Article

Efficient CNN Accelerator Based on Low-End FPGA with Optimized Depthwise Separable Convolutions and Squeeze-and-Excite Modules

Jiahe Shen [1,2], Xiyuan Cheng [3], Xinyu Yang [2], Lei Zhang [2], Wenbin Cheng [1,2,*] and Yiting Lin [4,5]

1 School of Electronic Information, Zhongshan Institute, University of Electronic Science and Technology of China, Zhongshan 528402, China; 202222011528@std.uestc.edu.cn
2 School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China; 202322011503@std.uestc.edu.cn (X.Y.); 202222011529@std.uestc.edu.cn (L.Z.)
3 School of Automation, Guangdong University of Technology, Guangzhou 510006, China; 2112304365@mail2.gdut.edu.cn
4 Guangdong-Hong Kong-Macao Joint Laboratory for Data Science, Beijing Normal-Hong Kong Baptist University, Zhuhai 519087, China; yitingLin@ieee.org
5 Department of Computer Science, Hong Kong Baptist University, Hong Kong 999077, China
* Correspondence: chengwenbin@zsc.edu.cn

Abstract

With the rapid development of artificial intelligence technology in the field of intelligent manufacturing, convolutional neural networks (CNNs) have shown excellent performance and generalization capabilities in industrial applications. However, the huge computational and resource requirements of CNNs have brought great obstacles to their deployment on low-end hardware platforms. To address this issue, this paper proposes a scalable CNN accelerator that can operate on low-performance Field-Programmable Gate Arrays (FPGAs), which is aimed at tackling the challenge of efficiently running complex neural network models on resource-constrained hardware platforms. This study specifically optimizes depthwise separable convolution and the squeeze-and-excite module to improve their computational efficiency. The proposed accelerator allows for the flexible adjustment of hardware resource consumption and computational speed through configurable parameters, making it adaptable to FPGAs with varying performance and different application requirements. By fully exploiting the characteristics of depthwise separable convolution, the accelerator optimizes the convolution computation process, enabling flexible and independent module stackings at different stages of computation. This results in an optimized balance between hardware resource consumption and computation time. Compared to ARM CPUs, the proposed approach yields at least a 1.47× performance improvement, and compared to other FPGA solutions, it saves over 90% of Digital Signal Processors (DSPs). Additionally, the optimized computational flow significantly reduces the accelerator's reliance on internal caches, minimizing data latency and further improving overall processing efficiency.

Keywords: low-end FPGA; depthwise separable convolution; squeeze-and-excite module; scalable CNN accelerator; parallel computing

Academic Editor: Demos T. Tsahalis

Received: 7 August 2025
Revised: 24 September 2025
Accepted: 28 September 2025
Published: 1 October 2025

Citation: Shen, J.; Cheng, X.; Yang, X.; Zhang, L.; Cheng, W.; Lin, Y. Efficient CNN Accelerator Based on Low-End FPGA with Optimized Depthwise Separable Convolutions and Squeeze-and-Excite Modules. *AI* **2025**, *6*, 244. https://doi.org/10.3390/ai6100244

1. Introduction

With the rapid development of artificial intelligence technology, neural networks are profoundly transforming modern manufacturing systems [1–6]. In smart manufacturing environments, visual inspection systems face critical engineering challenges in surface defect detection, component positioning, and quality control. These systems must run on resource-constrained edge devices and process high-resolution images of industrial products (such as metal components, circuit boards, or precision mechanical parts) under strict latency constraints [7–9]. The engineering challenge lies in balancing real-time processing requirements with the limited power budget and thermal constraints typical in factory environments. In industrial visual inspection in particular, existing inspection systems may be limited by computing resources, resulting in low efficiency. Among the many neural network models, convolutional neural networks (CNNs) have demonstrated their powerful capabilities in various image processing tasks with their excellent performance and excellent generalization ability, especially in image recognition [10–13], classification [14–16], segmentation [17–19], and other fields [20–22]. Even more impressive is that CNNs can maintain sufficient performance after varying degrees of lightweight processing, making it one of the preferred solutions for deploying neural networks on resource-constrained platforms [23–25]. However, despite the many advantages of CNNs, their huge computing requirements also bring significant challenges. On hardware with very limited resources, the computational efficiency of neural networks is often severely restricted, limiting their application in certain scenarios [26–28]. Therefore, in order to improve computing efficiency, reduce computational load, and make full use of hardware performance, researchers have adopted various optimization techniques [29–31].

Among the numerous hardware platforms, FPGAs (Field-Programmable Gate Arrays) [32] are particularly favored due to their unique advantages. An FPGA is a type of integrated circuit that can be reprogrammed after manufacturing to perform specific digital functions. Unlike fixed-function chips, FPGAs offer high flexibility, parallel processing, and real-time performance, making them widely used in areas such as signal processing, embedded systems, cryptography, and artificial intelligence acceleration. Compared to CPUs and GPUs, FPGAs can perform parallel computations with lower power consumption, making them ideal for deploying CNNs on embedded platforms. The flexibility of FPGAs also allows them to be customized for specific neural network models and application scenarios, further improving computational efficiency and reducing resource consumption. Several studies have explored FPGA-based CNN accelerators. Ref. [33] used data quantization and bandwidth optimization, and they designed specialized controllers and compilers to run VGG16-SVD on a Xilinx Zynq ZC706, achieving a processing speed of 4.45 fps and a Top-5 accuracy of 86.66%. Ref. [34] used reversed-pruning, peak-pruning, and quantization strategies to compress the parameters of the AlexNet network. They employed sparse matrix storage to reduce the storage space of the compressed parameters, achieving a 28× model size reduction, and achieved 9.73 fps on a Xilinx ZCU104 evaluation board, showing significant improvements in latency, throughput, and energy consumption compared to CPU and GPU platforms. Ref. [35] designed a lightweight CNN, LiteCNN, by reducing input size, applying depthwise separable convolutions, and fusing batch normalization layers to reduce computational complexity and storage requirements. They employed knowledge distillation training to achieve 95.71% accuracy and 0.071 s per frame detection speed in real-time plant disease recognition with only 176 K parameters and 78.47 M floating-point operations, consuming just 2.41 W of power. Ref. [36] reduced computational complexity by using integer operations and shift operations, balancing real-time inference with resource consumption. Additionally, they designed efficient schemes for skip connections in residual blocks, requiring only 24 M integer operations (IOPs) and a model size of 0.17 MB, achieving 960 MOPS and 40 FPS performance on a Xilinx ZC706 SoC. Ref. [37] designed an efficient dynamically reconfigurable CNN accelerator that improved performance and reduced energy consumption by optimizing configuration layer sequences and designing unified high-speed interfaces. When deployed with the YOLOV2-TINY network on a Xilinx KV260 FPGA platform, the accelerator achieved 75.1929 GOPS per second with a peak power consumption of 5.25 W and power efficiency of 13.6219 GOPS/W in target detection tasks. Ref. [38] optimized MobileNet by introducing the Ad-depth unit with optimized structures, changing activation functions, and compressing channels, reducing hardware consumption by 41% and achieving 88.76% classification accuracy on a Xilinx Vertex7 xc7vx980. Ref. [39] accelerated convolution calculations using the Winograd algorithm and optimized up-sampling and squeeze-and-excite operations. They used the MobileNetV3-SSDLite network for object detection on a Xilinx Zynq XC7Z045, achieving 23.68 FPS and 33.14 FPS performance with MobileNetV3-Small and MobileNetV3-Large backbones, respectively. Ref. [40] proposed a residual lookup-based dot product approximation to accelerate CNN operations on FPGAs, achieving operation with less than 5% precision loss for ResNet-18. The RLDA-CNN accelerator, running on a ZCU104 evaluation board, achieved performance comparable to the NVIDIA Jetson AGX Orin without requiring Digital Signal Processors (DSPs). Ref. [41] also designed a DSP-free CNN accelerator by dynamically allocating PE arrays to ensure sufficient computational resources for different network layers. They used varying computation precisions based on layer importance, achieving 88.6% accuracy while significantly reducing system power consumption on the Xilinx Virtex 7 VC709 platform. However, the CNN architectures proposed in these studies typically require a large amount of hardware resources, and many implementations use high-end FPGAs. Even though the solutions in [40,41] avoid using DSPs entirely, their use of Logic Elements (LEs) and Block RAMs (BRAMs) is still significant. FPGAs with that sufficient LE and BRAM resources often also have enough DSP resources. High-end FPGAs not only cost more but also generally have higher clock frequencies and power consumption and require more powerful, sometimes even active cooling systems, which can become disadvantages in embedded platform scenarios. Therefore, reducing the hardware resource consumption and power consumption of CNN accelerators while ensuring performance has become a key direction of current research.

