applied sciences

MDPI

Article

Research on the Lightweight Deployment Method of Integration of Training and Inference in Artificial Intelligence

Yangyang Zheng, Bin He * and Tianling Li

College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China; 2020200780@mail.buct.edu.cn (Y.Z.); 2021200821@mail.buct.edu.cn (T.L.)
* Correspondence: hebin@mail.buct.edu.cn

Abstract: In recent years, the continuous development of artificial intelligence has largely been driven by algorithms and computing power. This paper mainly discusses the training and inference methods of artificial intelligence from the perspective of computing power. To address the issue of computing power, it is necessary to consider performance, cost, power consumption, flexibility, and robustness comprehensively. At present, the training of artificial intelligence models mostly are based on GPU platforms. Although GPUs offer high computing performance, their power consumption and cost are relatively high. It is not suitable to use GPUs as the implementation platform in certain application scenarios with demanding power consumption and cost. The emergence of high-performance heterogeneous architecture devices provides a new path for the integration of artificial intelligence training and inference. Typically, in Xilinx and Intel's multi-core heterogeneous architecture, multiple high-performance processors and FPGAs are integrated into a single chip. When compared with the current separate training and inference method, heterogeneous architectures leverage a single chip to realize the integration of AI training and inference, providing a good balance of training and inference of different targets, further reducing the cost of training and implementation of AI inference and power consumption, so as to achieve the lightweight goals of computation, and to improve the flexibility and robustness of the system. In this paper, based on the LeNet-5 network structure, we first introduced the process of network training using a multi-core CPU in Xilinx's latest multi-core heterogeneous architecture device, MPSoC. Then, the method of converting the network model into hardware logic implementation was studied, and the model parameters were transferred from the processing system of the device to the hardware accelerator structure, composed of programmable logic through the bus interface AXI provided on the chip. Finally, the integrated implementation method was tested and verified in Xilinx MPSoC. According to the test results, the recognition accuracy of this lightweight deployment scheme on MNIST dataset and CIFAR-10 dataset reached 99.5 and 75.4% respectively, while the average processing time of the single frame was only 2.2 ms. In addition, the power consumption of the network within the SoC hardware accelerator is only 1.363 W at 100 MHz.

Keywords: deep learning; MPSoC; FPGA; CNN

Citation: Zheng, Y.; He, B.; Li, T. Research on the Lightweight Deployment Method of Integration of Training and Inference in Artificial Intelligence. *Appl. Sci.* **2022**, *12*, 6616. https://doi.org/10.3390/app12136616

Academic Editors: Qizhi Xu, Jin Zheng and Feng Gao

Received: 22 May 2022
Accepted: 28 June 2022
Published: 29 June 2022

1. Introduction

In recent years, CNN has been widely used in artificial intelligence, especially in the application of image classification [1,2]. However, due to the large amount of computation of convolutional neural networks, GPUs are traditionally used for training and then deployment. Although this method solves the problem of high computation requirements in CNN, it cannot address the problems of flexibility, cost, and power consumption in the application of CNN [3,4]. In specific scenarios, such as astronomical devices and Internet of Things devices, it is often difficult to satisfy the energy consumption of GPUs [5]. Therefore, researchers have proposed the method of CNN acceleration using FPGA, ASIC, etc., and compared them with GPUs [3,6,7]. In addition, the separation of training and deployment

makes the adaptability of the neural network to the environment less than ideal. For example, in some complex application scenarios, the parameters of the network model are also required to be adjusted dynamically rather than remaining constant. In a heterogeneous architecture device based on FPGA, on the one hand, a high-performance multi-core processor is integrated, which makes it possible to train the network and transmit the parameters of the neural network in the device. On the other hand, the hardware accelerator structure of CNN is realized in the programmable logic part of the device, so that a neural network can achieve the optimal power consumption, cost, and performance [8–10]. At the same time, it overcomes the poor flexibility caused by the separation of training and inference in the traditional deployment approach, and significantly improves the responsiveness of AI and the ability to adapt to the needs of different scenarios. Figure 1 shows the architecture of the lightweight deployment implementation approach proposed in this paper.

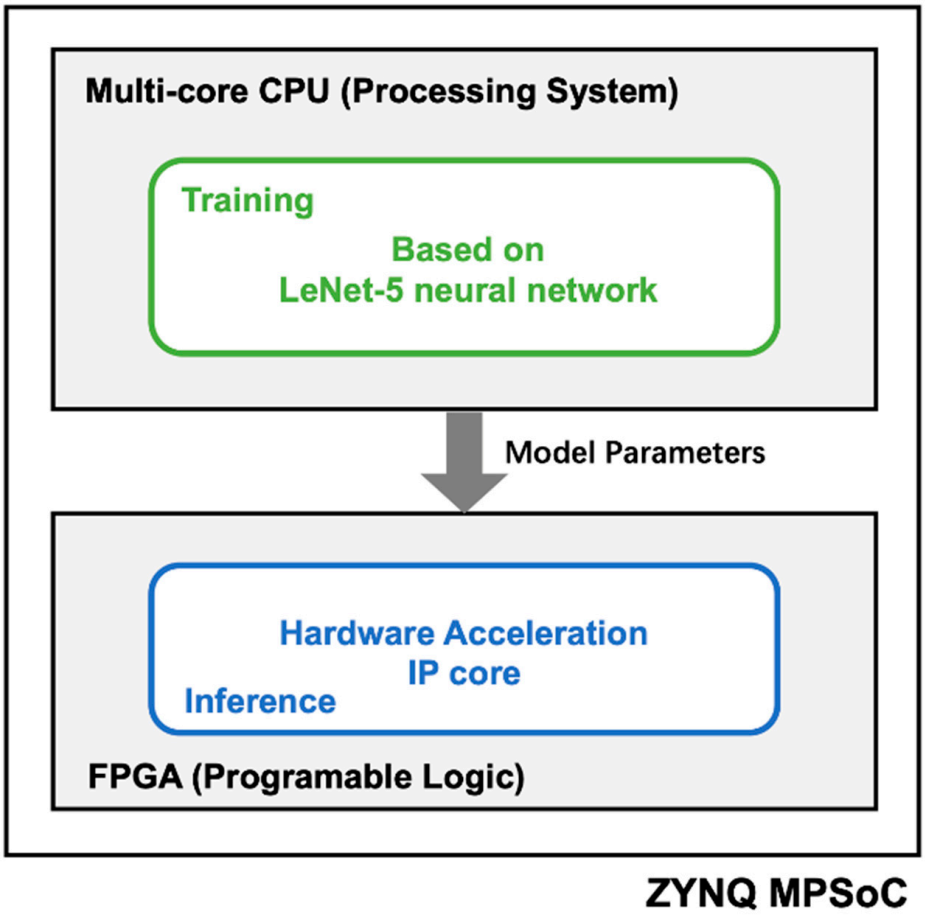


Figure 1. Implementation method of artificial intelligence lightweight deployment based on heterogeneous architecture. We train the neural network in PS, and then send the parameters to the accelerator in PL.

To sum up, the main contributions of this paper are listed as follows:

1. Taking advantage of the characteristics of the heterogeneous architecture, we propose a new lightweight deployment scheme of a neural network model, which enables AI applications in different scenarios to achieve a better balance between flexibility, performance, cost, power consumption and anti-interference capabilities. The integrated deployment of neural network training and forward inference acceleration with flexible weight adjustment has been realized.
2. Using Programable Logic (PL) of MPSoC, the balance between pipeline and parallelism was realized in the neural network layer based on the limited resources, and the performance of neural network forward inference was optimized. After optimizing

and packaging each layer, a flexible and customizable hardware-accelerated IP library was constructed.

