electronics

MDPI

Article

An Efficient FPGA-Based Implementation for Quantized Remote Sensing Image Scene Classification Network

Xiaoli Zhang, Xin Wei, Qianbo Sang, He Chen and Yizhuang Xie *

Beijing Key Laboratory of Embedded Real-time Information Processing Technology, Beijing Institute of Technology, Beijing 100081, China; zhangxl_me@126.com (X.Z.); weixin@bit.edu.cn (X.W.); sangqianbo@126.com (Q.S.); chenhe@bit.edu.cn (H.C.)

* Correspondence: xyz551_bit@bit.edu.cn; Tel.: +86-136-935-19576

Received: 11 July 2020; Accepted: 18 August 2020; Published: 20 August 2020

Abstract: Deep Convolutional Neural Network (DCNN)-based image scene classification models play an important role in a wide variety of remote sensing applications and achieve great success. However, the large-scale remote sensing images and the intensive computations make the deployment of these DCNN-based models on low-power processing systems (e.g., spaceborne or airborne) a challenging problem. To solve this problem, this paper proposes a high-performance Field-Programmable Gate Array (FPGA)-based DCNN accelerator by combining an efficient network compression scheme and reasonable hardware architecture. Firstly, this paper applies the network quantization to a high-accuracy remote sensing scene classification network, an improved oriented response network (IORN). The volume of the parameters and feature maps in the network is greatly reduced. Secondly, an efficient hardware architecture for network implementation is proposed. The architecture employs dual-channel Double Data Rate Synchronous Dynamic Random-Access Memory (DDR) access mode, rational on-chip data processing scheme and efficient processing engine design. Finally, we implement the quantized IORN (Q-IORN) with the proposed architecture on a Xilinx VC709 development board. The experimental results show that the proposed accelerator has 88.31% top-1 classification accuracy and achieves a throughput of 209.60 Giga-Operations Per Second (GOP/s) with a 6.32 W on-chip power consumption at 200 MHz. The comparison results with off-the-shelf devices and recent state-of-the-art implementations illustrate that the proposed accelerator has obvious advantages in terms of energy efficiency.

Keywords: remote sensing image; scene classification; accelerator; DCNN; FPGA; quantization

1. Introduction

Scene classification in remote sensing images refers to categorizing scene images into a discrete set of meaningful land use and land cover classes based on image content [1]. This useful remote sensing image processing task plays an essential role in natural disaster detection, vegetation mapping, urban planning and other applications [2–5]. Thus, it has become an important research topic in recent years. Drawing upon significant achievements in the field of computer vision, various deep convolution neural network (DCNN)-based methods have been proposed for remote sensing scene classification [6–9]. To employ efficient remote sensing scene classification in real-time, extensive researchers focus on building aerospace image processing systems, such as spaceborne or airborne systems. However, the large-scale remote sensing images and the intensive computations make the deployment of high-performance DCNN-based remote sensing image scene classification methods on power-limited real-time aerospace systems a challenging task.

To achieve this challenging task, many researchers are committed to building aerospace remote sensing image processing systems on low-power high-performance embedded devices. At present, extensive works adopt Central Processing Units (CPUs) and Graphics Processing Units (GPUs) as the implementation platforms of DCNNs for both training and inference phases. While CPUs and GPUs have high flexibility and excellent computing performance, their high power consumption and low performance–power ratio hinder their employment in power-limited systems. To achieve a trade-off between power consumption and computing performance, more attention has been paid to Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs) recently. Although FPGAs and ASICs hardly reach the same throughput as GPUs, their power consumption is limited. Furthermore, compared with FPGAs, the high cost and long development period of ASICs make it unable to keep up with the rapid changes of DCNNs. Thus, FPGA-based DCNN implementation has become a hotspot recently.

While FPGAs have relatively outstanding characteristics, the dense computations in DCNNs still cause difficulties for their deployment on FPGAs. Thus, optimizing DCNN models is of great significance. In recent years, many researchers have proposed various DCNN optimization methods to reduce network scale and computational complexity. In general, these optimization methods can be divided into three categories. The first category is to adopt computational transforms to reduce the volume of computation and the execution time of the network. Zhang et al. [10] used the Fast Fourier Transform (FFT) algorithm to optimize the convolution operation and achieved good performance in experiments. However, this method is effective only for large convolution kernels, but does little for small convolution kernels. Zhang et al. [11] adopted matrix multiplication to speed up the inference phase of the DCNN. However, the data replications in this method introduced a high memory cost. The second category is to reduce the amount of computation by cutting down the elements involved in DCNNs. Zhang et al. [12] effectively reduced the number of parameters in AlexNet by utilizing weight pruning and successfully deployed the compressed network on FPGA. However, due to not loading all calculations to the FPGA, their accelerator didn't achieve ideal calculation efficiency and good overall throughput. Mei et al. [13] compressed the network by applying low rank approximation to the full connection (FC) layer. However, since the computation of DCNNs is mainly concentrated in the convolution layers, their accelerator still occupied a lot of computing resources. The last category is to quantize the models to reduce the volume of parameters and calculations in DCNNs. Extensive research [14,15] has proved that the quantized model could reach similar performance as the floating-point one and could be well employed on FPGA platforms. Therefore, compared with the former two methods, the methods based on quantization schemes have received the most attention in the design of FPGA-based accelerators. Han et al. [16] quantized the weights of the convolutional layers and FC layers into 8-bit and 5-bit, respectively, to relieve the storage pressure. However, 5-bit precision is not effective for hardware implementation. Through comparative experiments, Gysel et al. [14] concluded that dynamic fixed-point (DFP) quantization had better performance than fixed-point quantization in the low-bit case. However, DFP quantization took up more on-chip storage compared with fixed-point quantization. Furthermore, many researchers have paid close attention to extreme bit compression [17,18]. Nevertheless, extreme bit compression reduces the network scale at the expense of network accuracy.

Although these optimization schemes provide significant convenience for the design of DCNN-based accelerators, the compressed DCNN still cannot be directly deployed on FPGA with limited resources. Aiming at this problem, the state-of-the-art implementations first advocate mapping a limited number of processing elements (PEs) on FPGA and reusing them during calculation. Then, to match the computing requirements of the PEs and obtain a high computing performance, these implementations also focus on configuring efficient storage schemes and data-paths when designing hardware architecture. To match the calculation amount with the memory bandwidth to a certain extent, Zhang et al. [19] designed a processing engine based on adder tree structure and theoretical roofline model. Similar to [19], Liu et al. [20] also applied the roofline model to coordinate

resource and bandwidth issues in their design. Alwani et al. [21] effectively reduced off-chip access by optimizing the order of input feature maps and fusing multiple adjacent convolutional layers. Sun et al. [22] improved accelerator performance by adopting different parallelism schemes for different convolutional layers and optimizing the data path with loop unrolling and tiling. Mei et al. [13] implemented the proposed hardware accelerator on a Xilinx VC709 evaluation board by employing dual-channel Double Data Rate Synchronous Dynamic Random-Access Memory (DDR), tile-grain on-chip feature map buffers and the proposed Propagate Partial Multiply-Accumulate processor. Li et al. [23] developed an efficient hardware architecture with the proposed neural processing units and hierarchical on-chip storage organization.