To tackle the challenge of excessive resource and power usage by CNNs on high-end FPGAs, we propose an efficient depthwise separable network accelerator tailored for use with low-end FPGA platforms. The accelerator achieves a flexible, configurable computational architecture by optimizing the computational flow of depthwise separable convolutions and SE modules. The expand module and pointwise convolution module in the accelerator achieve parallel processing through multiplier stacking, and different data-scheduling schemes are designed for different stages of the accelerator to improve computational efficiency. The paper analyzes in detail the impact of parallel computation on resource consumption and computational speed and proposes strategies to adjust parallelism according to actual needs, thereby achieving a balance between computational speed and resource consumption. The paper also optimizes data flow and caching mechanisms, reducing the need for intermediate data caching and improving computational efficiency. Most of the intermediate data generated at different stages of the accelerator can be directly read by the next stage without frequent memory read/write operations. Furthermore, the paper implements a straightforward valid signal synchronization scheme from front to back to ensure correct data transfer between modules, reducing the complexity of the control mechanism.

The main innovations and contributions of this paper are as follows:

- Aiming at the problem of large resource consumption and high power consumption of the existing convolutional neural network architecture on high-order FPGAs, a depthwise separable convolution network accelerator suitable for low-end FPGAs is proposed. The accelerator achieves a flexible and configurable computing architecture by optimizing the calculation process of the deep separable convolution and squeeze-and-excite module.

- The impact of parallel computing on resource consumption and computing speed is analyzed in detail, and a strategy of adjusting parallelism according to actual needs is proposed to achieve a balance between computing speed and resource consumption. The expand module and pointwise convolution module in the accelerator can be implemented in parallel by stacking multipliers, which improves the computational efficiency.
- By optimizing the data flow and caching mechanism, the caching requirements of intermediate data are reduced and the computational efficiency is improved. The intermediate data generated by different stages in the accelerator do not need to be cached, and they are read directly by the next level, avoiding frequent memory read and write operations. In addition, a simple forward–backward valid signal synchronization mechanism is proposed to ensure the correct transmission of data between modules and reduce the complexity of the control mechanism.

The remainder of this paper is organized as follows: Section 2 provides a brief introduction to depthwise separable convolution and the squeeze-and-excite module; Section 3 presents the proposed accelerator architecture; and Section 4 displays the experimental results and analysis.

2. Related Theories

2.1. Depthwise Separable Convolution

Depthwise separable convolution was introduced by Chollet in [42], which splits conventional convolutions into smaller convolutions. This approach significantly reduces the data and computational load of the convolution process at the cost of some feature extraction ability. In this paper, we use the simplest variant of depthwise separable convolution. For a convolution layer with M input channels and an output dimension of $W \times H \times N$, the conventional convolution employs N kernels of size $K \times K \times M$, requiring $K \times K \times M \times N$ parameters and performing $K \times K \times M \times N \times W \times H$ multiply-accumulate operations. In contrast, using depthwise separable convolution, the computation proceeds in two stages: first, a depthwise convolution is performed with a $K \times K \times M$ kernel, which is followed by a pointwise convolution with N kernels of size $1 \times 1 \times M$. This requires $K \times K \times M + M \times N$ parameters and performs $K \times K \times M \times W \times H + M \times N \times W \times H$ multiply-accumulate operations. The number of parameters and computing operations is reduced to $\frac{1}{N} + \frac{1}{K^2}$ of the conventional convolution, resulting in a substantial reduction in both the parameter count and computational load. Moreover, the reduction is more significant for convolution layers with a higher number of channels.

2.2. Squeeze and Excitation

Squeeze and excitation, proposed in [43], introduces a lightweight attention mechanism into CNNs. It enhances the feature extraction ability for certain features by adjusting the weights of different channels while suppressing irrelevant features, thus improving the network's performance. The SE block can directly replace network modules or be inserted into existing networks. It only requires a small number of operations to enhance network performance with a minimal increase in parameters and computational load compared to the original network. As shown in Figure 1, the SE structure consists of two parts: 'squeeze' and 'excitation'. The 'squeeze' structure compresses the feature map through global pooling, while the 'excitation' structure extracts global features using two fully connected layers and a nonlinear function to form channel weights, which are then applied to the input feature map to implement attention on different features.

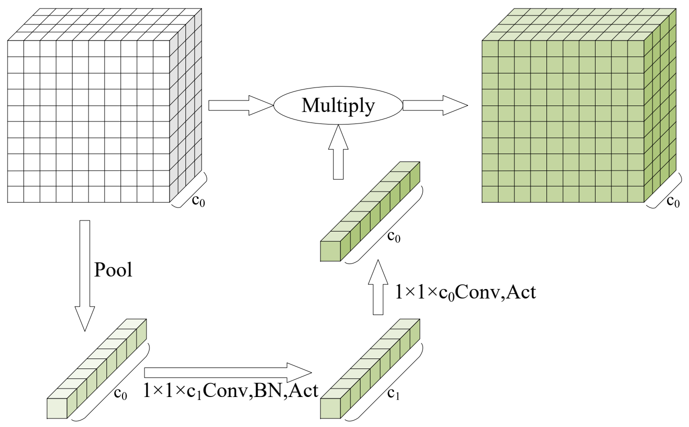


Figure 1. Structure of SE module.

2.3. MobileNet

MobileNet [44–46] is a convolutional neural network designed for mobile platforms. To adapt to the limited computational and storage resources of mobile devices, it replaces conventional convolutions with depthwise separable convolutions and significantly reduces the model's parameters and computational load with only a small performance loss. In MobileNet V3, linear bottlenecks and inverted residual blocks are introduced to further improve the efficiency and performance of the network. The network uses inverted residual blocks as the basic building units, which first expand the input feature map's channels, then extract features via lightweight depthwise convolutions, and finally project the features back to a low-dimensional representation, significantly reducing memory usage during the inference process. Additionally, skip connections are placed in locations with fewer channels, further reducing the computational load.

3. The Proposed Computing Architecture

In this paper, we propose an FPGA-optimized depthwise separable convolution network accelerator, whose functional block diagram is shown in Figure 2. The accelerator consists of five convolution modules, one pooling module, one multiplication module, and two caching modules. Among the convolution modules, there is one depthwise convolution module and four scalable pointwise convolution modules.

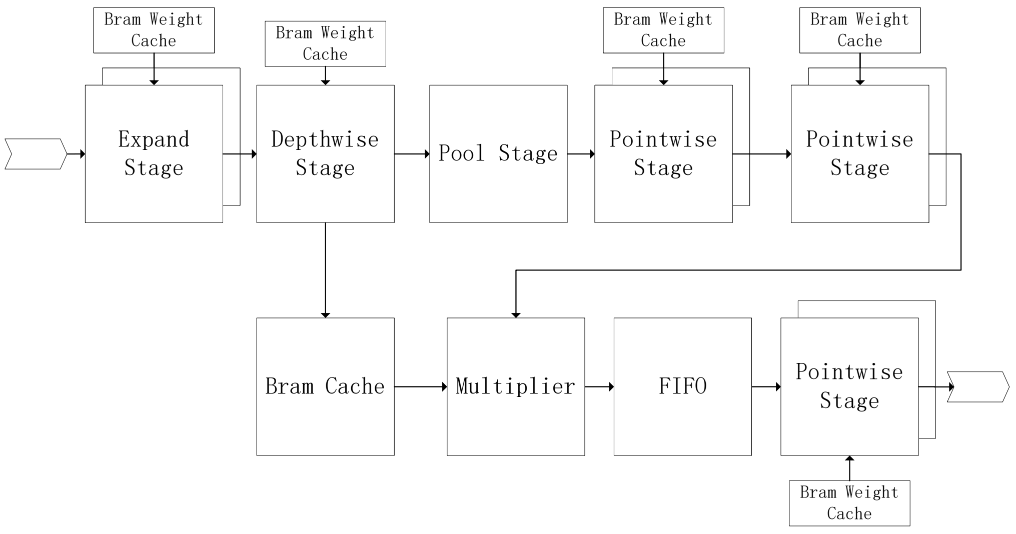


Figure 2. Block diagram of the proposed accelerator.

3.1. Module Design

3.1.1. Overall Design

As shown in Figure 2, the accelerator consists of multiple modules connected in series. During operation, a large amount of temporary data are generated. To ensure that the data produced at each stage are not misread or missed, data buffering or synchronization mechanisms are required between consecutive modules.