3. We deployed a neural network training framework in the Processing System (PS) of MPSoC to support neural network training. The automatic network weight parameter migration script was written to realize the data processing automation of lightweight deployment integration.

The rest of this article is as follows: Section 2 introduces some of the related pioneering work. Section 3 describes the lightweight deployment method of the neural network, introducing our architecture characteristics and the construction method of the LeNet-5 network training in the PS side and IP core acceleration in the PL side. Section 4 introduces the experiment of inference accelerated IP core, including the design of the SoC and verifies the function of accelerated IP core. In Section 5, the performance of the accelerator IP core is analyzed according to the experiment results. Then, we summarize our contribution and look into the future research direction in Section 6. Finally, we summarize the work of this paper in Section 7.

2. Related Work

The concept of CNN originated in the 1960s. Hubel and Wiesel, two neuroscientists, discovered the information processing pattern in the biological visual system and thus proposed the concept of receptive fields, proving that the visual cortex is hierarchical, which is the theoretical basis of the convolutional neural network. In 1990, Lecun conducted supervised learning on Neocoginitro network, applied back propagation to the training process, and made subsequent improvements. In 1998, Lecun proposed an artificial neural network with a multi-layer structure, which is the famous LeNet-5 model [11]. In order to solve complex problems in various application scenarios, scientists have continued to improve and innovate CNN and put forward different models. As a typical representation of CNN, LeNet-5 contains the basic modules of CNN, namely, the convolution layer, pooling layer, activation layer, and full connection layer.

The MNIST data set is a large handwritten character data set, including handwritten numeral images from zero to nine, with 60,000 training images and 10,000 test images. The image size is 28×28 pixels, and is a single-channel grayscale image [12]. The CIFAR-10 data set is suitable for a small data set of universal object recognition and contains 10 categories of RGB color images with a size of 32×32 pixels [13]. The 10 categories include airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset contains a total of 50,000 training images and 10,000 test images. In general, these two datasets are used to train various image processing systems and are widely used in the field of deep learning.

As mentioned in the Introduction, neural networks are algorithmically complex and require huge computing power for training and inference. From the perspective of a light weight and flexibility, FPGAs, which have developed rapidly in recent years, have become increasingly integrated, including more and more on-chip RAMs, DSPs for arithmetic operations, and reconfigurable logic units [7]. When compared with GPUs, FPGA has the characteristics of high resource density, high performance-to-power ratio, and flexibility, so it is more suitable for harsh operating environments such as outer space [3]. From the perspective of performance and power consumption, FPGA provides superior performance/Watt over CPU and GPU, because FPGA's on-chip BRAMs, hard DSPs, and reconfigurable fabric allow for efficiently extracting fine-grained parallelisms from small/medium size matrices [14]. Despite the popularity of GPUs in deep learning implementations, FPGAs are emerging as an alternative mainly because of advantages such as their high performance per watt, parallelism, and their flexibility in model level optimizations when compared to the fixed architectures of GPUs [15]. In low-power environments, FPGAs are more suitable for mobile devices and resource-limited platforms.

With the development of deep learning, the CNN model structure has become more and more complex, and it has become very difficult to conduct rapid validation and hardware-accelerated calculation for the model; at this stage, FPGA has come into the

sight of AI scientists. Seeking new CNN hardware acceleration solutions has gradually become a hot topic in the field of AI [16–18]. More and more deep learning models use accelerators based on FPGAs. S. I. Venieris and C. Bouganis developed fpgaConvNet, a new domain-specific modeling framework [19]. It implements an automatic design method for mapping ConvNets to a reconfigurable platform based on FPGA. When compared with the previous method, the hardware structure generated by this automatic design method can improve the performance density by 1.62 times. Based on Xilinx's SDAccel tool and FPGA, R. DiCecco et al. implemented the Winograd convolution engine, which can be used with other layers running on host processors to run multiple popular CNNs [20]. The results show that the framework achieved 50 GFLOPS in benchmark tests (using 3×3 convolution). Based on Xilinx Aritix-7 series FPGA, Hua optimized the design of a light-weight handwritten numeral system, and realized numeral recognition through the compressed CNN network, realizing the hardware acceleration of CNN and significantly improving the speed and accuracy of recognition [21]. Shahmustafa Mujawar et al. proposed and verified the architecture of CNN based on Xilinx FPGA using the MNIST handwritten dataset [22]. The adopted approaches of a sliding filter for convolution and parallel computation of Multiplication and Accumulation (MAC) operations resulted in an optimized hardware architecture with reduced arithmetic operations and faster computations. The architecture they mentioned has been implemented on Artix-7 FPGA and attained a significant improvement in speed when compared to existing architecture working at 300 MHz. Huang et al. proposed a novel composite hardware CNN accelerator architecture to solve the problem of the inefficient computing resource mapping mechanism and data supply [23]. They proposed a multi-CE architecture based on a row-level pipelined streaming strategy for convolution layers and a single-CE architecture based on a batch-based computing method for full-connection layers. Zhang et al. compared various FPGA-based CNN implementations, and finally achieved a peak performance of 61.62 GFLOPS under a 100 MHz working frequency, which outperforms previous approaches significantly, based on the VC707 FPGA board [24]. However, the flexibility of accelerator solutions using pure FPGA is obviously inferior to SoC with a heterogeneous architecture based on FPGA. The multi-core high-performance ARM processor integrated into SoC will provide a better solution for the deployment of CNN.

The neural network deployment of SoC based on heterogeneous architecture has gradually become a hot topic. Y. A. Bachtiar and T. Adiono proposed a configurable CNN and maximum pool processor architecture based on a low-capacity SoC, and completed the algorithm validation on Xilinx Zynq-7000 SoC [25]. In order to reduce the calculation time, S. Ghaffari and S. Sharifian proposed a new method to calculate the hyperbolic tangent activation function, and implemented the LeNet convolutional network architecture and used MNIST data sets as samples to identify handwritten numerals [26]. The experimental results show that the model reduces the latency significantly while maintaining accuracy. Based on Zynq SoC, Huang realized the design of the image classification recognition system, and used the Vivado HLS tool to design the IP core of the image classification system [27]. When compared with the prototype, the accelerator implementation method achieves up to 43 times greater acceleration on the CIFAR-10 data set, and maintains 73.7% classification accuracy and a low power consumption of 2.063 W, which indicates that the heterogeneous architecture SoC can better realize image classification and recognition, and has better real-time performance in embedded image applications. Liu et al. implemented a CNN accelerator based on the ZYNQ heterogeneous platform by coordinating resources and bandwidth for the design [28]. The accelerator has a high resource utilization and can accelerate both standard convolution and depth-wise separable convolution. The experimental results show that the accelerator designed in their paper can achieve 17.11 GOPS for 32 bit floating point, while it can also accelerate depth-wise separable convolution, which has obvious advantages when compared with other designs. Zhang et al. proposed a reconfigurable CNN accelerator with an AXI bus based on ARM + FPGA architecture [29]. They implemented the proposed architecture on the Xilinx ZCU102 FPGA for YOLOv2

and YOLOv2 Tiny models on COCO and VOC 2007 respectively, with peak performance of 289 GOPs at 300 MHz clock frequency. To satisfy the demand of mobile computing and low-power application scenes, Xie et al. proposed a general reconfigurable embedded system design of convolution neural networks based on Altera Cyclone-IV FPGA [30]. Based on the cooperation and interaction between hardware and software, the system is able to accomplish image identification and other CNN works with high speed and low power. The system achieves a peak performance of 59.52 GOPS under a 120 MHz working frequency, with the power of 1.35 W. P. Meloni et al. proposed an accelerator architecture that is suitable to be implemented on mid-to high-range FPGA devices, that can be re-configured at runtime to adapt to different filter sizes in different convolution layers [31]. It peaks at 120 GMAC/s and 129 GMAC/s when executing 5×5 and 3×3 filters, respectively, and consumes less than 10 W of power. More studies show that it is feasible to construct a new neural network inference framework based on a heterogeneous architecture SoC, and it has better flexibility in data transmission and optimization, and is better than other implementation schemes in terms of performance and power consumption [32–37].