In this paper, we propose an efficient remote sensing image scene classification accelerator. We adopt the improved oriented response network (IORN) [24] as the baseline, in which the average active rotating filters (A-ARFs) can effectively reduce the parameters of the network. To facilitate the deployment on FPGA, we further quantize the IORN with a quantization-awareness training method [25], which reduces the model size and computation amount without accuracy loss. Then, based on the quantized network, an efficient hardware architecture is proposed. During the development of the hardware architecture, we mainly consider parallel design and data-path optimization. We arrange parallel computing units with appropriate scale and propose an efficient hardware processing engine to maintain high hardware efficiency. In addition, a reasonable data-path is developed based on data reuse, efficient dual-channel DDR access mode and rational on-chip data processing scheme. Finally, we implement the quantized IORN (Q-IORN) with the proposed architecture on FPGA. Compared with off-the-shelf devices and other advanced implementations, the proposed accelerator can strike a better trade-off between performance and power consumption. The main contributions of this paper are summarized as follows:

- A Q-IORN is proposed to facilitate the implementation of remote sensing scene classification on FPGA. The quantization-awareness training method used in this network can convert the feature maps and parameters of the network from floating-point to fixed-point, which efficiently compresses the model size without accuracy loss.
- We analyze and optimize each calculation module of the proposed Q-IORN, which lays a foundation for the subsequent hardware implementation.
- An efficient hardware architecture is proposed to implement the proposed Q-IORN. In this architecture, efficient dual-channel DDR access mode, rational on-chip data processing scheme and high-performance processing engine are adopted.
- We verify the proposed hardware architecture on a Xilinx VC709 development board and evaluate it on an NWPU-RESISC45 dataset. The experimental results show that the classification accuracy of the proposed accelerator is consistent with that of GPU, i.e., 88.31%. The proposed accelerator achieves an overall energy efficiency of 33.16 Giga-Operations Per Second Per Watt (GOP/s/W) at 200 MHz, which is superior to CPU, GPU and other advanced accelerators.

The rest of this paper is organized as follows. Section 2 introduces the framework of the proposed Q-IORN. In Section 3, we discuss the algorithm basics and the mapping optimization scheme of each layer in the network in detail. The proposed hardware accelerator is illustrated in Section 4. The experiments and results are shown in Section 5. Finally, the conclusions are presented in Section 6.

2. Quantized IORN

In this paper, IORN [24] was adopted as the basic network since it can classify the remote sensing scene images with high accuracy and is conducive to hardware implementation. Then, we optimized the original IORN with network quantization to create quantized IORN (Q-IORN). The detailed descriptions of IORN and Q-IORN will be discussed in the following sub-sections.

2.1. IORN

Recently, DCNN-based models have attracted the most attention among various remote sensing image scene classification methods. However, DCNNs are not fully applicable to remote sensing scene classification [26]. Different from natural scene images, remote sensing scene images show frequent orientation variations due to the rotation of the earth and changing shooting angles, which increases the difficulty of recognizing scene categories. Aiming at this problem, [24] proposed an efficient network, IORN, for remote sensing scene classification. IORN was proposed by adding an additional A-ARF module and Squeeze-ORAlign (S-ORAlign) module on fundamental network VGG16 [27], which has satisfactory learning ability with less training data for different remote sensing scenes. As a DCNN using A-ARFs, IORN applies the prior knowledge of rotation to the most basic unit of DCNNs (i.e., convolution calculation unit) rather than introducing additional functional modules or new network topologies. According to [24,28], A-ARFs in IORN actively rotate during the convolution process to generate the feature maps with explicit location and orientation encoding. Due to the existence of A-ARFs, IORN can produce the features of within-class rotation-invariant while maintaining inter-class discrimination for classification tasks, which effectively handles the directionality of remote sensing image scenes. Moreover, with A-ARFs, IORN requires significantly less convolutional parameters with a negligible computation overhead. According to the different rotation angles of A-ARFs, two kinds of IORN were proposed in [24]: IOR4-VGG16 and IOR8-VGG16. The detailed description of A-ARF is introduced as follows.

According to [24,28], the construction of A-ARF is based on the circular displacement property of Fourier Transform. In practical application, only a limited number of orientations is required to ensure the accuracy. An A-ARF with N orientation channels consists of two parts: a materialized filter Γ and $N-1$ immaterialized filters derived from its rotation. The n-th immaterialized filter, $n \in [1, N-1]$, is produced by anticlockwise rotating Γ by $2\pi n/N$. In this work, the $3 \times 3 \times N_{in} \times 4$ A-ARFs are mainly used, where N_{in} and 4 are the number of input feature maps' channels and filter orientation channels, respectively. Taking a $3 \times 3 \times 64$ materialized filter H as an example, the calculation process of its θ anticlockwise rotated version H_θ contains two steps. First, the result of coordinate rotation Ψ_θ can be obtained according to Equation (1):

$$\Psi^n_{\theta,\,i} = H^n_{(i+k)\ mod\ 8},\ i \in I,\ n = 0, 1, \cdots, N_{in} - 1 \tag{1}$$

where $N_{in} = 64$, $\theta = k\frac{2\pi}{8}$, and $I = \begin{pmatrix} 0 & 1 & 2 \\ 7 & & 3 \\ 6 & 5 & 4 \end{pmatrix}$ is a mapping for the coordinate index, as shown in Figure 1. After the coordinate rotation, the orientation rotation is performed by Equation (2) to produce the other three unmaterialized filters.

$$H^{m\,+\,Np}_\theta = \Psi^{((m\,-\,j)\ mod\ N)\,+\,Np}_\theta \tag{2}$$

where $m, j = 0, 1, \cdots, N-1, p = 0, 1, \cdots, \frac{N_{in}}{N} - 1, \theta = j\frac{2\pi}{N}$ and $N = 4$, as shown in Figure 2.

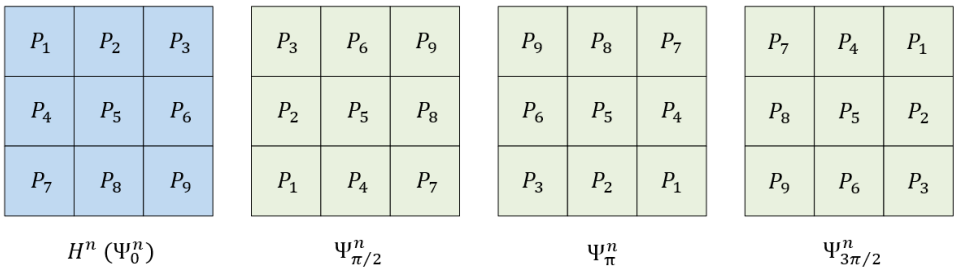

Figure 1. Materialized filter H in $3 \times 3 \times 64 \times 4$ average active rotating filter (A-ARF) anticlockwise rotates three times to produce the result of coordinate rotation.