To reduce the complexity of the control mechanism, this paper adopts a simple synchronization approach using a VALID signal from front to back. This synchronization method avoids increasing the complexity of the upstream module by being synchronized through the downstream module, which could lead to data congestion, increased latency, and reduced performance. Between adjacent modules, synchronization is achieved through the OVALID port of the upstream module and the IVALID port of the following module. When the upstream module outputs valid data, it raises OVALID. The following module detects that IVALID has been raised, considers the data valid, and reads it in. This synchronization method avoids increasing module complexity by synchronizing the upstream module through the downstream module, which could lead to data congestion and increased latency, thus reducing performance.

To reduce resource consumption and improve computation speed, auxiliary modules such as buffers in the accelerator should be minimized without affecting the computation. Considering that most modules require multiple computations from the upstream module to generate enough data to start their operations, for most of the time, the speed at which the downstream module consumes data is much higher than the speed at which the upstream module generates data. Therefore, by synchronizing the timing of the data between adjacent modules, most modules can be connected directly without adding data buffers.

Each convolution module has an input channel count port and an output channel count port for flexible configuration, which are driven by its internal finite-state machine (FSM).

The batch normalization and activation layers in the accelerator are relatively simple and fixed, and the batch normalization layer can be fused with adjacent convolution layers to eliminate that layer. Therefore, these are omitted in the following discussion.

3.1.2. Pointwise Convolution Module

Pointwise convolution performs convolution on the input feature map of size $C \times W \times H$ with convolution kernels of size $N \times C \times 1 \times 1$. In this paper, the feature map is stored in a depth-first manner, so the size of the computation window is $C \times 1 \times 1$. The data of the input feature map within the window are read sequentially into the module, multiplied by the corresponding weights, and then sent to the accumulator. After accumulating C data points, the output feature map data are generated.

To enable continuous computation, the convolution weights are managed using a ping-pong buffering mechanism. Two simple dual-port BRAM modules, referred to as *BRAM A* and *BRAM B*, are used. On completion of reading from BRAM A, a selection signal switches the data source to BRAM B, and BRAM A is rewritten with the weights required for the next computation cycle by an AXI BRAM Controller. The two modules alternate between reading and writing.

After completing one convolution, the multiplication window of the module needs to move to the next position. To achieve the smoothness of the overall computation process, the pointwise convolution is divided into two types, each employing different window shifting strategies to handle data at various stages more effectively.

The pointwise convolution module at the beginning of the accelerator differs from the other pointwise convolution modules and is referred to as the *expand module*. As the first stage of the pipeline, the expand module reads data from AXI DMA ip core. To match its output data flow with the downstream depthwise module, the multiplication window is moved over the input feature map in the order of input channels–width–height–output channels. The computation flow of this module is illustrated in Algorithm 1 and Figure 3, where $a_{i,j,k}$ and $b_{n,j,k}$ represent the data at position (j, k) in the i-th input feature map channel and the n-th output feature map channel, respectively, and $w_{n,i}$ denotes the weight of the i-th channel in the n-th convolution kernel.

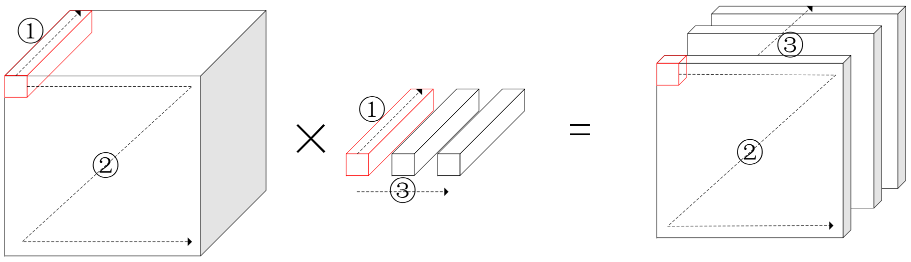


Figure 3. Block diagram of the expand module.

Algorithm 1 Pointwise Convolution in Expand Module

```
1:  procedure POINTWISECONVOLUTION($a, w$)
2:      for $n \leftarrow 0$ to $C_{out} - 1$ do
3:          for $k \leftarrow 0$ to $H - 1$ do
4:              for $j \leftarrow 0$ to $W - 1$ do
5:                  $b_{n,j,k} \leftarrow 0$
6:                  for $i \leftarrow 0$ to $C_{in} - 1$ do
7:                      $b_{n,j,k} \leftarrow b_{n,j,k} + a_{i,j,k} w_{n,i}$
8:                  end for
9:                  Output $b_{n,j,k}$
10:             end for
11:         end for
12:     end for
13: end procedure
```

In the downstream pointwise convolution modules, hereafter referred to as the *regular pointwise convolution modules*, the multiplication window is moved in the order of input channels–output channels–width–height. Unlike the first convolution module, these modules do not have the flexibility to freely control the timing of data reads. Therefore, they need to match the output timing of the upstream module while also ensuring the data are accessible and suitable for computation by the following module. The computation process is shown in Algorithm 2 and Figure 4, where $a_{i,j,k}$ and $b_{n,j,k}$ represent the data at position (j, k) in the i-th input feature map channel and the n-th output feature map channel, respectively, and $w_{n,i}$ denotes the weight of the i-th channel in the n-th convolution kernel.

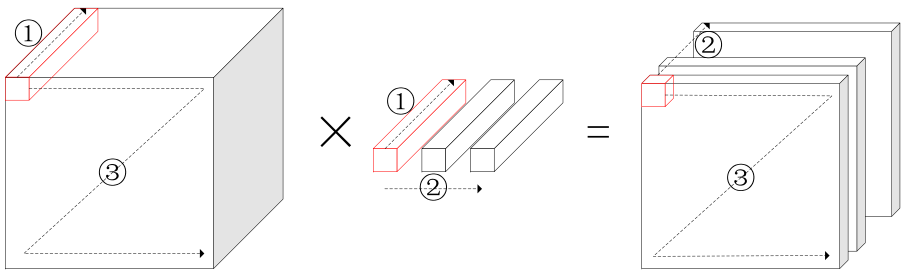


Figure 4. Block diagram of pointwise convolution module.

Algorithm 2 Regular Pointwise Convolution

```
1:  procedure POINTWISE CONVOLUTION($a, w$)
2:      for $k \leftarrow 0$ to $H - 1$ do
3:          for $j \leftarrow 0$ to $W - 1$ do
4:              for $n \leftarrow 0$ to $C_{out} - 1$ do
5:                  $b_{n,j,k} \leftarrow 0$
6:                  for $i \leftarrow 0$ to $C_{in} - 1$ do
7:                      $b_{n,j,k} \leftarrow b_{n,j,k} + a_{i,j,k} w_{n,i}$
8:                  end for
9:                  Output $b_{n,j,k}$
10:             end for
11:         end for
12:     end for
13: end procedure
```

3.1.3. Depthwise Convolution Module

The depthwise convolution module is the next level after the expand module. It operates by performing 2D convolution for each input feature map channel with its corresponding convolution kernel without the need to sum the results of multiple channels. After the expand module, the data flow transforms into a width–height–depth format, allowing computation for each channel individually. Given that this module involves multiply-accumulate operations, employing DSP-based multiplier-accumulators (MACs) facilitates better control over the computation timing. Considering lower-end FPGAs have very limited DSP resources, it is necessary to control the usage of DSPs. Since only one data point is input per valid clock cycle in this module, only one channel is actively computed at any given time, and a single 2D convolution module is sufficient to meet the computational requirements.

The 2D convolution adopts a sliding window calculation with the structure shown in Figure 5. It consists of a multiplier-accumulator (MAC) array, shift registers (SRs), and a flow controller. The MAC array and SR together form a pipeline that can be accessed independently at each level. At the input of the multipliers, all MACs in the array use the same input feature map data $a_{i,j}$. The MAC at position (m, n) in the array uses $w_{n \times k+m}$, so each computation cycle combines the input feature map data from $a_{i,j,k}w_0$ to $a_{i,j,k}w_{k*k-1}$ with the $k \times k$ weight data. The MAC outputs in each row, except for the last one, are connected to the next MAC's summation input. The output of the last MAC in the row is connected to the SR, and the SR's output is connected to the first MAC of the next row. The final output of the MAC at $(k-1, k-1)$ position is the output of this module. By combining different product terms, the 2D convolution result can be obtained.

Similar to regular convolution, the depthwise convolution also requires padding and stride. Padding is necessary to ensure that the width and height of the output feature map do not shrink after convolution. In an unoptimized calculation flow, padding zeros must be added to the beginning of the input feature map data, as well as to the beginning of each row, before the data from the upstream module can be read in. This can severely affect the computation timing. Since this module is located at the second level of the accelerator, the input data rate is relatively high. Using the unoptimized flow could reduce computation efficiency and strictly limit the time window for reading input data. Thus, modifications to the upstream module are required to synchronize its output timing with the depthwise convolution module, or a cache must be added.