To sum up, hardware acceleration based on heterogeneous architecture SoC makes the deployment of neural network more flexible, but FPGA still needs to be reconfigured repeatedly in implementation, and its adaptability to new scenarios has been less capable. Therefore, it is of more important research value to make full use of multi-core high-performance processors in heterogeneous architecture to build neural network training environments and improve the robustness of CNN deployment, which will provide a better solution for the application of artificial intelligence lightweight deployment.

3. Method

3.1. Integrated Architecture of Artificial Intelligence

In general, the lightweight deployment of artificial intelligence includes two aspects. One is to train the model with the multi-core processor part in the heterogeneous architecture device, and then obtain the trained model parameters. The other is to construct and optimize the inference model of the network in the programmable logic part of the heterogeneous architecture device. The inference model receives the model parameters obtained by training through the bus interface inside the SoC.

Figure 2 shows the overall implementation structure of lightweight AI integration. The neural network training framework was deployed at the PS side of MPSoC to support the training of deep learning networks. At the same time, IP core of neural network accelerator was constructed at the PL side. By transferring model weights and feature graph data to the IP core for neural network acceleration, the integration of neural network training and forward inference acceleration with flexible weight adjustment was finally realized. The implementation method is embodied in the following aspects:

(1) Firstly, a multi-core processor was used to train CNN in the PS terminal inside the SoC, and a large amount of image data was sent into the network. After several iterations, the CNN model with the best performance on the test set was finally obtained. Then, the model weights were exported by script and passed to the CNN accelerator IP core.
(2) The basic network layer of CNN was realized on the PL side inside the SoC, and these basic network layers were connected according to the network structure to form a complete neural network structure.
(3) Through advanced extensible interface (AXI) specifications, PS and PL were linked together to form a complete hardware structure for training and inference [38]. In this hardware structure, the CNN accelerator IP core was encapsulated into API functions that can be called directly by the Linux operating system, so as to constitute a complete implementation structure for training and acceleration.

3.2. Training Methods of Neural Networks

By running the neural network training framework PyTorch on an ARM multi-core processor in MPSoC, the training for AI lightweight deployment was realized. This method

can significantly reduce the overall power consumption of the system when training the network. For an application scenario with low requirements on the convergence speed of training model, using the high-performance multi-core CPU in MPSoC to perform network training can significantly reduce the overall cost and power consumption of the system. Figure 3 shows the modified LeNet-5 network model.

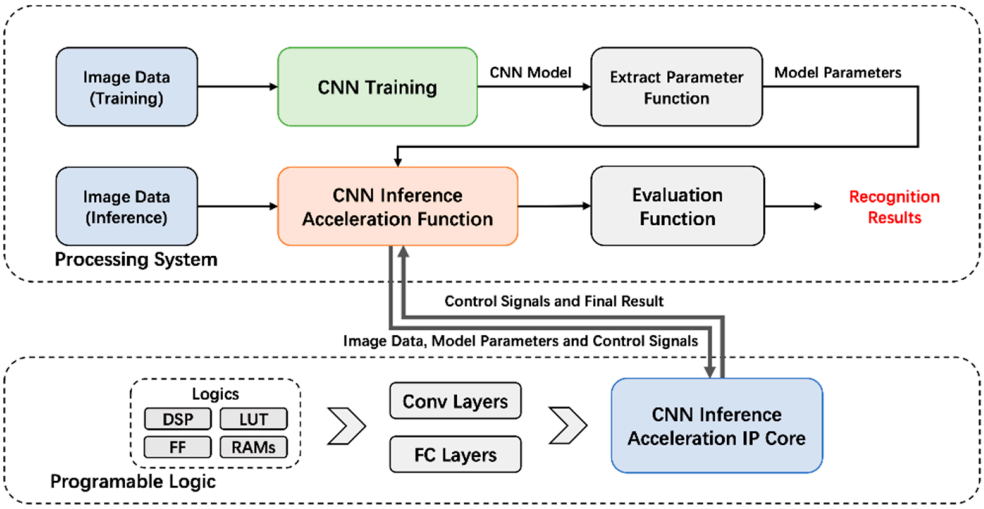


Figure 2. Overall system architecture. It mainly shows the direction of data flow in PS and PL and How to exchange data between PS and PL.

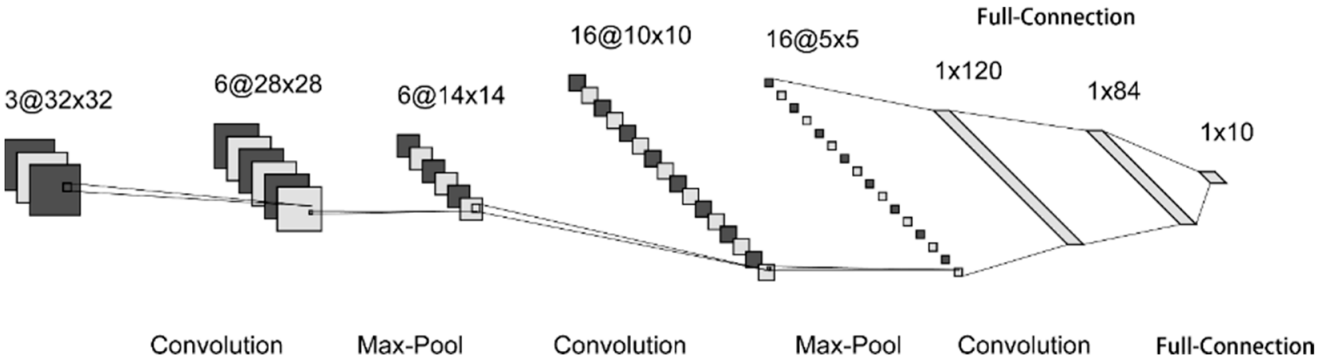


Figure 3. LeNet-5 network model. Compared with the original LeNet-5, we changed the input layer from a single channel to three channels.

In order to reduce the training load of PS in MPSoC, the neural network can be trained on the computer first and then the pre-trained model can be obtained when the implementation method of integrated artificial intelligence lightweight deployment is adopted. The model can be efficiently run in MPSoC's PS to complete the final training and extract the parameters of the network model.

The LeNet-5 network used in this paper has five network layers. The first three are the convolutional layers, and the input layer is the feature map of 32 × 32 pixels of single channel. Each convolutional layer is followed by the maximum pooling layer to control the network scale. The latter two are fully connected layers, and the last layer of the two fully connected layers completes the output of the image's ten categories. The LeNet-5 network uses the ReLU function as an activation function. The output of the output layer is normalized by the log_softmax function.

