating hybrid extremely low bit-width neural network in embedded FPGA. In: 2018 28th international conference on field programmable logic and applications (FPL), pp. 163–1636. IEEE (2018)

19. Ki, S., Park, J., Kim, H.: Dedicated FPGA implementation of the Gaussian TinyYOLOv3 accelerator. IEEE Trans. Circuits Syst. II Express Briefs **70**(10), 3882–3886 (2023)
20. Mittal, S.: A survey of FPGA-based accelerators for convolutional neural networks. Neural Comput. Appl. **32**(4), 1109–1139 (2020)
21. Kuon, I., Tessier, R., Rose, J.: FPGA architecture: survey and challenges. Found. Trends Electron. Des. Autom. **2**(2), 135–253 (2008)
22. Jang, J.-H., Shin, J., Park, J.-T., Hwang, I.-S., Kim, H.: In-depth survey of processing-in-memory architectures for deep neural networks. J. Semicond. Technol. Sci. **23**(5), 322–339 (2023)
23. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
24. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C.: Mobilenetv2: inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510–4520 (2018)
25. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-Level Accuracy with 50x Fewer Parameters and < 0.5 mb Model Size. arXiv preprint arXiv:1602.07360 (2016)
26. Kim, J., Lee, J.K., Lee,, K.M.: Accurate image super-resolution using very deep convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1646–1654 (2016)
27. Park, J., Bin, K., Lee, K.: mGEMM: low-latency convolution with minimal memory overhead optimized for mobile devices. In: Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services, pp. 222–234 (2022)
28. Papaphilippou, P., Luk, W.: Accelerating database systems using FPGAs: a survey. In: 2018 28th International Conference on Field Programmable Logic and Applications (FPL), pp. 125–1255. IEEE (2018)
29. Xilinx, Getting started with Alveo data center accelerator cards, bit.ly/48gwXiT, pDF document (2022)
30. Intel, Intel acceleration stack quick start guide for intel programmable acceleration card with Intel Arria 10 gx FPGA, bit.ly/48gwXiT, PDF document (2018)
31. Seng, K.P., Lee, P.J., Ang, L.M.: Embedded intelligence on FPGA: survey, applications and challenges. Electronics **10**(8), 895 (2021)
32. Shawahna, A., Sait, S.M., El-Maleh, A.: FPGA-based accelerators of deep learning networks for learning and classification: a review. IEEE Access **7**, 7823–7859 (2018)
33. Jinghong, D., Yaling, D., Kun, L.: Development of image processing system based on DSP and FPGA. In: 2007 8th International Conference on Electronic Measurement and Instruments, pp. 2–791. IEEE (2007)
34. Ryu, S., Oh, Y., Kim, J.-J., Mobileware: a high-performance mobilenet accelerator with channel stationary dataflow. In: 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pp. 1–9. IEEE (2021)
35. Pacini, T., Rapuano, E., Dinelli, G., Fanucci, L.: A multi-cache system for on-chip memory optimization in FPGA-based CNN accelerators. Electronics **10**(20), 2514 (2021)
36. Motamedi, M., Gysel, P., Akella, V., Ghiasi, S., Design space exploration of FPGA-based deep convolutional neural networks. In: 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 575–580. IEEE (2016)
37. Li, H., Fan, X., Jiao, L., Cao, W., Zhou, X., Wang, L.: A high performance FPGA-based accelerator for large-scale convolutional neural networks. In: 2016 26th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–9. IEEE (2016)
38. Jia, X., Zhang, Y., Liu, G., Yang, X., Zhang, T., Zheng, J., Xu, D., Wang, H., Zheng, R., Pareek, S., et al.: XVDPU: a high performance CNN accelerator on the versal platform powered by the AI engine. In: 2022 32nd International Conference on Field-Programmable Logic and Applications (FPL), pp. 01–09. IEEE (2022)
39. Podili, A., Zhang, C., Prasanna, V., Fast and efficient implementation of convolutional neural networks on FPGA. In: 2017 IEEE 28th International Conference on Application-Specific Systems, Architectures and Processors (ASAP), pp. 11–18. IEEE (2017)
40. Li, G., Liu, Z., Li, F., Cheng, J.: Block convolution: toward memory-efficient inference of large-scale CNNs on FPGA. IEEE Trans. Comput.-Aid. Des. Integr. Circuits Syst. **41**(5), 1436–1447 (2021)
41. Bai, L., Zhao, Y., Huang, X.: A CNN accelerator on FPGA using depthwise separable convolution. IEEE Trans. Circuits Syst. II Express Briefs **65**(10), 1415–1419 (2018)
42. Fan, H., Ferianc, M., Que, Z., Li, H., Liu, S., Niu, X., Luk, W.: Algorithm and hardware co-design for reconfigurable CNN accelerator. In: 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 250–255. IEEE (2022)