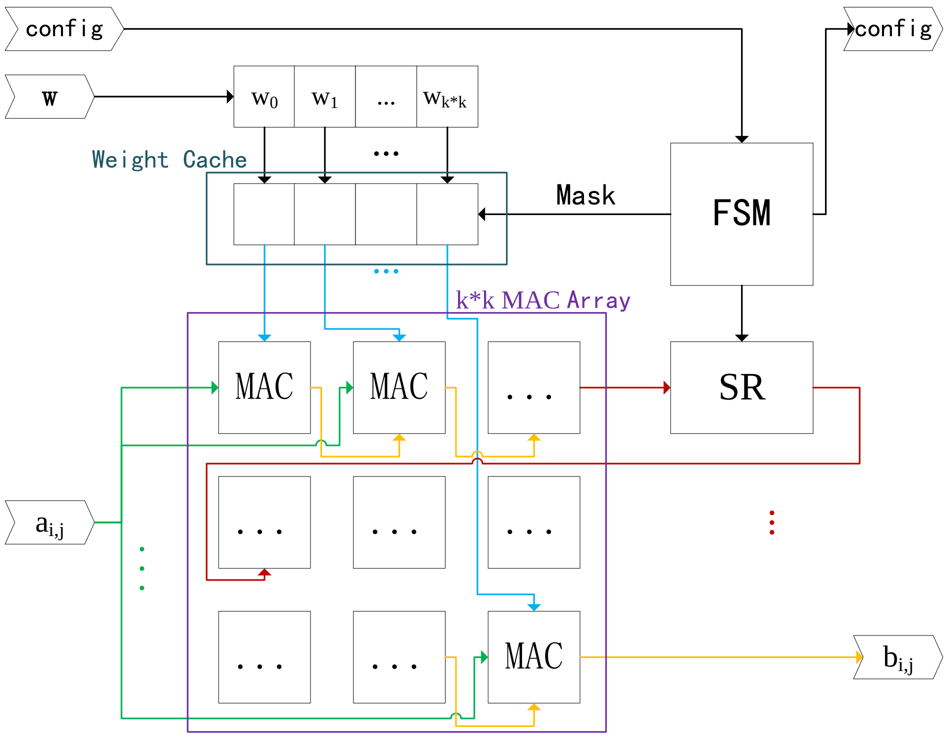


Figure 5. Block diagram of depthwise convolution module.

For sliding window convolutions, when switching rows and reading the beginning or end of each row, invalid combinations can occur. Padding can be easily implemented by forcing a specific MAC unit in the MAC array to output zero at specific moments, as shown in Figure 6. Only the last row of the input feature map requires actual zero-padding. For other padding locations, there is no need to actually zero out the input data, saving extra time and allowing continuous data input. Forcing the MAC unit to output zero can be achieved by zeroing out the input feature map data or the weights, while simply using the synchronous clear port will affect the internal computation pipeline of the MAC units. In this module, a mask is applied to the weights to ensure that the MACs output the result at the same time as other non-pad MAC units.

The stride operation makes sure the width and height of the input feature map halve at specific depths in the network. To implement stride, a parity counter is added to the module, which controls the VALID signal so that when the output is at even columns or even rows, it is marked as invalid without affecting the module's input and output timing.

The computation process of the depthwise convolution module is shown in Algorithm 3 and Figure 7. Here, $a_{i,j,k}$ represents the data at the (j,k) position of the i-th channel in the input feature map, $\mathbf{w}_i$ represents the vector of weights for the i-th channel's convolution kernel, $\mathbf{a}[i,j,k]$ represents the matrix of all input feature map data in the MAC array when $a_{i,j,k}$ is read, and $\mathbf{M}_{m,n}$ is the mask matrix used to extract valid terms from the MAC array when $a_{i,j,k}$ is read. Therefore, the calculation formula of depthwise convolution is shown in Equation (1). For example, when the module is in the state shown in Figure 6a, the following computations will result in the convolution after padding, as shown in Figure 6c. Note that the algorithm only demonstrates the calculation of the raw output data, and the final validity of the output data needs to be judged together with the VALID signal, which can be acquired using an input data counter.

$$b_{i,j,k} = \sum_{m=0}^{4} \sum_{n=0}^{4} a_{i,j+m-2,k+n-2} w_{m+1,n+1} \mathbf{M}_{j,k}[m,n] \tag{1}$$

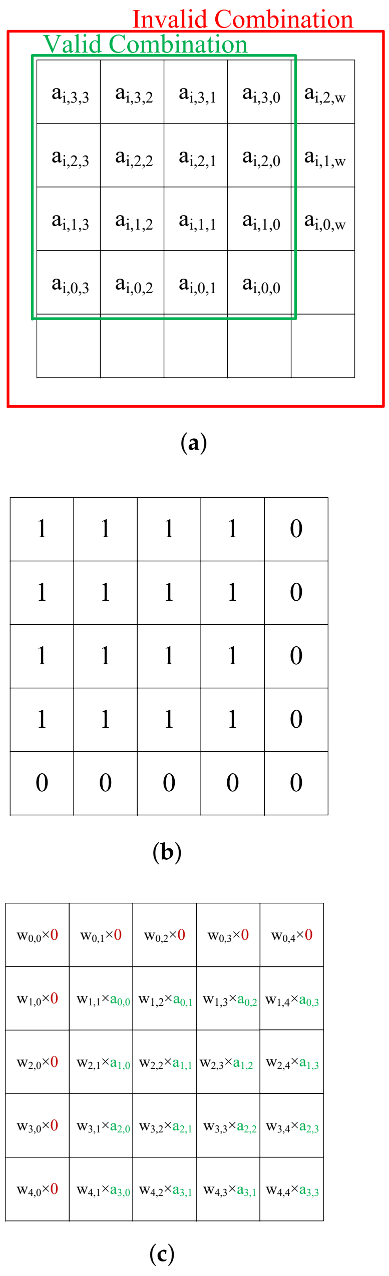

Invalid Combination
Valid Combination
a_{i,3,3} a_{i,3,2} a_{i,3,1} a_{i,3,0} a_{i,2,w}
a_{i,2,3} a_{i,2,2} a_{i,2,1} a_{i,2,0} a_{i,1,w}
a_{i,1,3} a_{i,1,2} a_{i,1,1} a_{i,1,0} a_{i,0,w}
a_{i,0,3} a_{i,0,2} a_{i,0,1} a_{i,0,0}
(a)
1 1 1 1 0
1 1 1 1 0
1 1 1 1 0
1 1 1 1 0
0 0 0 0 0
(b)
w_{0,0}×0 w_{0,1}×0 w_{0,2}×0 w_{0,3}×0 w_{0,4}×0
w_{1,0}×0 w_{1,1}×a_{0,0} w_{1,2}×a_{0,1} w_{1,3}×a_{0,2} w_{1,4}×a_{0,3}
w_{2,0}×0 w_{2,1}×a_{1,0} w_{2,2}×a_{1,1} w_{2,3}×a_{1,2} w_{2,4}×a_{1,3}
w_{3,0}×0 w_{3,1}×a_{2,0} w_{3,2}×a_{2,1} w_{3,3}×a_{2,2} w_{3,4}×a_{2,3}
w_{4,0}×0 w_{4,1}×a_{3,0} w_{4,2}×a_{3,1} w_{4,3}×a_{3,1} w_{4,4}×a_{3,3}
(c)

Figure 6. (**a**) Output of the MAC array; (**b**) mask required to obtain valid result; (**c**) convolution with padding.

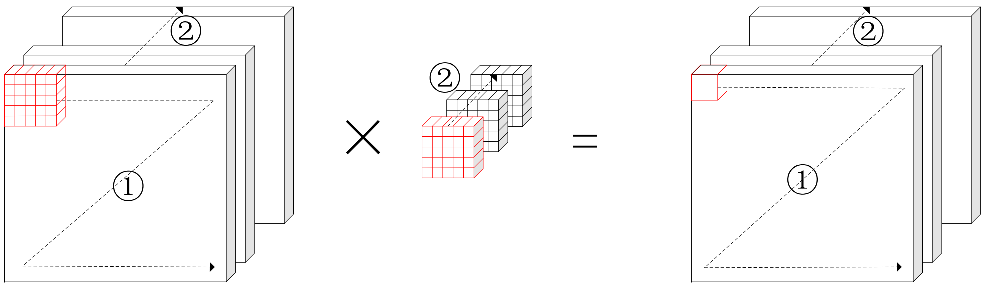


Figure 7. Block diagram of depthwise convolution module.