When the CIFAR-10 data set is used to train the network model, the input of the first convolution layer of LeNet-5 network is extended from single channel to three channels. When using MNIST data set to train the network model, it is necessary to copy the MNIST

data set in one-dimensional form into three-dimensional form, and then pass it to the input channel of the first convolution layer of LeNet-5 network.

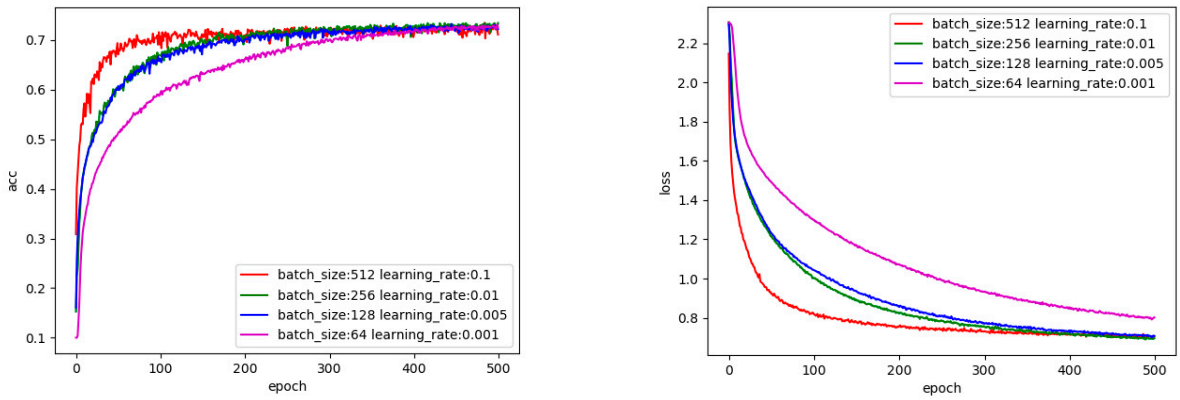
To train the modified LeNet-5 model, it is required to install an embedded Linux operating system on the PS side of the MPSoC to run the PyTorch framework. In the process of training the model, if the convergence is not ideal, the convergence speed of the model and the performance of the optimized model on the test set can be improved by adjusting the super-parameters such as learning rate and optimizer.

Both the CPU and GPU can support neural network training, and the CUDA acceleration platform launched by Nvidia accelerates neural network training. At present, the vast majority of neural network training is completed on GPUs. We compared the training speed and power consumption of the two platforms under the PyTorch framework. As shown in Table 1, the training time of GTX1050 to complete an epoch is 0.08 times that of MPSoC integrated multi-core, but the power consumption is about 30 times that of PS. In more demanding power consumption scenarios such as edge-computing systems, CPUs with a lower power consumption can also provide reliable support for neural network training, although they are not as fast as GPUs.

Table 1. Comparison of training performance.

	GPU	ZYNQ UltraScale + MPSoC
Device	GTX1050	quad-core Arm Cortex-A53
Power consumption	75 W	Less than 2 W
Training duration/epoch	22.6 s	286 s

When compared with GPUs, the computing performance of the multi-core processor in MPSoC is limited, so the batch size needed to be reduced to decrease the workload of the multi-core processor during training. As can be seen from the experiment results given in Figure 4b, within 500 epochs, no matter how the value of the batch size changed, the loss curve on the training set always maintained a normal downward trend, which would eventually make the model converge correctly without over-fitting. When combined with the experimental results given in Figure 4a, it can be seen that the larger the batch size value is, the faster the convergence rate of recognition accuracy is, which means the shorter the time required for training the model, thus significantly reducing the overall power consumption of the multi-core processor. Therefore, from the perspective of accuracy and loss, the value of batch size can be reduced in practical model training, so that the convergence speed of the model will not be significantly affected during training and the CPU load will not be too heavy.



(**a**) (**b**)

Figure 4. The effect of different batch sizes on the model convergence speed: (**a**) Recognition accuracy (acc) changes with epoch; (**b**) loss (loss) changes with epoch.

After the training is completed, the weight parameters of the LeNet-5 network can be extracted from the trained model through Python script. The model parameters are composed of a "key" value and "value" value. Key is the name of the network layer, value is the parameter value of the corresponding network layer, and the data type is a tensor.

In this design, first of all, the corresponding value was extracted according to the network layer name key, and its data type was transformed from the tensor into the array numpy of data type FP32. Then, we saved it to a file named after each layer; finally, the model parameters obtained by training were passed to the hardware accelerator for forward inference in PL by AXI specification.

3.3. Hardware Accelerator Structure Design of Convolution Layer

As mentioned above, the LeNet-5 network has five network layers, mainly including three convolution layers and two fully connected layers. Therefore, the structure of these two network layers can be optimized when designing a neural network accelerator. The first three network layers are all convolution layers with the same implementation methods. In this design, the convolution layer, activation function and pooling layer were placed in the same module, which reduces the difficulty of interface design and improves the readability and maintainability of the code.

The data flow operation process of hardware accelerator of convolution layer is divided into the convolution operation, ReLU activation function and pooling operation.

The convolution operation is expressed as:

$$f_i^{out} = \sum_{j=1}^{in} f_j^{in} \otimes w_{i,j} + b_i \tag{1}$$

where $w_{i,j}$ is the convolution kernel corresponding to the j-th input feature graph and the i-th output feature graph, and b_i is the bias of the i-th output feature graph.

The activation function ReLU is expressed as:

$$f(x) = max(x, 0) \tag{2}$$

where x is the value of the input feature graph, and $f(x)$ is the value of the output feature graph.

The pooling operation is expressed as:

$$f_{i,j}^{out} = \underset{p*p}{max} \begin{pmatrix} f_{m,n}^{in} & \cdots & f_{m,n+p-1}^{in} \\ \vdots & & \vdots \\ f_{m+p-1,n}^{in} & \cdots & f_{m+p-1,n+p-1}^{in} \end{pmatrix} \tag{3}$$

where p is the sampling core size of maximum pooling. The redundant information in the feature map can be eliminated by the pooling layer operation, which can reduce the amount of computation, but will not affect the accuracy of recognition.

Through analysis of Equations (1)–(3), the hardware accelerator operation process of the convolution layer is expressed as:

$$f_{conv\ layer}^{out} = MaxPool(ReLU(Conv(x, w_{kernel}, b))) \tag{4}$$

The convolutional layer operation model represented by Equation (4) was mapped to the most basic hardware accelerator structure, as shown in Figure 5. By optimizing the basic structure, the overall processing performance of the convolution layer hardware accelerator was further improved.

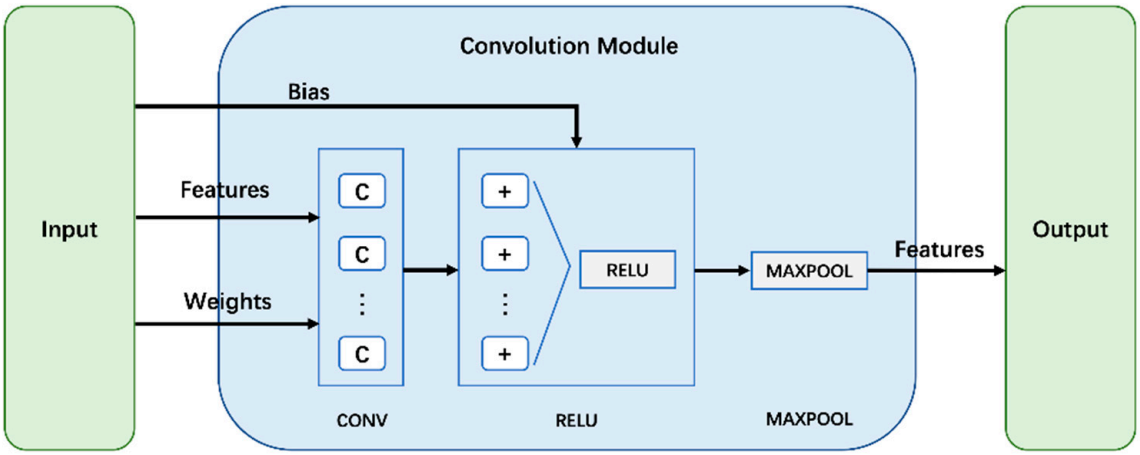


Figure 5. Implementation of convolution modules from the perspective of hardware logic.