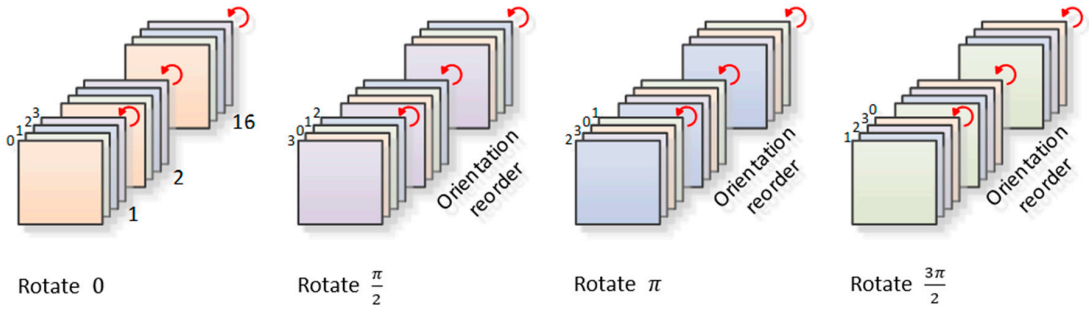

Figure 2. The process of generating H_θ by the orientation rotation of Ψ_θ.

2.2. Q-IORN

Since IOR4-VGG16 strikes a better trade-off between classification accuracy and the volume of parameters than IOR8-VGG16, this paper adopted the IOR4-VGG16 as the basic network for remote sensing scene classification. To facilitate hardware implementation, this paper further proposes a Q-IORN (i.e., Quantized IOR4-VGG16) based on the original IORN (i.e., IOR4-VGG16) with the following steps. First, the S-ORAlign layer is removed in the proposed Q-IORN, as it has no significant effects on performance and the reorder operation in this layer is not hardware-friendly. Second, a network quantization scheme is applied to the original IORN, where the A-ORConv layer (the convolutional layer with the A-ARFs) and FC layer in the original IORN are replaced by the quantized A-ORConv layer and quantized FC layer, respectively. With this quantization scheme, the weights and feature maps of the network are reduced by four times on the basis of the original IORN. Taking the A-ORConv layer and quantized A-ORConv layer as an example, the differences between them are shown in Figure 3. According to Figure 3, the quantized A-ORConv layer adopts more fixed-point arithmetic units with low bit-width than A-ORConv layer. The adopted network quantization scheme obtains the integer-only parameters through a quantization-awareness training method instead of directly quantizing the trained floating-point parameters. This can effectively reduce the network scale while maintaining high classification accuracy. Moreover, applying this method ensures the consistency of the network prediction between the training phase on a general computing platform and the inference phase on the proposed accelerator.

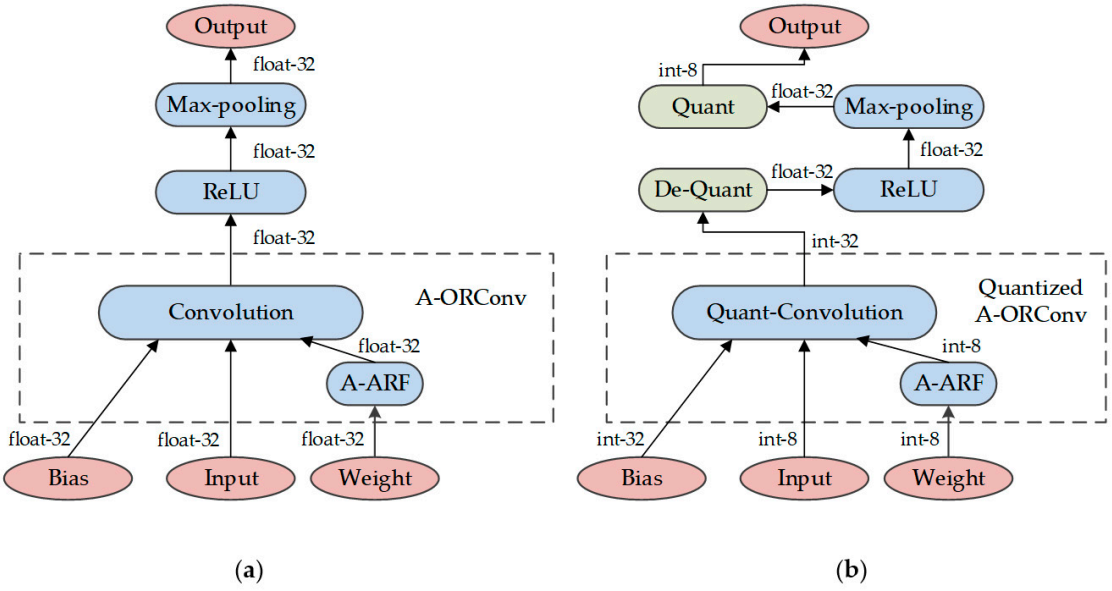

Figure 3. The comparison between the convolutional layer with the A-ARFs (A-ORConv) and quantized A-ORConv layer. (**a**) The calculation process of the A-ORCov layer; (**b**) The calculation process of the Quantized A-ORConv layer. 'Quant' and 'De-Quant' refer to 'quantized' and 'de-quantized', respectively.

3. Algorithm Basics and Mapping Optimization Scheme

The proposed Q-IORN is mainly composed of Quantized A-ORConv layers, activation function layers, pooling layers, Quantized FC layers and a Softmax layer. The algorithm basics and mapping scheme of each layer are discussed in this Section.

3.1. Quantized A-ORConv Layer

3.1.1. Network Quantization

Based on our previous work [25], an efficient quantization scheme is applied to IORN. With this scheme, the floating-point matrices of the network can be converted to integer matrices with low bit-width. The relationship between the two matrices is shown in Equation (3):

$$M_{float} = q \times M_{int} \tag{3}$$

where M_{float} indicates a floating-point matrix, M_{int} indicates an integer matrix, and q is quantization interval. Since 8-bit quantization is employed in this work, the quantization interval q in Equation (3) can be computed with:

$$q = \frac{\max\left(\left|M_{float}\right|\right)}{127} \tag{4}$$

From Equation (4), calculating the floating-point constant q first needs to find the maximum value of the floating-point matrix $|M_{float}|$. After network training, the weight matrix is determined. Thus, the quantization interval of the weights q_w can be easily computed by Equation (4). However, the feature map matrixes are different when different images are fed into the network. It requires large online computations to find the maximum element for different input images during the inference phase. To solve this problem, the quantization interval of the input feature maps q_f is obtained through the following two steps: estimating the largest absolute value during training, and calculating q_f based on the estimation. After the quantization interval is determined, the quantized integer matrix can be calculated by the following equation:

$$M_{int} = clamp\left(round\left(\frac{M_{float}}{q}\right)\right) \tag{5}$$

where q is the quantization interval, $round(\cdot)$ is used to find the nearest integer, and $clamp(\cdot)$ is used to limit the quantized elements to a specific range. For 8-bit quantization, the range is $[-127,\ 127]$.