43. Ma, Y., Cao, Y., Vrudhula, S., Seo, J.-s.: Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks. In: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 45–54 (2017)
44. Zhang, C., Sun, G., Fang, Z., Zhou, P., Pan, P., Cong, J.: Caffeine: toward uniformed representation and acceleration for deep convolutional neural networks. IEEE Trans. Comput.-Aid. Des. Integr. Circuits Syst. **38**(11), 2072–2085 (2018)
45. Basalama, S., Sohrabizadeh, A., Wang, J., Guo, L., Cong, J.: FlexCNN: an end-to-end framework for composing CNN accelerators on FPGA. ACM Trans. Reconfig. Technol. Syst. **16**(2), 1–32 (2023)
46. Gao, M., Yang, X., Pu, J., Horowitz, M., Kozyrakis, C., Tangram: optimized coarse-grained dataflow for scalable NN accelerators. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 807–820 (2019)
47. Aydonat, U., O'Connell, S., Capalija, D., Ling, A.C., Chiu, G.R.: An opencl^{TM} deep learning accelerator on Arria 10. In: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 55–64 (2017)
48. Song, Y., Wu, B., Yuan, T., Liu, W.: A high-speed CNN hardware accelerator with regular pruning. In: 2022 23rd International Symposium on Quality Electronic Design (ISQED), pp. 1–5. IEEE (2022)
49. Guo, K., Sui, L., Qiu, J., Yu, J., Wang, J., Yao, S., Han, S., Wang, Y., Yang, H.: Angel-eye: a complete design flow for mapping CNN onto embedded FPGA. IEEE Trans. Comput.-Aid. Des. Integr. Circuits Syst. **37**(1), 35–47 (2017)
50. Park, J., Sung, W.: FPGA based implementation of deep neural networks using on-chip memory only. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1011–1015. IEEE (2016)
51. Vogel, S., Liang, M., Guntoro, A., Stechele, W., Ascheid, G.: Efficient hardware acceleration of CNNs using logarithmic data representation with arbitrary log-base. In: 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8. ACM (2018)
52. Lee, S., Sim, H., Choi, J., Lee, J.: Successive log quantization for cost-efficient neural networks using stochastic computing. In: Proceedings of the 56th Annual Design Automation Conference 2019, pp. 1–6 (2019)
53. Qiu, J., Wang, J., Yao, S., Guo, K., Li, B., Zhou, E., Yu, J., Tang, T., Xu, N., Song, S. et al.: Going deeper with embedded FPGA platform for convolutional neural network. In: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 26–35 (2016)

54. Sun, M., Li, Z., Lu, A., Li, Y., Chang, S.-E., Ma, X., Lin, X., Fang, Z.: FILM-QNN: Efficient FPGA acceleration of deep neural networks with intra-layer, mixed-precision quantization. In: Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 134–145 (2022)
55. Meng, J., Venkataramanaiah, S.K., Zhou, C., Hansen, P., Whatmough, P., Seo, J.-s.: FIXYFPGA: Efficient fpga accelerator for deep neural networks with high element-wise sparsity and without external memory access. In: 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), pp. 9–16. IEEE (2021)
56. Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M.A., Dally, W.J.: EIE: efficient inference engine on compressed deep neural network. ACM SIGARCH Comput. Archit. News **44**(3), 243–254 (2016)
57. Pellauer, M., Shao, Y.S., Clemons, J., Crago, N., Hegde, K., Venkatesan, R., Keckler, S.W., Fletcher, C.W., Emer, J.: Buffets: an efficient and composable storage idiom for explicit decoupled data orchestration. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 137–151 (2019)
58. Liu, M., Zhou, C., Qiu, S., He, Y., Jiao, H.: CNN accelerator at the edge with adaptive zero skipping and sparsity-driven data flow. IEEE Trans. Circuits Syst. Video Technol. **33**(12), 7084–7095 (2023)
59. Kim, N.J., Kim, H.: Trunk pruning: highly compatible channel pruning for convolutional neural networks without fine-tuning. IEEE Trans. Multimed. **26**, 5588–5599 (2023)
60. Wang, H., Lu, J., Lin, J., Wang, Z.: An FPGA-based reconfigurable CNN training accelerator using decomposable Winograd. In: 2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 1–6 (2023). https://doi.org/10.1109/ISVLSI59464.2023.10238574
61. Kim, S., Kim, H.: Zero-centered fixed-point quantization with iterative retraining for deep convolutional neural network-based object detectors. IEEE Access **9**, 20828–20839 (2021)
62. Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M.W, Keutzer, K.: A Survey of Quantization Methods for Efficient Neural Network Inference, arXiv preprint arXiv:2103.13630 (2021)