When a software model is transformed into a hardware structure by the HLS tool, the expression of the software model may adversely affect the final hardware accelerator structure. Typically, the convolution operations given in Equation (1) are usually implemented in nested loops. In order to minimize the impact of high latency caused by repeated reading of feature graphs, the sequence of convolution operations is described as follows (Algorithm 1).

Algorithm 1: Convolution

OUT: Output
IN: Input
W: Weight
R: Row of output feature map
C: Column of output feature map
K: Kernel size
CHin: In-Channel size
CHout: Out-Channel size

```
1. for(kr = 0; kr < K; kr++)
2.     for(kc = 0; kc < K; kc++)
3.         for(r = 0; r < R; r++)
4.             for(c = 0; c < C; c++)
5.                 for(chi = 0; chi < Chin; chi++)
6.                     for(cho = 0; cho < CHout; cho++)
7.                         OUT[cho][r][c] = W[cho][chi][kr][kc] * IN[chi][r + kr][c + kc];
```

In the implementation of the hardware accelerator, the Pipeline command can be used to shorten the instruction trigger interval within the for cycle, so as to improve throughput and reduce delay. As shown in Table 2, in the absence of the Pipeline command, both the latency and interval for completing an image recognition are 7,709,158 clock cycles. After the Pipeline command was added in the fifth line of code of the algorithm given above, the latency and interval of one image recognition significantly reduced from 7,709,158 to 353,263 clock cycles, and the performance was about 22 times improved. However, resource utilization has also increased significantly. For example, the use of digital signal processing block DSP48E increased by about 27 times, and the use of flip-flop (FF) and Look-Up Table (LUT) has also increased by about 2 times. This is consistent with the tendency of hardware accelerators to increase the utilization of logical resources in exchange for improved data processing capability.

Table 2. Comparison of resource usage of convolutional layers before and after optimization.

		No Optimization	Pipelined
Latency & Interval (clock cycles)		7,709,158	353,263
Utilization Estimates	BRAM_18K	79	83
	DSP48E	9	244
	FF	29,061	53,049
	LUT	24,651	41,842

When using the Pipeline command, if the number of Pipeline layers is too high, which may lead to the rapid increase of resource usage or the delay cannot reach the expected value, the larger cycle can be divided into two smaller cycles as follows (the multiplication of the two smaller cycles equals the larger cycle times), so as to reduce the number of Pipeline layers (Algorithm 2).

Algorithm 2: For Loop Optimization

```
OUT: Output
IN: Input
KL: Inner loop
KH: Outer loop
K = KL * KH: Original loop count
    1. for(h = 0; h < KH; h++)
    2.     for(l = 0; l < KL; l++)
    3.         OUT[h * KL + l] = f ( IN[h * KL + l]);
```

When converting to hardware accelerator structure, after adding the Pipeline command at line 2 of the algorithm given above, the high-level synthesis tool will only expand the loop at line 2, which significantly reduces the number of Pipeline layers and resource consumption.

3.4. Hardware Accelerator Structure Design of the Full-Connection Layer

As mentioned earlier, the last two layers of the LeNet-5 network are fully connected layers. The hardware acceleration structure of the full connection layer is relatively simple, and its operation model can be expressed as:

$$f_{out} = w_{in,out} * f_{in} + b \tag{5}$$

where $w_{in,out}$ is the weight of the full connection layer, corresponding to the *in*-th input feature graph and the *out*-th output feature graph; b is the offset of the *out*-th input feature graph.

After the activating function ReLU, Equations (2) and (5) are combined to obtain the operation model of the full-connection layer as follows:

$$f^{out}_{fc\ layer} = ReLU\Big(Full\text{-}Connection\Big(x, w_{fc}, b\Big)\Big) \tag{6}$$

According to Equation (6), when the *Full-Connection* module is realized, the output result of the *Full-Connection* module can be obtained through the *ReLU* activation function after the weighted operation of the feature graph and weight. The most basic hardware accelerator structure of a *Full-Connection* module is shown in Figure 6.

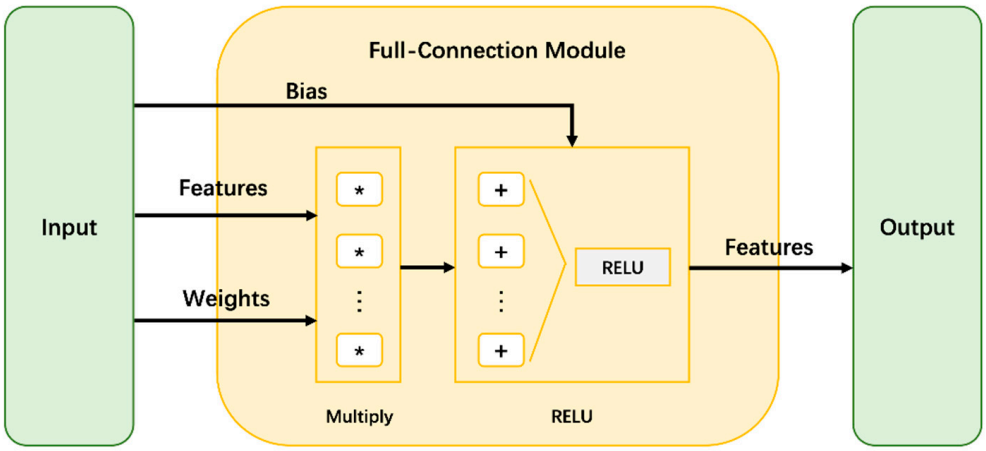


Figure 6. Implementation of full-connection modules from the perspective of hardware logic.

For the fully connected layer, the input f_{in} and output f_{out} are one-dimensional arrays with the weight $w_{in,out}$ being two-dimensional arrays. The model of the full-connection layer can be represented as follows (Algorithm 3).

Algorithm 3: Full-Connection

```
OUT: Output
IN: Input
W: Weight
CHin: In-Channel size
CHout: Out-Channel size
1. for(chi = 0; chi < Chin; chi++)
2.     for(cho = 0; cho < CHout; cho++)
3.         OUT[cho] = W[cho][chi] * IN[chi];
```

Using the Unroll command, users can transform a fully connected layer software model represented by a loop into an efficient hardware accelerator structure. As shown in Table 3, adding the Unroll command at line 2 of the algorithm given above reduces the latency and interval of the full-connection layer hardware accelerator from 128,856 to 45,696 clock cycles. Obviously, the throughput has improved by about three times. Again, this is to increase the overall utilization of logical resources to improve the data-processing capacity of the whole connection layer.

Table 3. Comparison of resource usage of full-connection layers before and after optimization.