Algorithm 3 Depthwise Convolution

```
1: procedure DEPTHWISE CONVOLUTION($(a, w)$)
2:     $pad\_size \leftarrow \lfloor \frac{kernel\_size}{2} \rfloor$
3:     for $i \leftarrow 0$ to $C-1$ do
4:         for $k \leftarrow 0$ to $H-1$ do
5:             for $j \leftarrow 0$ to $W-1$ do
6:                 if $k \leqslant h + pad\_size$ then
7:                     Read $a_{i,j,k}$
8:                 else
9:                     $a_{i,j,k} \leftarrow 0$
10:                end if
11:                $\mathbf{m} \leftarrow \mathbf{M}[j,k]$
12:                Output $\sum_{m=0}^{kernel\_size} \sum_{n=0}^{kernel\_size} a_{i,m,n} w_{m+1,n+1} \mathbf{M}_{j,k}[m,n]$
13:            end for
14:        end for
15:    end for
16: end procedure
```

3.1.4. Squeeze-and-Excite Module

The SE module consists of pooling, pointwise convolution, and buffers. The pooling module is the first stage of the SE module, which is the next stage after the depthwise convolutional module. It performs average pooling on each channel of the input feature map, as shown in Equation (2). To reduce precision loss, this module first performs summation followed by division (multiplication). Since the output data flow of the upstream stage is still in the format of width–height–depth, no additional buffer is needed for this module. The module can directly accumulate the input data while simultaneously counting. After accumulating $W \times H$ data points, division is performed once, thus reducing the number of division operations.

$$b_i = \frac{1}{WH} \sum_{j=0}^{W-1} \sum_{k=0}^{H-1} a_{i,j,k} \tag{2}$$

After the pooling module, two pointwise convolutions are used to implement the nonlinear transformations. Since the amount of data for the nonlinear transformation is relatively small, register buffers are used for direct storage until they are needed later. The squeeze-and-excite module uses one BRAM buffer and one FIFO buffer. The BRAM buffer is located at the head of the SE module, which is immediately after the depthwise convolutional module. It receives the output from the depthwise convolutional module simultaneously with the Pool Module and stores it.

Since the SE module requires repeated reading of the upstream module's output data at different phases, and unlike the expand module, this module is situated in the middle of the accelerator, it would take considerable extra time to recalculate these values starting from the expand module. To ensure that data are read and processed in real time, the second round of computation must start in advance, which increases the scheduling complexity. Using the BRAM buffer simplifies the scheduling and allows the data flow format to be adjusted, outputting the cached data in a depth–width–height format, which is convenient for downstream computations.

The last step of the SE module's computation involves multiplying the input feature map data stored in BRAM with the results of the pointwise convolutions. The output timing of this operation differs from other convolution modules' input timings. Since the final stage of the accelerator is a pointwise convolutional module, and like other pointwise convolutional modules in the accelerator, this module also requires two stages for data reading and computation, the output rate of the squeeze-and-excite module must be synchronized with the input rate of the downstream pointwise convolutional module. Therefore, a FIFO buffer is placed at the end of the SE module, and the control is achieved using the FIFO's empty and full signals.

3.2. Parallelism

Both the expand module and the pointwise convolution module in this work can be implemented with parallel computation by stacking multipliers. During computation, for each input feature map channel, it is divided into $m = \frac{C}{n}$ parts, where n is the number of parallel multipliers. The module then reads n channels of feature map data and corresponding weights at a time and performs n-way multiplication. The results are sent to an n-input adder tree and subsequently to an accumulator. The m results are accumulated to obtain one output data.

The time required for the initial reading and caching of one channel's data depends solely on the upstream module, and parallel processing offers no benefits. In the downstream calculations of the feature map channel, without considering the delay of the multipliers themselves, the time required for a non-parallel computation is C_{in} clock cycles, while for parallel computation, it is $\frac{C_{in}}{n} + \lceil \log_2 n \rceil + t_{Acc}$ clock cycles, where t_{Acc} is the time used in the accumulation process. The time saved is $C_{in} - \frac{C_{in}}{n} - \lceil \log_2 n \rceil - t_{Acc}$ clock cycles. The additional resource consumption is $m - 1$ multipliers and one n-input adder tree. It is clear that as n increases, the saved time reaches its maximum when $n = C_{in} \ln 2$, but the resource consumption continues to grow. To fully utilize the adder tree, n should preferably be a power of 2.

In the accelerator, there are four modules that can be parallelized: the expand module, pointwise convolution #1 of the SE module, pointwise convolution #2 of the SE module, and the pointwise convolution module. The overall parallelism of the accelerator is represented by $N_{EXC} - N_{SE1} - N_{SE2} - N_{PWC}$. To enable convolution modules to read multiple input data at a time, for the expand module, the input bit-width is increased, while for the other pointwise convolution modules, the input feature map data are cached in registers after the first read, allowing multiple data to be read directly from the cache at a time.

4. Experimental Results and Analysis

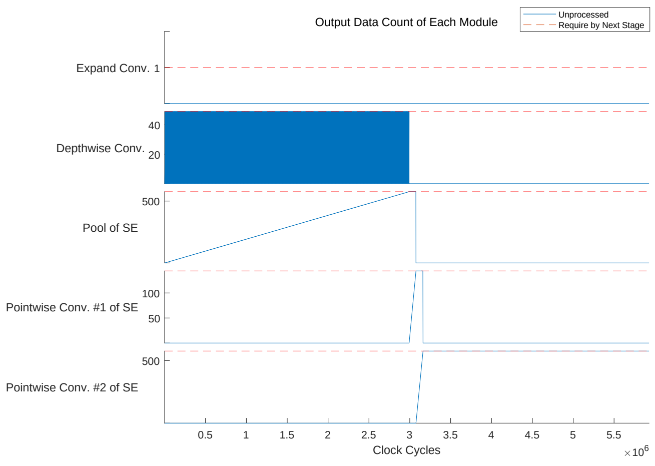
4.1. Simulation Result

Since the accelerator does not have a feedback mechanism, once a module finishes its computation, it immediately outputs the result to the next stage. Therefore, it is essential to ensure that every stage's output data are received by the next stage in time so that the data can be fully processed.

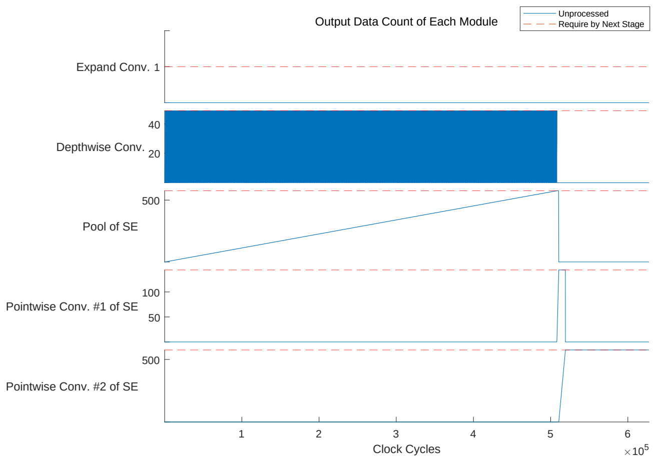
The experiment was conducted using the last bottleneck module from MobileNetV3-Small with the module parameters shown in Table 1. A pipeline system simulation was performed for the overall accelerator. Each time a module finishes a computation, the output count is incremented. When the output data of the current stage are enough for the next stage to perform a computation, the output count is decremented by the required data amount. This helps to track whether there is any data backlog in each module. The results are shown in Figure 8, where Figure 8a shows the simulation result for non-parallel computation, and Figure 8b shows the simulation result for a parallelism configuration of 48-288-72-288. As can be seen, even when the parallelism is increased to unreasonable values, the temporary data generated by each module are still processed in time without any data congestion.

Table 1. Parameters of module.

Input Size	Input Channel	Expand Channel	Squeeze Channel	Output Size	Output Channel
7 × 7	96	576	144	7 × 7	96



(**a**)



(**b**)

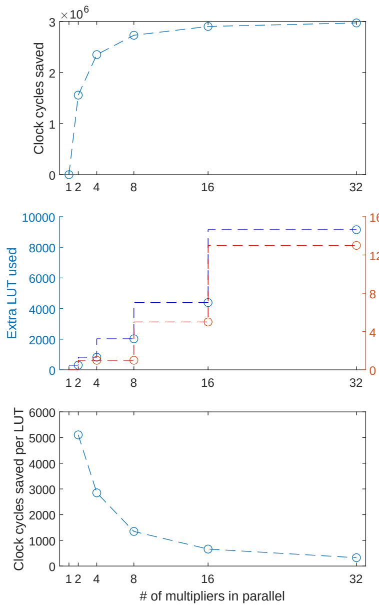
Figure 8. Simulation result: (**a**) no parallel (1-1-1-1); (**b**) extreme parallel (48-288-72-288).