3.1.2. Quantized A-ORConv

The A-ORConv layer is a custom convolutional layer, which replaces general filters with A-ARFs. Since an A-ARF contains N orientation channels, the output feature map $\widetilde{Z}$ calculated by it also contains N orientation channels, and the j-th channel can be obtained by Equation (6):

$$\widetilde{Z}^j = \sum_{n=0}^{N_{in}-1} H_\theta^n \times Z^n + Bias_j,\ \theta = j\frac{2\pi}{N},\ j = 0, 1, \cdots N \tag{6}$$

where H_θ is the j-th rotated version of the materialized filter H, Z is the input feature map, $Bias_j$ is the bias corresponding to H_θ, Z^n and H_θ^n are the n-th channel of Z and H_θ, respectively.

To reduce the consumption of storage resources and computing resources, an efficient quantization scheme is applied to both parameters and feature maps in the network. With Equations (3) and (6), the Quantized A-ORConv is defined as follows:

$$\widetilde{Z}^j = \sum_{n=0}^{N_{in}-1} \left(q_w H^n_{\theta int}\right) \times \left(q_f {Z^n}_{int}\right) + Bias_j \tag{7}$$

where q_w and q_f are quantization intervals of weights and input feature maps, respectively, $H^n_{\theta int}$ and ${Z^n}_{int}$ are quantized H_θ and quantized Z^n, respectively. Since q_w and q_f are both non-zero constants, the expression of $q_b = q_w q_f$ is applied to Equation (7). Thus, the Quantized A-ORConv can be written as:

$$\widetilde{Z}^j = q_b \left(\sum_{n=0}^{N_{in}-1} H^n_{\theta int} \times {Z^n}_{int} + \frac{Bias_j}{q_b} \right) \tag{8}$$

With this expression, the biases can be represented by integers. Moreover, the floating-point calculations of the A-ORConv layer are mainly converted to low-bit fixed-point calculations. This can greatly relieve the computational pressure when deploying the network on the proposed accelerator.

Based on the above-mentioned descriptions, the Quantized A-ORConv can be mapped on FPGA with the following steps. First, 8-bit quantization is performed on weights, biases and input feature maps by Equation (5). Second, only the materialized filters need to be stored in memory, while the immaterialized filters for convolution calculation are produced by logic. The mapping of coordinate rotation can be accomplished by simply exchanging register values. To implement orientation rotation, a suitable storage scheme is indispensable. Finally, the corresponding convolution calculations are completed by using the processed weights, biases and input feature maps. As fixed-point multiplication and fixed-point addition are used in convolution operations, it's necessary to pay attention to the variation of data bit-width during hardware deployment.

3.2. Activation Function

To enhance the nonlinear expression ability of DCNNs, various nonlinear activation functions have been widely used, including Sigmoid, Tanh, Rectified Linear Unit (ReLU) and so on. Among them, ReLU has drawn the most attention in recent times because it can efficiently alleviate gradient vanishing with low computational complexity. The neurons of the proposed Q-IORN are also activated by ReLU, as shown in Equation (9):

$$\widetilde{Z} = \begin{cases} 0, & Z < 0 \\ Z, & Z \geq 0 \end{cases} \tag{9}$$

where $\widetilde{Z}$ and Z in Equation (9) indicate the output and input of the activation function, respectively. According to Equation (9), the hardware implementation of ReLU can be completed by simply judging the symbol bit of input neurons.

3.3. Pooling Layer

The pooling layer in DCNNs is mainly used to reduce the scale of the features. Max-pooling and mean-pooling are two common pooling methods. In Q-IORN, max-pooling is employed as pooling layers. The output neuron $\widetilde{Z}$ of the 2×2 max-pooling layer can be calculated as:

$$\tilde{Z} = \max_{\substack{0 \leq x \leq 1 \\ 0 \leq y \leq 1}} \left(Z_{xy}\right) \tag{10}$$

where $Z_{xy} = \begin{pmatrix} Z_{00} & Z_{01} \\ Z_{10} & Z_{11} \end{pmatrix}$ is the neuron matrix in the pooling window. From Equation (10), the max-pooling operation can be regarded as finding the maximum value from the pooling window, which can be achieved by hardware comparators.

3.4. Quantized FC Layer

In classification networks, FC layers are generally used to integrate the features of the convolutional layers. Each output neuron of the FC layer is related to all input neurons, and the j-th output neuron $\widetilde{Z_J}$ can be represented as:

$$\widetilde{Z_J} = \sum_{i=1}^{N} Z_i \times W_{ij} + Bias_j \tag{11}$$

where N is the number of input neurons, Z_i is the i-th input neuron, W_{ij} is the weight corresponding to Z_i when generating the j-th output neuron, and $Bias_j$ is the bias for the j-th output neuron. Since the quantization scheme is applied to the FC layer, the j-th output neuron of the quantized FC layer is defined as:

$$\widetilde{Z_J} = q_b \left(\sum_{i=1}^{N} Z_{inti} \times W_{intij} + \frac{Bias_j}{q_b} \right) \tag{12}$$

where $q_b = q_w q_f$, q_w and q_f are quantization intervals of weights and input neurons, respectively, Z_{inti} and W_{intij} are quantized Z_i and quantized W_{ij}, respectively. Similar to the quantized convolutional layers, the mapping of the quantized FC layers also relies on fixed-point multiplication and fixed-point addition. Moreover, the volume of parameters in the FC layers is too large to be stored in on-chip memory. Thus, these parameters need to be stored in external memory when deploying the accelerator.

3.5. Softmax Layer

The Softmax layer is mostly applied in multi-classification networks, which converts the output neurons of the last FC layer to probabilities. The Softmax layer is defined as follows:

$$\widetilde{Z_J} = \frac{e^{Z_j}}{\sum_{i=1}^{N} e^{Z_i}} \ (j = 1,\ 2, \cdots, N) \tag{13}$$

where N is the number of output neurons of the last FC layer, $\widetilde{Z_J}$ and Z_j are the j-th output of the last FC layer and Softmax layer, respectively. As shown in Equation (13), the Softmax layer contains exponential operations and division operations. These operations are unfriendly to hardware implementation. Thus, the Softmax layer can be replaced by a simple maximum operation. This can reduce the logical resource consumption while obtaining the correct classification prediction.

4. Hardware Implementation

Based on the detailed algorithm description in Section 3, an efficient hardware accelerator for classifying remote sensing image scene is proposed. The architecture of the proposed hardware accelerator is shown in Figure 4. In the proposed accelerator, we adopt the Advanced RISC Machine (ARM) as a processing system (PS). The PS is used to load the trained model and original images to programmable logic (PL), in addition to recording image classification results. The PL is composed of an FPGA chip and independent dual-channel DDR. To ensure low power consumption and short time overhead, the FPGA chip is utilized to implement and accelerate all calculations of the Q-IORN. Due to the limited on-chip memory, the dual-channel DDR is adopted to store the trained model and the output feature map of each layer. As illustrated in Figure 4, PL is the main part of the proposed hardware accelerator. To employ efficient network calculations on PL, two efficient storage schemes and a high-performance processing engine are proposed. The details are described in the following sub-sections.