		No Optimization	Pipelined
FC Layers' Latency & Interval (clock cycles)		128,856	45,696
Utilization Estimates	BRAM_18K	79	88
	DSP48E	9	14
	FF	29,061	29,259
	LUT	24,651	28,196

4. Experiments

4.1. Creating the Validation Platform

Xilinx XCZU3EG-SBVA484 MPSoC was used as the core device to verify the design of integrated artificial intelligence lightweight deployment. An ARM quad-core Cortex-A53 application processing unit and ARM dual-core Cortex-R5F real-time processing unit were integrated in the PS side of the device. The PL side of the device integrates up to 7.6 MB of Block RAM (BRAM) and up to 360 digital signal processing modules, known as DSP48E, as

well as other logical resources. The hardware platform based on this device was equipped with 2 GB LPDDR4 memory [39].

Based on the modified LeNet-5 network and IP packaging and reuse technology, the design created an image recognition system within MPSoC [40,41]. As shown in Figure 7, the hardware accelerator IP core used to realize image recognition function was connected to the PS side of the chip through AXI specification. In the figure, the IP core functions of each module are as follows:

(1) LeNet-5 module. This module implements the image recognition hardware accelerator structure based on the modified LeNet-5 network structure.
(2) Zynq Ultrascale + MPSoC module. This module is an abstraction of PS in the MPSoC. It contains the quad-core Cortex-A53 APU and the dual-core Cortex-R5F RPU.
(3) AXI Interconnect module. In compliance with the AXI specification, multiple AXI memory-mapped master devices were connected to multiple memory-mapped slave devices through a switch structure, which was used as a bridge to connect S_AXI peripherals.
(4) AXI SmartConnect module. Similar to AXI Interconnect, it was used to connect AXI peripherals to PS, in this structure primarily as a bridge to connect M_AXI.
(5) Process System Reset module. The processor system reset module was used in this structure to generate reset signals for PS and three other modules.

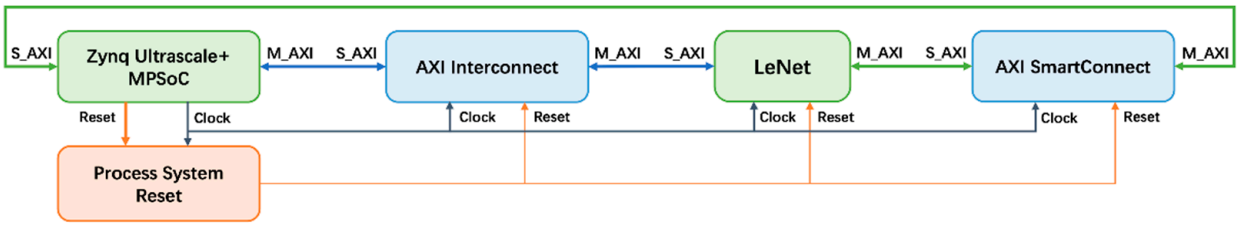


Figure 7. Design diagram of system on chip built in MPSoC. It mainly shows the connection relationship and data flow of each IP core.

After Xilinx Vivado software's synthesis and implementation of the overall structure, the logical resources used in PL are shown in Table 4.

Table 4. Resource consumption report after Vivado synthesis and implementation.

Sources	Utilization Estimates	Utilization (%)
BRAMs	41.5	19.2
DSP	248	68.89
FF	51,434	36.45
LUT	48,966	69.40
LUTRAM	3288	11.42

According to the power analysis tool in Vivado software, the total power consumption of the image recognition body system is only 3.22 W, while the power consumption of LeNet-5, the hardware accelerator module used to realize the forward inference of the image recognition, is only 1.363 W.

4.2. Design of Validation Method

Based on the PYNQ framework, the software code of image recognition system was written in Python [42]. The overlay programming library provided within the framework generates callable Python APIs for the IP core, making the system's hardware and software co-design easier. The data flow of the system is shown in Figure 8.

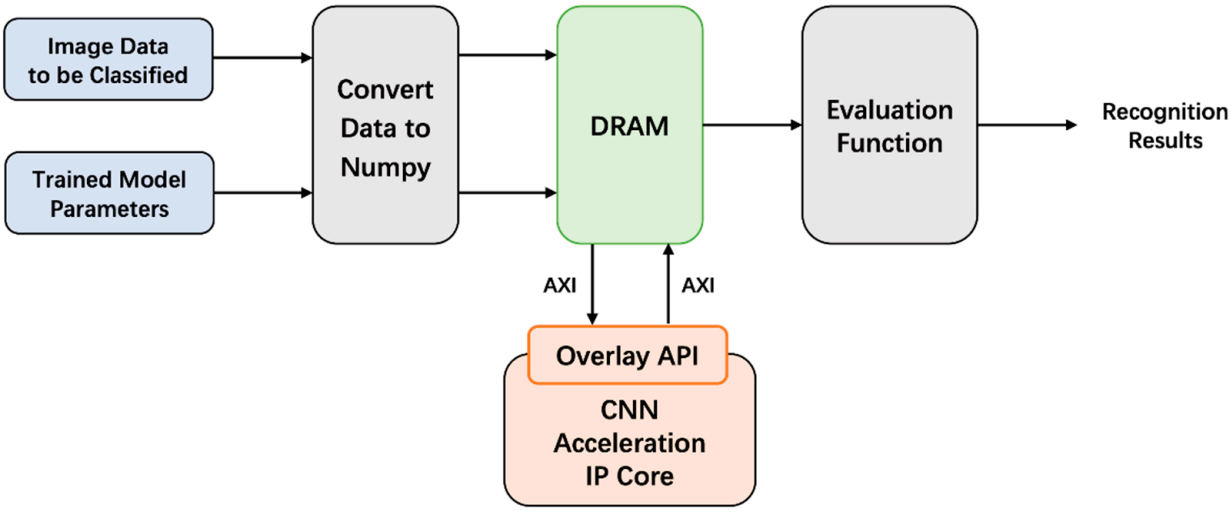


Figure 8. Data flow of accelerated inference on the PL side. It mainly shows the process of data processing by software and the way of interacting with hardware.

After importing the image data to be recognized and the model parameters that have been trained, they are transformed into an FP32 numpy array. These numpy arrays are stored in DRAM, waiting to be read by the CNN accelerator. The interface of the CNN accelerator IP core built on the PL side will be packaged by the overlay programming library as a callable Python API. When the program is run to call the accelerator, the numpy array stored in DRAM is passed into the accelerator through the AXI bus for forward inference calculation. When the accelerator execution is complete, the returned results are transmitted back to DRAM over the AXI bus. Finally, the classification result is obtained after the evaluation function processing.

5. Results

Through the validation of 10,000 images on the test set, the inference speed of an accelerator IP check single frame is 2.2 ms, and the average number of recognized frames is about 450 frames per second. The processing time of single frame of the model on the Inter I5-4300U CPU is 8.1 ms, which not only improves the inference speed by about four times, but also has more advantages in power consumption performance, satisfying more lightweight deployment requirements. The performance comparison of the two devices is shown in Table 5.

Table 5. Performance comparison of CPU and MPSoC.

	CPU	ZYNQ UltraScale + MPSoC
Device	Inter i5-4300U	XCZU3EG-SBVA484
Frequency	2.5 GHz	100 MHz
Power	44 W	3.22 W
Inference time (single frame)	8.1 ms	2.2 ms

Finally, the recognition accuracy of the accelerator IP checking the MNIST handwritten numeral set is 99.5%, and the recognition accuracy of CIFAR-10 validation set classification is 75.4%, which are consistent with the performance of the model on the computer. The screenshot below shows part of the experiment results.