4.2. Experimental Results and Analysis

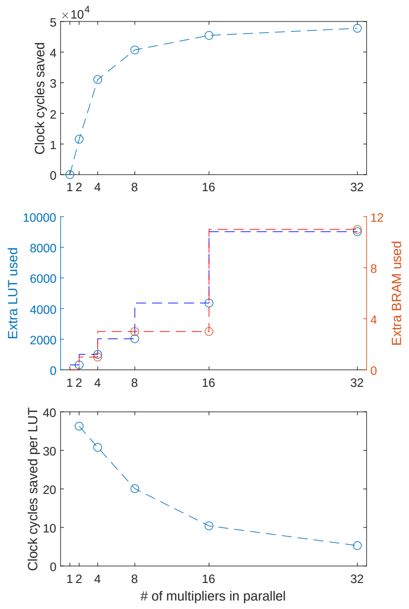
The modules shown in Table 1 were tested on both the Xilinx XC7Z020 FPGA and an ARM CPU. The feature map data and weight data were represented using 16-bit fixed-point numbers. When testing on the FPGA, except for the MACs in the depthwise convolution module, which used DSP-based MACs, other modules used LUT-based multipliers and accumulators.

4.2.1. Improvements on Modules

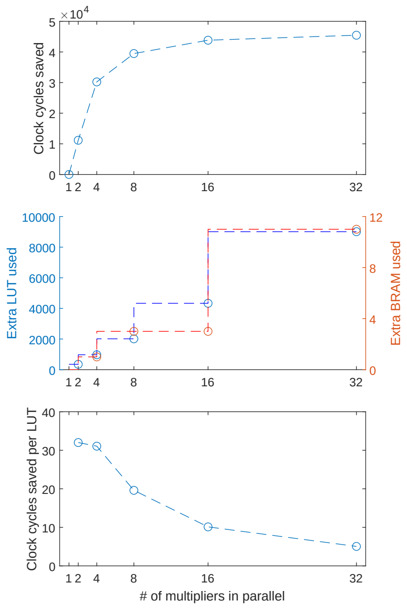
This section presents the experimental results on the impact of stacking 1, 2, 4, 8, 16, and 32 multipliers in the four parallelizable modules of the accelerator. The experiments examine the effect of these configurations on the overall computation speed and resource consumption. To clearly showcase the performance and resource consumption of different module configurations, Figure 9 compares the reduction in computation time and the increase in resource consumption relative to the non-parallel case.



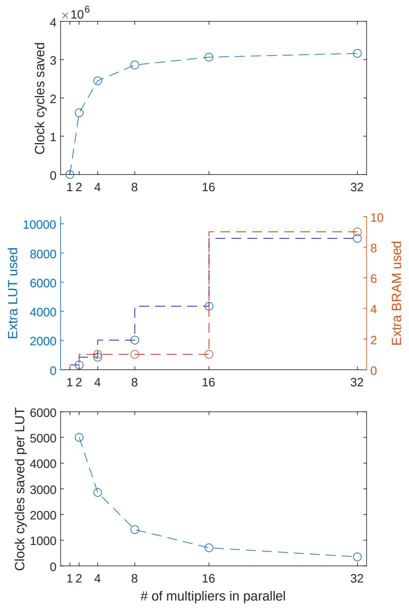
(**a**)



(**b**)



(**c**)



(**d**)

Figure 9. Impacts of different stack configurations: (**a**) expand module; (**b**) the 1st convolution of the SE module; (**c**) the 2nd convolution of the SE module; (**d**) pointwise module.

Analysis of the experimental outcomes reveals that as the parallelism of each module increases, the gains in computation time decrease rapidly. Since the parallelism is set to powers of two, increasing the parallelism causes the LUT resource consumption to double. As parallelism increases, the BRAM used to cache the weight data also needs to read multiple data points at once, which increases the data bit-width accordingly. Additionally, the available capacity of BRAM is also affected by the bit width, resulting in a sharp increase in usage when a certain level of parallelism is reached. Figure 10 illustrates the trend of computation speed improvement as different parallelism levels are set for the expand module and pointwise convolution module. The SE module mentioned in the figure has a much smaller computational load because its input feature map has a width and height of 1×1, which results in faster computation. However, the gains in computation time from increasing parallelism are much smaller compared to the expand module and pointwise module. Nevertheless, the LUT and BRAM resource consumption in the SE module is comparable to that of the expand module and pointwise convolution module. The impact on performance is much smaller in the SE module, which is why the parallelism of the two pointwise convolutions in the SE module is set to 1. From the figure, it can be observed that as parallelism increases from 16 to 32, the computation time savings become minimal, while BRAM consumption increases by 160%, 267%, 267%, and 800%, respectively. This marginal effect is better reflected in the time savings per LUT.

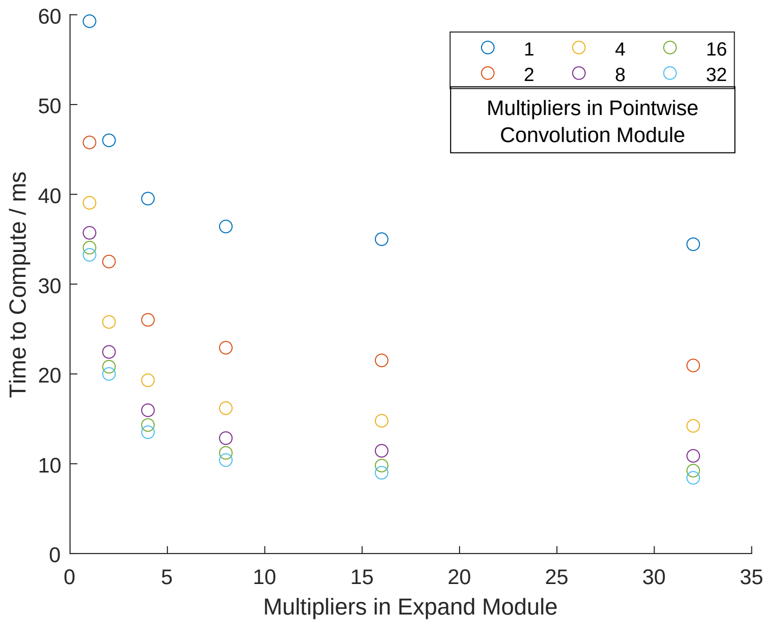


Figure 10. Performance of different stack configurations.

4.2.2. Comparison Against Other Implementations

This section compares the performance of the FPGA accelerator and the ARM Cortex A9 CPU in the same chip as well as other solutions from the existing literature. The CPU clock frequency of the ZYNQ 7020 is set to 666.67 MHz, and the FPGA clock frequency is set to 100 MHz. The same bottleneck module as shown in Table 1 is used for comparison. Figure 11 shows the time required to perform the same computation on the MobileNet V3-small using the CPU and FPGA implementations. The labels on the left indicate the number of input channels, the number of channels after the expand module, the number of output channels, and the width and height of the input feature map. Although there are several optimization techniques available on the ARM CPU, such as utilizing the NE10 library to exploit the ARM NEON instruction set for SIMD (single instruction multiple data) operations, the FPGA accelerator, even without additional stacked multipliers, is able to outperform the pure CPU solution. Among the listed computation modules, the speedup can range from 1.47× to 54×.

To implement the complete network using the modules in this paper, one can either reuse a module by modifying its size configuration for different depths in the network, use two modules in a ping-pong fashion, or directly chain multiple modules together. Table 2 compares the solution in this paper with other approaches from the existing literature. The solution in this paper uses two accelerator modules with configurations of (16-1-1-16) for performing depthwise convolutions with 3×3 and 5×5 convolution kernels. Compared to other solutions, the FPGA hardware resource consumption is reduced across the board, particularly the DSP usage, which is significantly reduced. When compared to other FPGA solutions using DSPs, our approach reduces the DSP usage by more than 90%. Due to the relatively high parallelism setting of 16, the use of LUT resources is higher. However, by sacrificing performance and adopting a lower parallelism level, it is possible to reduce the usage of LUT resources.