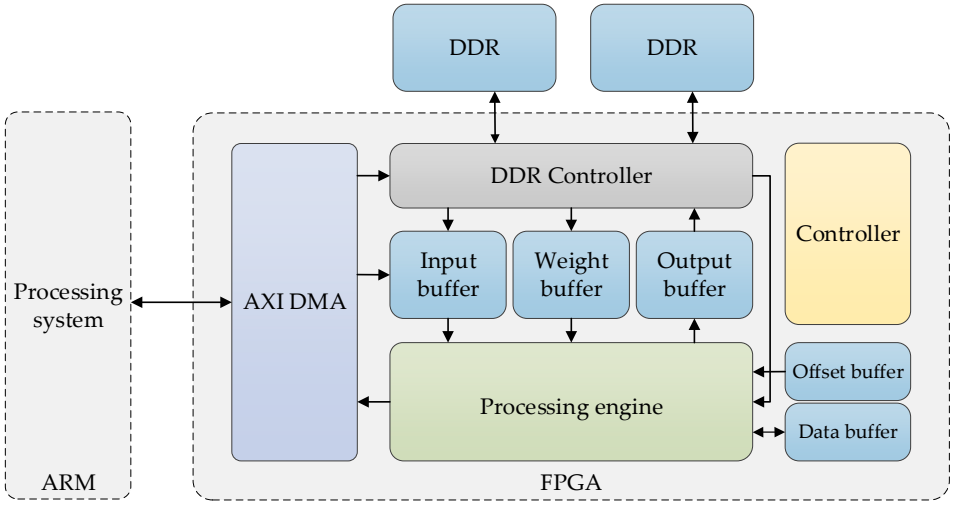

Figure 4. The overall architecture of the proposed hardware accelerator.

4.1. Efficient Storage Scheme

In our accelerator, both off-chip memory and on-chip memory are adopted to store weights and features. The external memory directly connected with PL is utilized to save the model parameters and the output features of each layer. The on-chip memory with low power consumption and fast access speed is used to cache a part of the input data and intermediate results. The on-chip memory is divided into several independent sub-storage units, including input buffer, weight buffer, offset buffer, output buffer and data buffer (for storing intermediate results). The depth of each buffer is configured independently according to the computing requirements. In addition, we propose two efficient data storage schemes to significantly reduce the occupation of storage resources and the time overhead caused by data interaction.

Firstly, a reasonable layer-wise storage scheme is proposed to take advantage of the independent dual-channel DDR. This scheme could avoid the conflict between the storage of the output feature map and the access of the input feature map in each layer by using the ping-pong buffer technique. Moreover, the scheme also reasonably arranges the storage and access of network weights according to the idle time of dual-channel DDR. The working state of the dual-channel DDR is changed alternately by using DDR controller, as shown in Figure 5a. For the n-th layer, the output features of the previous layer and the weights of the current layer are read from DDR3-B and DDR3-A, respectively. DDR3-A is also used to receive the output features of the current layer. Regarding the $(n + 1)$-th layer, the working mode is exchanged between DDR3-A and DDR3-B. This can greatly reduce the time overhead between layers and ensure the successful implementation of the full pipeline within layer. The storage space of the dual-channel DDR is also reasonably arranged. As shown in Figure 5b, the weights and output features of the odd-numbered convolutional layers are stored in DDR3-A, while the corresponding data of the even-numbered convolutional layers and the weights of the FC layers are stored in DDR3-B. Moreover, the storage space occupied by the output features is reused in this scheme, which helps to reduce the resource consumption caused by additional logical control.

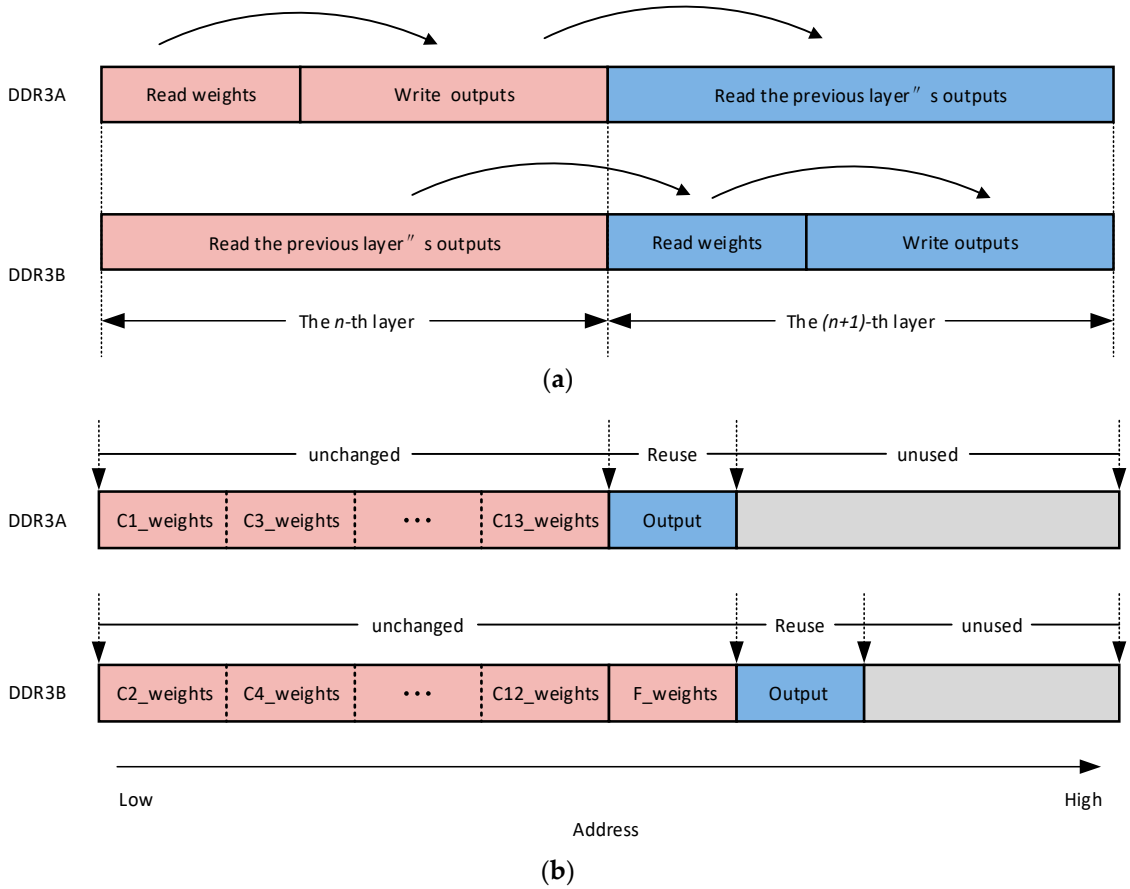

Figure 5. The arrangement of the independent dual-channel Double Data Rate Synchronous Dynamic Random-Access Memory (DDR). (**a**) The working state of the dual-channel DDR; (**b**) The address arrangement of the dual-channel DDR. Ci_weights ($i = 1, 2, \cdots, 13$) and F_weights refer to the weights of the i-th convolutional layer and the full connection (FC) layers, respectively.