Loading MNIST network weights and using test sets to verify the results are shown in Figure 9:

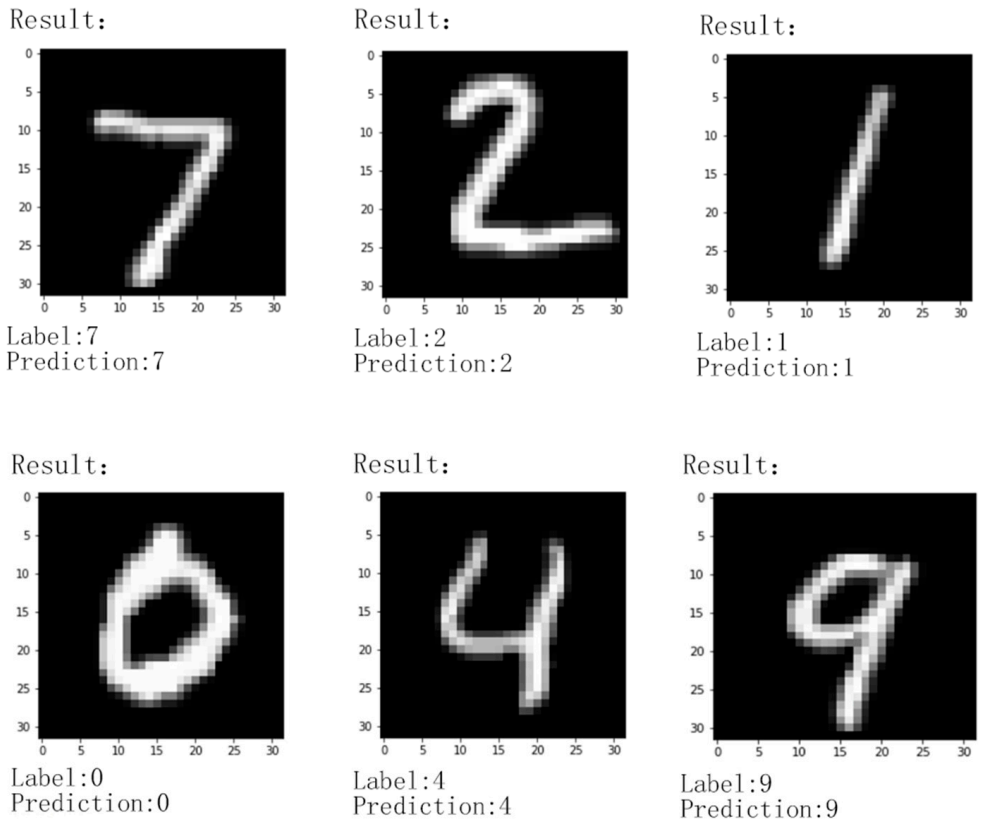


Figure 9. Recognition results of the accelerator on the MNIST dataset.

The result of loading CIFAR-10 network weights and using test set validation is shown in Figure 10:

Figure 10. Recognition results of the accelerator on the CIFAR-10 dataset.

6. Discussion

In this paper, MPSoC was used to build the training and inference model of the integrated LeNet-5 network, which better meets the different requirements of training and inference and significantly reduces the implementation cost and power consumption. The flexibility of data processing between training and inference was addressed in terms of computational power. Since the implementation method of a single chip was adopted, the robustness of the entire system was significantly improved in terms of computational power, and a new solution was found for artificial intelligence in the application scenarios with strict requirements for cost, performance and power consumption.

We believe that with the continuous development of semiconductor integrated circuit technology, there will be more and more high-performance heterogeneous architecture devices in the future, so as to provide a better computing power platform for the development of artificial intelligence, and to drive the continuous development of artificial intelligence technology, so that it can better serve the progress of humanity. In future studies, we will focus on applying this architecture to more complex application scenarios and explore the implementation of more complex network models.

7. Conclusions

In this paper, based on the LeNet-5 network structure, we first introduced the process of network training using a multi-core CPU in Xilinx's latest multi-core heterogeneous architecture device, MPSoC. Then, the method of converting the network model into hardware logic implementation was studied, and the model's parameters were transferred from the processing system of the device to the hardware accelerator structure, composed of programmable logic through the bus interface AXI provided on the chip. Finally, the integrated implementation method was tested and verified in Xilinx MPSoC. When compared with the current separate training and inference method, heterogeneous architectures leverage a single chip to realize the integration of AI training and inference, providing a good balance of training and inference of different targets, further reducing the cost of training and implementation of AI inference and power consumption, so as to achieve the lightweight goals of computation, and to imporve the flexibility and robustness of the system.

Author Contributions: Conceptualization, Y.Z.; Investigation, Y.Z.; Methodology, B.H.; Supervision, B.H.; Visualization, T.L.; Writing—original draft, Y.Z.; Writing—review & editing, B.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Alzubaidi, L.; Zhang, J.; Humaidi, A.J.; Al-Dujaili, A.; Duan, Y.; Al-Shamma, O.; Santamaría, J.; Fadhel, M.A.; Al-Amidie, M.; Farhan, L. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *J. Big Data* **2021**, *8*, 53. [CrossRef] [PubMed]
2. Pak, M.; Kim, S. A review of deep learning in image recognition. In Proceedings of the 2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT), Kuta Bali, Indonesia, 8–10 August 2017; pp. 1–3.
3. Hu, Y.; Liu, Y.; Liu, Z. A Survey on Convolutional Neural Network Accelerators: GPU, FPGA and ASIC. In Proceedings of the 2022 14th International Conference on Computer Research and Development (ICCRD), Shenzhen, China, 7–9 January 2022; pp. 100–107.
4. Zaman, K.S.; Reaz, M.B.I.; Ali, S.H.M.; Bakar, A.A.A.; Chowdhury, M.E.H. Custom Hardware Architectures for Deep Learning on Portable Devices: A Review. *IEEE Trans. Neural Networks Learn. Syst.* **2021**, *32*, 1–21. [CrossRef] [PubMed]
5. Vipin, K. ZyNet: Automating Deep Neural Network Implementation on Low-Cost Reconfigurable Edge Computing Platforms. In Proceedings of the 2019 International Conference on Field-Programmable Technology (ICFPT), Tianjin, China, 9–13 December 2019; pp. 323–326.