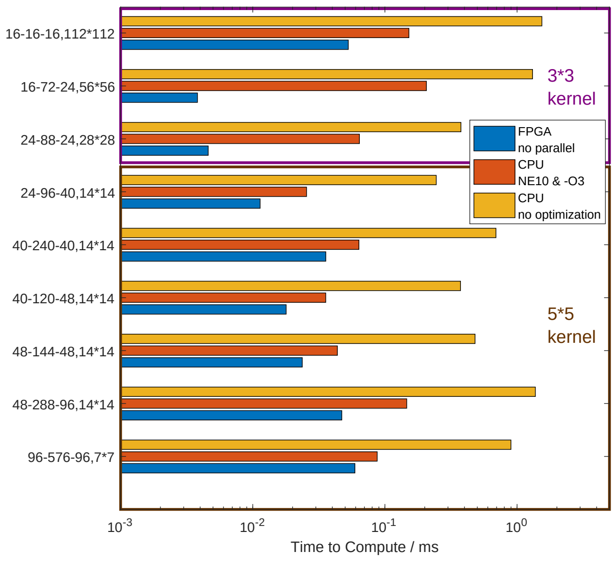


Figure 11. Performance comparison between FPGA and ARM CPU.

Table 2. Resource utilization comparison.

	This Work (16-1-1-16)	[41]	[47]
Network	MobileNetV3-Small	MobileNetV2	MobileNetV1-Lite
FPGA Model	Xilinx XC7Z020	Xilinx VC709	Xilinx XCKU040
Quantization	16b fixed	float	16b fixed
Logic	**49,074**/53,200	107,325/433,200	59,862/242,400
DSP	**34**/220	**0**/3600	603/1920
BRAM	**124**/140	13.7 Mb/51.7 Mb	233/1200
	[48]	[49]	[50]
Network	VGG-16	VGG-16	VGG-16
FPGA Model	Xilinx ZCU102	Xilinx ZCU102	Xilinx XC7Z045
Quantization	8b fixed	8b fixed	16b fixed
Logic	324 K (64%)	-	182,616/218,600
DSP	528 (21%)	1024/2520	780/900
BRAM	1108 (60%)	-	486/545

5. Conclusions

This paper introduces a scalable and efficient CNN accelerator designed for low-end FPGAs, which is optimized specifically for depthwise separable convolutions and the SE module. The proposed accelerator significantly enhances computational efficiency and resource utilization, making it suitable for deployment on hardware platforms with limited resources. By capitalizing on the characteristics of depthwise separable convolutions and optimizing the computational flow, the accelerator achieves a flexible balance between hardware resource consumption and computation speed. Experimental results demonstrate that the proposed approach outperforms ARM CPUs and other FPGA-based solutions in terms of both speed and resource utilization. It offers at least a 1.47× performance improvement over ARM Cortex A9 CPUs and reduces DSP usage by over 90% compared to existing FPGA solutions. This work focuses on the development of a low-cost FPGA-based CNN accelerator that enhances depthwise separable convolutions and the squeeze-and-excite module, providing a flexible and configurable computational architecture. It includes an in-depth analysis of how parallel computing impacts resource usage and processing speed, while proposing strategies to adjust parallelism to meet specific requirements, thereby balancing computational efficiency with resource consumption. Moreover, the optimization of data flow and caching mechanisms reduces the need for intermediate data storage, further improving computational efficiency. The synchronization mechanism ensures accurate data transfer between modules, simplifying the control process. In summary, this paper addresses the challenges of deploying CNNs on low-end FPGAs by proposing an efficient accelerator architecture that minimizes resource consumption while maintaining high performance. This approach facilitates the deployment of the module on a wider range of low-end, resource-constrained FPGAs, enabling the full utilization of FPGA parallelism while reducing hardware costs for the edge deployments of convolutional neural networks. Consequently, it allows for the deployment of the module on more resource-limited FPGAs or the development of more complex systems. Future work may focus on further optimizing parallelism strategies and exploring additional lightweight neural network architectures to expand the applicability of the accelerator on even more constrained platforms.

Author Contributions: J.S. is mainly responsible for the supervision and leadership of the planning and implementation of scientific research activities. X.Y. and L.Z. are mainly responsible for the research design and code writing and article writing. Y.L. is mainly responsible for the literature search and format proofreading. X.C. and W.C. are mainly responsible for LaTex typesetting and drawing. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2023A1515011717, Special Projects for Key Fields of the Education Department of Guangdong Province under Grant 2024ZDZX1048.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets used and analyzed during the current study are available from the corresponding author upon reasonable request. All data generated or analyzed during this study are included in this article.

Conflicts of Interest: The authors declare no competing interests.

References

1. Wang, C.; Zhang, Q.; Wang, X.; Zhou, L.; Li, Q.; Xia, Z.; Ma, B.; Shi, Y.Q. Light-Field Image Multiple Reversible Robust Watermarking Against Geometric Attacks. *IEEE Trans. Dependable Secur. Comput.* **2015**, *189*, 106896. [CrossRef]
2. Liu, Y.; Wang, C.; Lu, M.; Yang, J.; Gui, J.; Zhang, S. From Simple to Complex Scenes: Learning Robust Feature Representations for Accurate Human Parsing. *IEEE Trans. Pattern Anal. Mach. Intell.* **2024**, *46*, 5449–5462. [CrossRef]
3. Ma, B.; Li, K.; Xu, J.; Wang, C.; Li, X. A High-Performance Image Steganography Scheme Based on Dual-Adversarial Networks. *IEEE Signal Process. Lett.* **2024**, *31*, 2655–2659. [CrossRef]
4. Wan, Y.; Xie, X.; Chen, J.; Xie, K.; Yi, D.; Lu, Y.; Gai, K. ADS-CNN: Adaptive Dataflow Scheduling for lightweight CNN accelerator on FPGAs. *Future Gener. Comput. Syst.* **2024**, *158*, 138–149. [CrossRef]
5. Wan, Y.; Xie, X.; Yi, L.; Jiang, B.; Chen, J.; Jiang, Y. Pflow: An end-to-end heterogeneous acceleration framework for CNN inference on FPGAs. *J. Syst. Archit.* **2024**, *150*, 103113. [CrossRef]
6. Nakamura, T.; Saito, S.; Fujimoto, K.; Kaneko, M.; Shiraga, A. Spatial- and time-division multiplexing in CNN accelerator. *Parallel Comput.* **2022**, *111*, 102922. [CrossRef]
7. Hu, X.; Yu, S.; Zheng, J.; Fang, Z.; Zhao, Z.; Qu, X. A hybrid CNN-LSTM model for involuntary fall detection using wrist-worn sensors. *Adv. Eng. Inform.* **2025**, *65*, 103178. [CrossRef]