The second efficient storage scheme is proposed based on the data bit-width relationship among input buffer, output buffer and external memory. The proposed data interaction scheme transfers the output matrix through the steps shown in Figure 6. Since the output of the n-th layer in DCNNs serves as the input of the $(n + 1)$-th layer, we first arrange the n-th layer's output elements in the order of the input elements required by the $(n + 1)$-th layer. The order is displayed in Figure 6a. Then, in the n-th calculation layer, we store the spliced 512-bit data a_i in the output buffer instead of the 8-bit output element $c_{i,j}$. As shown in Figure 6b, a_i is generated by splicing the elements of the i-th column in Figure 6a. By this means, we effectively reduce the storage resource consumption and match the data bit-width of DDR. Next, in the $(n + 1)$-th calculation layer, the a_i read from DDR is stored in the input buffer. The relationship between the 32-bit output element $b_{i,k}$ and the 512-bit input element a_i in the input buffer is presented in Figure 6c. Finally, the element $b_{i,k}$ read from input buffer is grouped and reorganized based on the computing requirements. An example of the data rearrangement process is revealed in Figure 6d. Evidently, the order of the obtained input elements in Figure 6d is consistent with that of the elements in Figure 6a. With this scheme, we successfully meet the requirements of pipeline implementation within layer and significantly reduce the storage resource consumption. Moreover, the contradiction between calculation speed and data storage speed of the first layer in the network is eliminated.

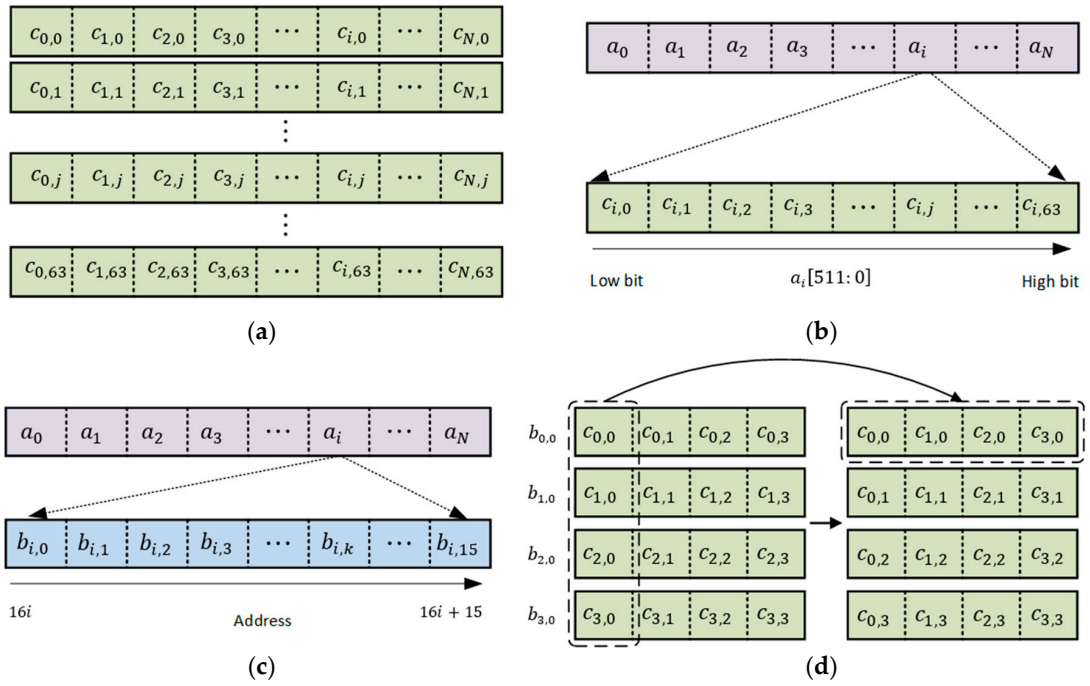

Figure 6. An efficient on-chip storage scheme. (**a**) The order of the $(n + 1)$-th layer's input elements (or the n-th layer's output elements). $c_{i,j}$ is the element generated by the j-th calculation unit and stored in the i-th address of the output buffer. N is the maximum storage address of the output elements; (**b**) The composition of a_i (i.e., the n-th layer's DDR input element or the $(n + 1)$-th layer's DDR output element); (**c**) The relationship of the output element $b_{i,k}$ and the input element a_i in the input buffer. $b_{i,k}$ is the element stored in the $(16i + k)$-th output address of the input buffer; (**d**) The process of obtaining the required input element order.

4.2. Processing Engine Architecture

To implement the proposed Q-IORN efficiently, a hardware processing engine architecture is proposed, as shown in Figure 7. The proposed hardware processing engine is mainly composed of a convolutional processing engine and an FC processing engine. The former achieves the calculation for the Quantized A-ORConv layer, ReLU layer and max-pooling layer, and the latter is designed to calculate the Quantized FC layer, ReLU layer and Softmax layer. The implementation of the key modules in the hardware processing engine is described as follows.

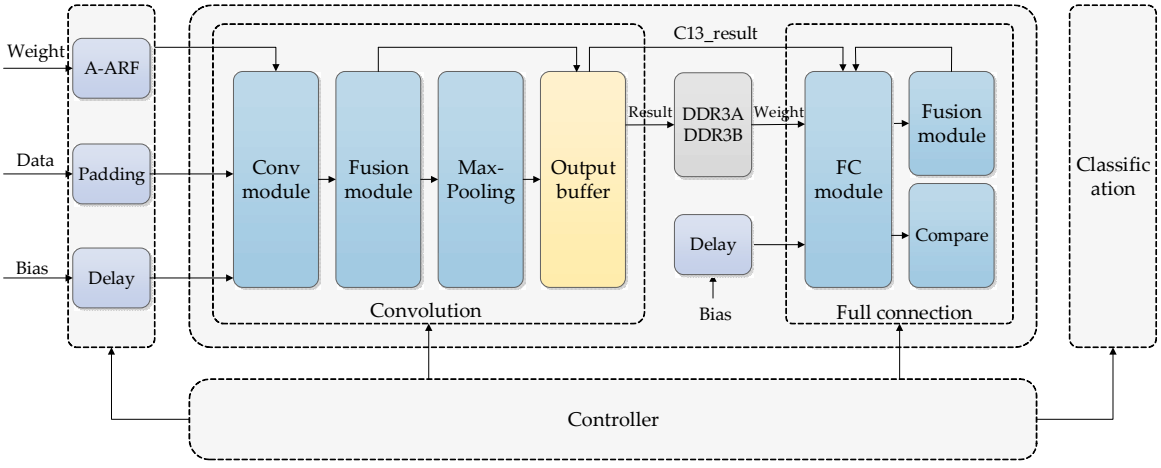

Figure 7. The hardware processing engine architecture. Conv module refers to convolution module. C13_result refers to the output of the last convolutional layer.