6. Colbert, I.; Daly, J.; Kreutz-Delgado, K.; Das, S. A competitive edge: Can FPGAs beat GPUs at DCNN inference acceleration in resource-limited edge computing applications? *arXiv* **2021**, arXiv:2102.00294.
7. Nurvitadhi, E.; Sheffield, D.; Sim, J.; Mishra, A.; Venkatesh, G.; Marr, D. Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC. In Proceedings of the 2016 International Conference on Field-Programmable Technology (FPT), Xi'an, China, 7–9 December 2016; pp. 77–84.
8. Lacey, G.; Taylor, G.W.; Areibi, S. Deep learning on fpgas: Past, present, and future. *arXiv* **2016**, arXiv:1602.04283.
9. Dias, M.A.; Ferreira, D.A.P. Deep Learning in Reconfigurable Hardware: A Survey. In Proceedings of the 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Rio de Janeiro, Brazil, 20–24 May 2019; pp. 95–98.
10. Seng, K.P.; Lee, P.J.; Ang, L.M. Embedded Intelligence on FPGA: Survey, Applications and Challenges. *Electronics* **2021**, *10*, 895. [CrossRef]
11. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
12. Deng, L. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Process. Mag.* **2012**, *29*, 141–142. [CrossRef]
13. CIFAR-10 and CIFAR-100 datasets. Available online: https://www.cs.toronto.edu/~{}kriz/cifar.html (accessed on 19 May 2022).
14. Nurvitadhi, E.; Sim, J.; Sheffield, D.; Mishra, A.; Krishnan, S.; Marr, D. Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC. In Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, 29 August–2 September 2016; pp. 1–4.
15. Sateesan, A.; Sinha, S.; KG, S.; Vinod, A.P. A Survey of Algorithmic and Hardware Optimization Techniques for Vision Convolutional Neural Networks on FPGAs. *Neural Process. Lett.* **2021**, *53*, 2331–2377. [CrossRef]
16. Hamdan, M.K.; Rover, D.T. VHDL generator for a high performance convolutional neural network FPGA-based accelerator. In Proceedings of the 2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, Mexico, 4–6 December 2017; pp. 1–6.
17. Liu, Z.; Dou, Y.; Jiang, J.; Xu, J. Automatic code generation of convolutional neural networks in FPGA implementation. In Proceedings of the 2016 International Conference on Field-Programmable Technology (FPT), Xi'an, China, 7–9 December 2016; pp. 61–68.
18. Ahmed, H.O.; Ghoneima, M.; Dessouky, M. Concurrent MAC unit design using VHDL for deep learning networks on FPGA. In Proceedings of the 2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE), Penang, Malaysia, 28–29 April 2018; pp. 31–36.
19. Venieris, S.I.; Bouganis, C. fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs. In Proceedings of the 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Washington, DC, USA, 1–3 May 2016; pp. 40–47.
20. DiCecco, R.; Lacey, G.; Vasiljevic, J.; Chow, P.; Taylor, G.; Areibi, S. Caffeinated FPGAs: FPGA framework for convolutional neural networks. In Proceedings of the 2016 International Conference on Field-Programmable Technology (FPT), Xi'an, China, 7–9 December 2016; pp. 265–268.
21. Hua, S. Design and optimization of a light-weight handwritten digital system based on FPGA. *Electron. Manuf.* **2020**, *16*, 6–7+37.
22. Mujawar, S.; Kiran, D.; Ramasangu, H. An Efficient CNN Architecture for Image Classification on FPGA Accelerator. In Proceedings of the 2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC), Bangalore, India, 9–10 February 2018; pp. 1–4. [CrossRef]
23. Huang, W.; Wu, H.; Chen, Q.; Luo, C.; Zeng, S.; Li, T.; Huang, Y. FPGA-Based High-Throughput CNN Hardware Accelerator With High Computing Resource Utilization Ratio. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *32*, 1–15. [CrossRef] [PubMed]
24. 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, Monterey, CA, USA, 22–24 February 2015; pp. 161–170.
25. Bachtiar, Y.A.; Adiono, T. Convolutional Neural Network and Maxpooling Architecture on Zynq SoC FPGA. In Proceedings of the 2019 International Symposium on Electronics and Smart Devices (ISESD), Badung, Indonesia, 25 November 2019; pp. 1–5.
26. Ghaffari, S.; Sharifian, S. FPGA-based convolutional neural network accelerator design using high level synthesize. In Proceedings of the 2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS), Tehran, Iran, 14–15 December 2016; pp. 1–6.
27. Huang, W. Design of Deep learning Image Classification and Recognition System Based on Zynq. Master's Thesis, Guangdong University of Technology, Guangzhou, China, 2018.
28. Liu, B.; Zou, D.; Feng, L.; Feng, S.; Fu, P.; Li, J. An FPGA-Based CNN Accelerator Integrating Depthwise Separable Convolu-tion. *Electronics* **2019**, *8*, 281. [CrossRef]
29. Zhang, S.; Cao, J.; Zhang, Q.; Zhang, Q.; Zhang, Y.; Wang, Y. An FPGA-Based Reconfigurable CNN Accelerator for YOLO. In Proceedings of the 2020 IEEE 3rd International Conference on Electronics Technology (ICET), Chengdu, China, 8–12 May 2020; pp. 74–78.
30. Xie, W.; Zhang, C.; Zhang, Y.; Hu, C.; Jiang, H.; Wang, Z. An Energy-Efficient FPGA-Based Embedded System for CNN Application. In Proceedings of the 2018 IEEE International Conference on Electron Devices and Solid State Circuits (EDSSC), Shenzhen, China, 6–8 June 2018; pp. 1–2.

31. Meloni, P.; Deriu, G.; Conti, F.; Loi, I.; Raffo, L.; Benini, L. A high-efficiency runtime reconfigurable IP for CNN acceleration on a mid-range all-programmable SoC. In Proceedings of the 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, Mexico, 30 November–2 December 2016; pp. 1–8.
32. Feng, G.; Hu, Z.; Chen, S.; Wu, F. Energy-efficient and high-throughput FPGA-based accelerator for Convolutional Neural Networks. In Proceedings of the 2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), Hangzhou, China, 25–28 October 2016; pp. 624–626.
33. Hailesellasie, M.; Hasan, S.R.; Khalid, F.; Wad, F.A.; Shafique, M. Fpga-based convolutional neural network architecture with reduced parameter requirements. In Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 27–30 May 2018; pp. 1–5.
34. Dong, R.; Tang, Y. Accelerator Implementation of Lenet-5 Convolution Neural Network Based on FPGA with HLS. In Proceedings of the 2019 3rd International Conference on Circuits, System and Simulation (ICCSS), Nanjing, China, 13–15 June 2019; pp. 64–67.
35. Shi, Y.; Gan, T.; Jiang, S.; Shi, Y.; Gan, T.; Jiang, S. Design of Parallel Acceleration Method of Convolutional Neural Network Based on FPGA. In Proceedings of the 2020 IEEE 5th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), Chengdu, China, 10–13 April 2020; pp. 133–137.
36. Maraoui, A.; Messaoud, S.; Bouaafia, S.; Ammari, A.C.; Khriji, L.; Machhout, M. PYNQ FPGA Hardware implementation of LeNet-5-Based Traffic Sign Recognition Application. In Proceedings of the 2021 18th International Multi-Conference on Systems, Signals & Devices (SSD), Monastir, Tunisia, 22–25 March 2021; pp. 1004–1009.
37. Liu, J.; Feng, J. Design of embedded digital image processing system based on ZYNQ. *Microprocess. Microsyst.* **2021**, *83*, 104005. [CrossRef]
38. AXI Reference Guide. Available online: https://docs.xilinx.com/v/u/en-US/ug761_axi_reference_guide (accessed on 19 May 2022).
39. Zynq UltraScale+ MPSoC Data Sheet: Overview. Available online: https://docs.xilinx.com/v/u/en-US/ds891-zynq-ultrascale-plus-overview (accessed on 19 May 2022).
40. He, B. *Xilinx FPGA Design Guide: Based on Vivado 2018 Integrated Development Environment*, 1st ed.; Publishing House of Electronics Industry: Beijing, China, 2018; pp. 274–422.
41. Vivado Design Suite Tutorial: Design Flows Overview. Available online: https://docs.xilinx.com/v/u/2019.1-English/ug888-vivado-design-flows-overview-tutorial (accessed on 19 May 2022).
42. PYNQ—Python productivity for Zynq—Home. Available online: http://www.pynq.io/ (accessed on 19 May 2022).