8. Xu, Z.; Zhang, B.; Luo Fan, L.; Hengzhou Yan, E.; Li, D.; Zhao, Z.; Sze Yip, W.; To, S. Deep-learning-driven intelligent tool wear identification of high-precision machining with multi-scale CNN-BiLSTM-GCN. *Adv. Eng. Inform.* **2025**, *65*, 103234. [CrossRef]
9. Nguyen Thanh, P.; Cho, M.Y. Advanced AIoT for failure classification of industrial diesel generators based hybrid deep learning CNN-BiLSTM algorithm. *Adv. Eng. Inform.* **2024**, *62*, 102644. [CrossRef]
10. Liu, T.; Zheng, H.; Zheng, P.; Bao, J.; Wang, J.; Liu, X.; Yang, C. An expert knowledge-empowered CNN approach for welding radiographic image recognition. *Adv. Eng. Inform.* **2023**, *56*, 101963. [CrossRef]
11. Wu, H.; Li, H.; Chi, H.L.; Peng, Z.; Chang, S.; Wu, Y. Thermal image-based hand gesture recognition for worker-robot collaboration in the construction industry: A feasible study. *Adv. Eng. Inform.* **2023**, *56*, 101939. [CrossRef]
12. Liu, C.; Jiang, Q.; Peng, D.; Kong, Y.; Zhang, J.; Xiong, L.; Duan, J.; Sun, C.; Jin, L. QT-TextSR: Enhancing scene text image super-resolution via efficient interaction with text recognition using a Query-aware Transformer. *Neurocomputing* **2025**, *620*, 129241. [CrossRef]
13. Yang, Z.; Tian, Y.; Wang, L.; Zhang, J. Enhancing Generalization in Camera Trap Image Recognition: Fine-Tuning Visual Language Models. *Neurocomputing* **2025**, *634*, 129826. [CrossRef]
14. Fu, C.; Zhou, T.; Guo, T.; Zhu, Q.; Luo, F.; Du, B. CNN-Transformer and Channel-Spatial Attention based network for hyperspectral image classification with few samples. *Neural Netw.* **2025**, *186*, 107283. [CrossRef]
15. Wei, H.; Yi, D.; Hu, S.; Zhu, G.; Ding, Y.; Pang, M. Multi-granularity classification of upper gastrointestinal endoscopic images. *Neurocomputing* **2025**, *626*, 129564. [CrossRef]
16. Mou, K.; Gao, S.; Deveci, M.; Kadry, S. Hyperspectral Image Classification Based on Dual Linear Latent Space Constrained Generative Adversarial Networks. *Appl. Soft Comput.* **2025**, *174*, 112962. [CrossRef]
17. Shu, X.; Li, Z.; Tian, C.; Chang, X.; Yuan, D. An active learning model based on image similarity for skin lesion segmentation. *Neurocomputing* **2025**, *630*, 129690. [CrossRef]
18. Xu, Z.; Wang, H.; Yang, R.; Yang, Y.; Liu, W.; Lukasiewicz, T. Aggregated Mutual Learning between CNN and Transformer for semi-supervised medical image segmentation. *Knowl. Based Syst.* **2025**, *311*, 113005. [CrossRef]
19. Li, K.; Yuan, F.; Wang, C. An effective multi-scale interactive fusion network with hybrid Transformer and CNN for smoke image segmentation. *Pattern Recognit.* **2024**, *159*, 111177. [CrossRef]
20. Ding, W.; Huang, Z.; Huang, Z.; Tian, L.; Wang, H.; Feng, S. Designing efficient accelerator of depthwise separable convolutional neural network on FPGA. *J. Syst. Archit.* **2019**, *97*, 278–286. [CrossRef]
21. Yan, Y.; Ling, Y.; Huang, K.; Chen, G. An efficient real-time accelerator for high-accuracy DNN-based optical flow estimation in FPGA. *J. Syst. Archit.* **2022**, *136*, 102818. [CrossRef]
22. Karamimanesh, M.; Abiri, E.; Shahsavari, M.; Hassanli, K.; van Schaik, A.; Eshraghian, J. Spiking neural networks on FPGA: A survey of methodologies and recent advancements. *Neural Netw.* **2025**, *186*, 107256. [CrossRef]
23. Lin, Y.; Xie, Z.; Chen, T.; Cheng, X.; Wen, H. Image privacy protection scheme based on high-quality reconstruction DCT compression and nonlinear dynamics. *Expert Syst. Appl.* **2024**, *257*, 124891. [CrossRef]
24. Zhang, L.; Lin, Y.; Yang, X.; Chen, T.; Cheng, X.; Cheng, W. From Sample Poverty to Rich Feature Learning: A New Metric Learning Method for Few-Shot Classification. *IEEE Access* **2024**, *12*, 124990–125002. [CrossRef]
25. Rani, M.; Yadav, J.; Rathee, N.; Panjwani, B. Optifusion: Advancing visual intelligence in medical imaging through optimized CNN-TQWT fusion. *Vis. Comput.* **2024**, *40*, 7075–7092. [CrossRef]
26. Shang, J.; Zhang, K.; Zhang, Z.; Li, C.; Liu, H. A high-performance convolution block oriented accelerator for MBConv-Based CNNs. *Integration* **2022**, *88*, 298–312. [CrossRef]
27. Cittadini, E.; Marinoni, M.; Buttazzo, G. A hardware accelerator to support deep learning processor units in real-time image processing. *Eng. Appl. Artif. Intell.* **2025**, *145*, 110159. [CrossRef]
28. Liu, F.; Li, H.; Hu, W.; He, Y. Review of neural network model acceleration techniques based on FPGA platforms. *Neurocomputing* **2024**, *610*, 128511. [CrossRef]
29. Nandanwar, H.; Katarya, R. Deep learning enabled intrusion detection system for Industrial IOT environment. *Expert Syst. Appl.* **2024**, *249*, 123808. [CrossRef]
30. Wang, B.; Yu, D. Orthogonal Progressive Network for Few-shot Object Detection. *Expert Syst. Appl.* **2025**, *264*, 125905. [CrossRef]
31. Li, T.; Jiang, H.; Mo, H.; Han, J.; Liu, L.; Mao, Z.G. Approximate Processing Element Design and Analysis for the Implementation of CNN Accelerators. *J. Comput. Sci. Technol.* **2023**, *38*, 309–327. [CrossRef]
32. Chatterjee, S.; Pandit, S.; Das, A. Coupling of a lightweight model of reduced convolutional autoencoder with linear SVM classifier to detect brain tumours on FPGA. *Expert Syst. Appl.* **2025**, *290*, 128444. [CrossRef]
33. 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 (FPGA '16), New York, NY, USA, 21–23 February 2016; pp. 26–35. [CrossRef]
34. Zhang, M.; Li, L.; Wang, H.; Liu, Y.; Qin, H.; Zhao, W. Optimized Compression for Implementing Convolutional Neural Networks on FPGA. *Electronics* **2019**, *8*, 295. [CrossRef]

35. Luo, Y.; Cai, X.; Qi, J.; Guo, D.; Che, W. FPGA–accelerated CNN for real-time plant disease identification. *Comput. Electron. Agric.* **2023**, *207*, 107715. [CrossRef]
36. Kim, J.; Kang, J.K.; Kim, Y. A Low-Cost Fully Integer-Based CNN Accelerator on FPGA for Real-Time Traffic Sign Recognition. *IEEE Access* **2022**, *10*, 84626–84634. [CrossRef]
37. Shi, K.; Wang, M.; Tan, X.; Li, Q.; Lei, T. Efficient Dynamic Reconfigurable CNN Accelerator for Edge Intelligence Computing on FPGA. *Information* **2023**, *14*, 194. [CrossRef]
38. Bouguezzi, S.; Fredj, H.B.; Belabed, T.; Valderrama, C.; Faiedh, H.; Souani, C. An Efficient FPGA-Based Convolutional Neural Network for Classification: Ad-MobileNet. *Electronics* **2021**, *10*, 2272. [CrossRef]
39. Cai, L.; Wang, C.; Xu, Y. A Real-Time FPGA Accelerator Based on Winograd Algorithm for Underwater Object Detection. *Electronics* **2021**, *10*, 2889. [CrossRef]
40. Fuketa, H.; Katashita, T.; Hori, Y.; Hioki, M. Multiplication-Free Lookup-Based CNN Accelerator Using Residual Vector Quantization and Its FPGA Implementation. *IEEE Access* **2024**, *12*, 102470–102480. [CrossRef]
41. Xuan, L.; Un, K.F.; Lam, C.S.; Martins, R.P. An FPGA-Based Energy-Efficient Reconfigurable Depthwise Separable Convolution Accelerator for Image Recognition. *IEEE Trans. Circuits Syst. II Express Briefs* **2022**, *69*, 4003–4007. [CrossRef]
42. Chollet, F. Xception: Deep Learning with Depthwise Separable Convolutions. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Los Alamitos, CA, USA, 21–26 July 2017; pp. 1800–1807. [CrossRef]
43. Hu, J.; Shen, L.; Sun, G. Squeeze-and-Excitation Networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Los Alamitos, CA, USA, 18–22 June 2018; pp. 7132–7141. [CrossRef]
44. 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* **2017**, arXiv:1704.04861.
45. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Los Alamitos, CA, USA, 18–22 June 2018; pp. 4510–4520. [CrossRef]
46. Howard, A.; Sandler, M.; Chen, B.; Wang, W.; Chen, L.C.; Tan, M.; Chu, G.; Vasudevan, V.; Zhu, Y.; Pang, R.; et al. Searching for MobileNetV3. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Los Alamitos, CA, USA, 27 October–2 November 2019; pp. 1314–1324. [CrossRef]
47. Neris, R.; Rodríguez, A.; Guerra, R.; López, S.; Sarmiento, R. FPGA-Based Implementation of a CNN Architecture for the On-Board Processing of Very High-Resolution Remote Sensing Images. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2022**, *15*, 3740–3750. [CrossRef]
48. Zhang, Y.; Wang, H.; Pan, Z. An efficient CNN accelerator for pattern-compressed sparse neural networks on FPGA. *Neurocomputing* **2025**, *611*, 128700. [CrossRef]
49. Zhang, Y.; Jiang, H.; Liu, X.; Cao, H.; Du, Y. High-efficient MPSoC-based CNNs accelerator with optimized storage and dataflow. *J. Supercomput.* **2022**, *78*, 3205–3225. [CrossRef]
50. 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.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 35–47. [CrossRef]