(1) A-ARF generation module. As shown in Section 3.1, the unmaterialized filters of A-ARF are produced by coordinate rotation and orientation rotation. The mapping order of coordinate rotation and orientation rotation is exchanged in the proposed hardware accelerator. Specifically, the orientation rotation is implemented before the coordinate rotation. This can reduce the logical resource consumption while generating the required unmaterialized filters. Figure 8 depicts the procedure of generating the m-th input channel of A-ARF. The materialized filter H is stored with a custom data arrangement, as shown in Figure 8. In this case, four adjacent channels (i.e., $\{H_{4i-3}, H_{4i-2}, H_{4i-1}, H_{4i}\}$) of the filter H required for orientation rotation can be provided simultaneously. The orientation rotation module reads the i-th column in the storage unit and reorders it according to the value of the counters. For example, when $j = 1$, the sequence is converted from $\{H_{4i-3}, H_{4i-2}, H_{4i-1}, H_{4i}\}$ to $\{H_{4i-3}, H_{4i}, H_{4i-1}, H_{4i-2}\}$ (defined as F_1), and H_i represents the i-th input channel of the materialized filter H. After the orientation rotation, the m-th input channel of A-ARF is finally generated by rotating the coordinates of each kernel in F_j separately. The detailed process of the coordinate rotation is shown in Figure 1. In conclusion, the implementation of A-ARF mainly depends on reasonable storage arrangement and efficient logical control. The proposed scheme can provide the convolution kernels required for convolution calculation in real-time, and relieve the on-chip storage pressure caused by storing immaterialized filters.

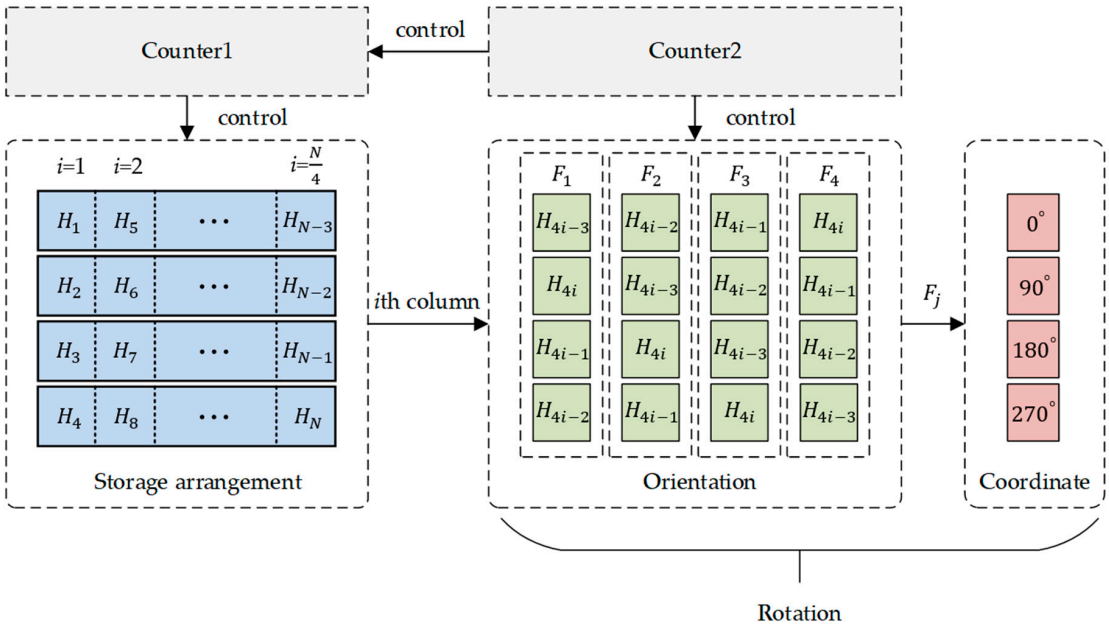

Figure 8. The process of generating the m-th input channel of A-ARF. $m = 4(i-1) + j$, i and j are controlled by counters. N is the number of the convolution kernel's input channels.

(2) Feature padding module. Unlike the implementation in [29], which used additional Block Random Access Memory (BRAM) to store padding data, the proposed hardware accelerator achieves the padding operation during the calculation by judging the position of the current calculation patch in the feature map. With this optimization, only a few register resources are required for mapping this module.

(3) Convolution calculation module. During the deployment of the convolution calculation module, the main challenge is to map the dense multiplication and addition operations to FPGA. Thus, we focus on exploring a suitable parallel computing scheme. In the proposed convolution calculation module, a total of 64 PEs are placed. Each PE with nine multipliers adopts an adder tree structure to carry out the calculation in a 3×3 window. Based on the network structure and the proposed parallel scheme, our accelerator reads the three consecutive rows of the input feature map (with a

size of $3 \times N_W \times N_{if}$) and the required convolution kernel (with a size of $3 \times 3 \times N_{if} \times N_{of}$) to the corresponding on-chip buffer each time. Then a row of the output feature map (with a size of $1 \times N_W \times N_{of}$) can be generated according to the obtained input feature map and convolution kernel, as shown in Figure 9. The detailed calculation process for generating the m-th row of the output feature map is illustrated in Figure 10. After obtaining the required input feature map and convolution kernel, we divide them into N_{if} groups. The i-th group contains three rows of the i-th input channel in the input feature map (with a size of $3 \times N_W \times 1$) and the i-th channel of the convolution kernel (with a size of $3 \times 3 \times 1 \times N_{of}$). This operation provides a guarantee for the realization of pipelining within layer. As 64 PEs are set in this module, the convolution kernels in the i-th group are further grouped by 64 (i.e., the kernels in each group are further divided into $\frac{N_{of}}{64}$ units, and each unit contains 64 3×3 windows), as shown in Figure 10. After that, the final outputs are produced by combining data reuse, vector inner product within group and accumulation between groups.

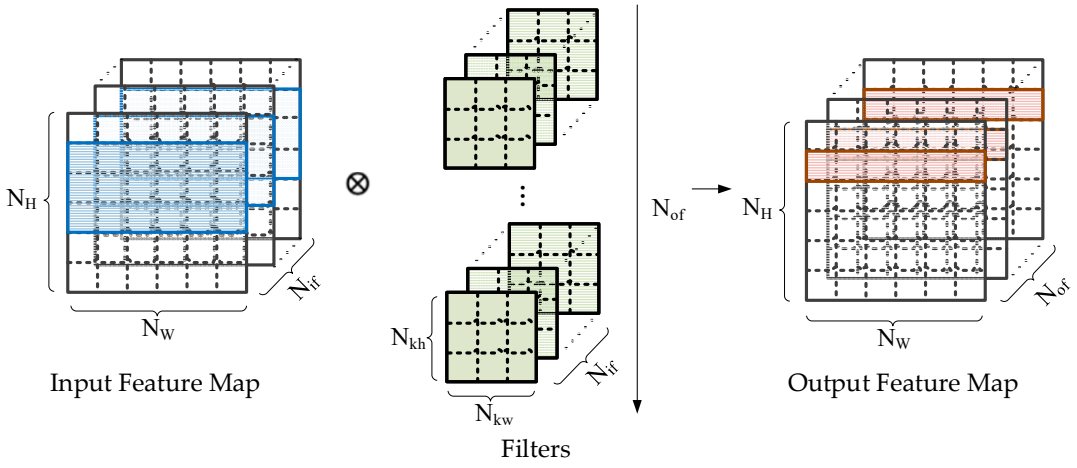

Figure 9. The calculation process of generating a row of the output feature map. N_{if} and N_{of} represent the input channels and output channels, respectively. N_W and N_H are the width and height of the feature map, respectively. $N_{kw} = 3$ and $N_{kh} = 3$ are the width and height of the filters, respectively.