63. Alwani, M., Chen, H., Ferdman, M., Milder, P.: Fused-layer CNN accelerators. In: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 1–12. IEEE (2016)
64. Erdem, A., Babic, D., Silvano, C.: A tile-based fused-layer approach to accelerate DCNNs on low-density FPGAs. In: 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 37–40. IEEE(2019)
65. Indirli, F., Erdem, A., Silvano, C.: A tile-based fused-layer CNN accelerator for FPGAs. In: 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 1–4. IEEE (2020)
66. Wu, C.-B., Wu, R.-F., Chan, T.-W.: Hetero layer fusion based architecture design and implementation for of deep learning accelerator. In: 2022 IEEE International Conference on Consumer Electronics-Taiwan, pp. 63–64. IEEE (2022)
67. Shen, Y., Ferdman, M., Milder, P.: Maximizing CNN accelerator efficiency through resource partitioning. ACM SIGARCH Comput. Archit. News **45**(2), 535–547 (2017)
68. Wu, D., Zhang, Y., Jia, X., Tian, L., Li, T., Sui, L., Xie, D., Shan, Y.: A high-performance CNN processor based on FPGA for mobilenets. In: 2019 29th International Conference on Field Programmable Logic and Applications (FPL), pp. 136–143. IEEE (2019)
69. Qararyah, F., Azhar, M.W., Trancoso, P., Fibha: fixed budget hybrid CNN accelerator. In: 2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), pp. 180–190. IEEE (2022)
70. Wei, X., Yu, C.H., Zhang, P., Chen, Y., Wang, Y., Hu, H., Liang, Y., Cong, J.: Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In: Proceedings of the 54th Annual Design Automation Conference 2017, pp. 1–6 (2017)
71. Selvam, S., Ganesan, V., Kumar, P., FuSeConv: fully separable convolutions for fast inference on systolic arrays. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 651–656. IEEE (2021)
72. Qiao, Y., Shen, J., Xiao, T., Yang, Q., Wen, M., Zhang, C.: FPGA-accelerated deep convolutional neural networks for high throughput and energy efficiency. Concurr. Comput. Pract. Exp. **29**(20), e3850 (2017)
73. Wang, Z., Xu, K., Wu, S., Liu, L., Liu, L., Wang, D.: Sparse-YOLO: hardware/software co-design of an FPGA accelerator for YOLOv2. IEEE Access **8**, 116569–116585 (2020)
74. Meloni, P., Capotondi, A., Deriu, G., Brian, M., Conti, F., Rossi, D., Raffo, L., Benini, L.: Neuraghe: Exploiting CPU-FPGA synergies for efficient and flexible CNN inference acceleration on ZYNQ SOCs. ACM Trans. Reconfig. Technol. Syst (TRETS) **11**(3), 1–24 (2018)
75. Liu, W., Li, Y., Yang, Y., Zhu, J., Liu, L., Design an efficient DNN inference framework with PS-PL synergies in FPGA for edge computing. In: 2022 China Automation Congress (CAC), pp. 4186–4190. IEEE (2022)
76. Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., Cong, J.: Optimizing FPGA-based accelerator design for deep convolutional neural networks. In: Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 161–170 (2015)
77. Zhang, J., Zhang, W., Luo, G., Wei, X., Liang, Y., Cong, J.: Frequency improvement of systolic array-based CNNs on FPGAS. In: 2019 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–4. IEEE (2019)
78. Zhang, M., Li, L., Wang, H., Liu, Y., Qin, H., Zhao, W.: Optimized compression for implementing convolutional neural networks on FPGA. Electronics **8**(3), 295 (2019)
79. Liu, Z., Dou, Y., Jiang, J., Xu, J., Automatic code generation of convolutional neural networks in FPGA implementation, In: 2016 International Conference on Field-Programmable Technology (FPT), pp. 61–68. IEEE (2016)
80. Li, X., Cai, Y., Han, J., Zeng, X., A high utilization FPGA-based accelerator for variable-scale convolutional neural network. In: 2017 IEEE 12th International Conference on ASIC (ASICON), pp. 944–947. IEEE (2017)
81. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A.C.: SSD: single shot multibox detector. In: Computer Vision–ECCV 2016: 14th European Conference. Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14, pp. 21–37. Springer, Berlin (2016)
82. Sang, X., Ruan, T., Li, C., Li, H., Yang, R., Liu, Z.: A real-time and high-performance mobilenet accelerator based on adaptive dataflow scheduling for image classification. J. Real-Time Image Process. **21**(1), 4 (2024)
83. Giuffrida, G., Diana, L., de Gioia, F., Benelli, G., Meoni, G., Donati, M., Fanucci, L.: Cloudscout: a deep neural network for on-board cloud detection on hyperspectral images. Remote Sens. **12**(14), 2205 (2020)

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.