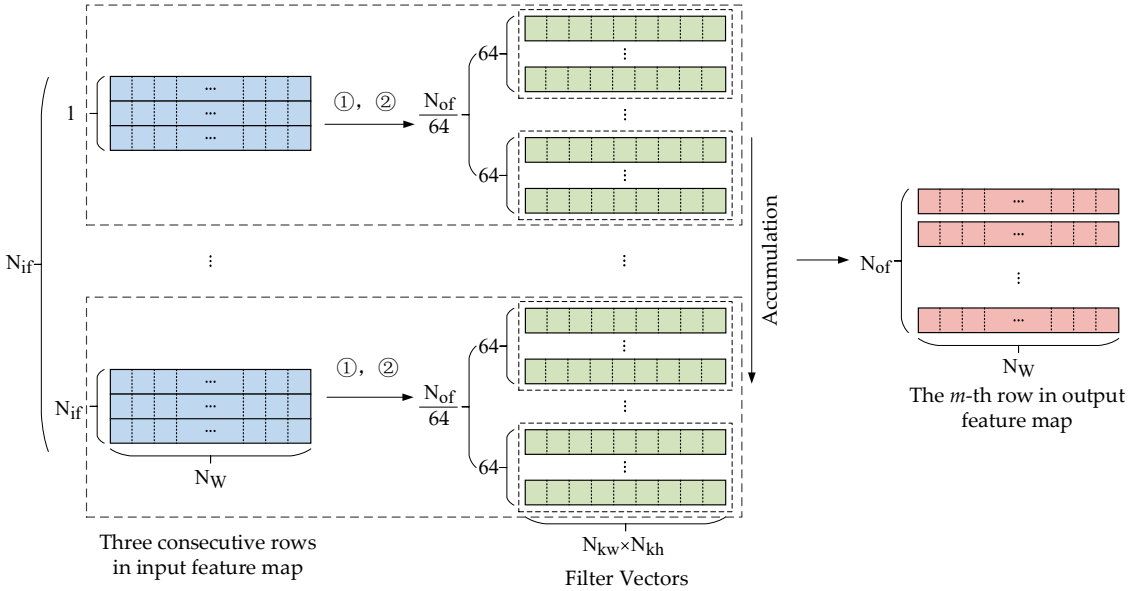

Figure 10. The detailed calculation process for generating the m-th row of the output feature map in each convolutional layer. ① refers to vectorization. ② is vector inner product.

(4) FC calculation module. The mapping of the FC calculation module is mainly limited by the access of weights. Since DDR can only provide a 512-bit data per clock cycle, we set up a PE with 64 multipliers for FC calculation. With this scheme, we can make full use of the bandwidth of DDR. The detailed calculation process of the FC layer is shown in Figure 11. To match the bit-width of the DDR, the input neurons and weights are processed into a custom form through flattening and splicing (the input neurons are processed online, and the weights are processed offline). In this way, the input neurons and corresponding weights are divided into $\frac{N_i}{64}$ groups, and each group has a 512-bit input and N_o 512-bit weights. The input neurons are reused during the calculation process, which effectively reduces the access to on-chip storage. The weights are read from the external memory in real-time for calculation. Similar to convolution calculation module, the final output neurons are also generated by applying the vector inner product within group and the accumulation between groups.

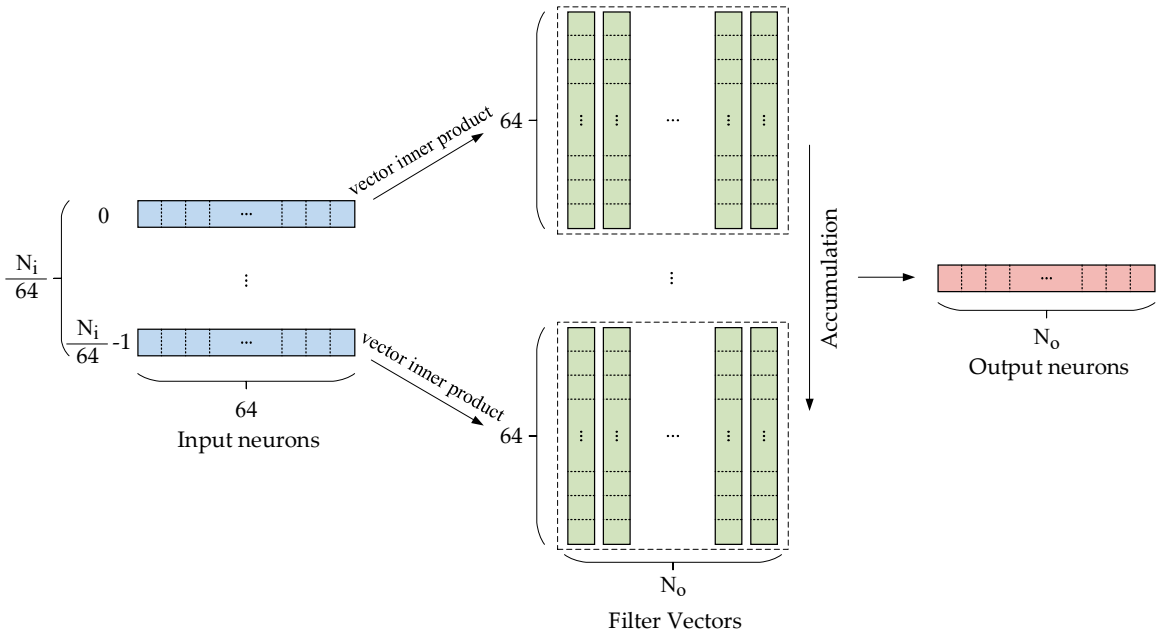

Figure 11. The calculation process of each FC layer. N_i and N_o are the number of the input neurons and output neurons, respectively.

(5) Fusion calculation module. Based on our previous work [25], we propose a fused layer to apply the quantization operation to the proposed processing engine. The proposed fused layer is obtained by merging the n-th de-quantized layer and the $(n + 1)$-th quantized layer and is implemented before ReLU. From Equations (3) and (5), the implementation of the de-quantization and quantization rely on the multiplication operation and division operation, respectively. While for fused layer, only multiplication operation is required, which can reduce the consumption of computing resources. Figure 12 depicts the process of producing the $(n + 1)$-th convolutional layer's input element $e_{0,0}$ in general network inference and optimized network inference. As shown in Figure 12, this optimized implementation scheme can also optimize the input bit-width of the ReLU module and max-pooling module from 32-bit to 8-bit, which effectively reduces the consumption of storage resources.

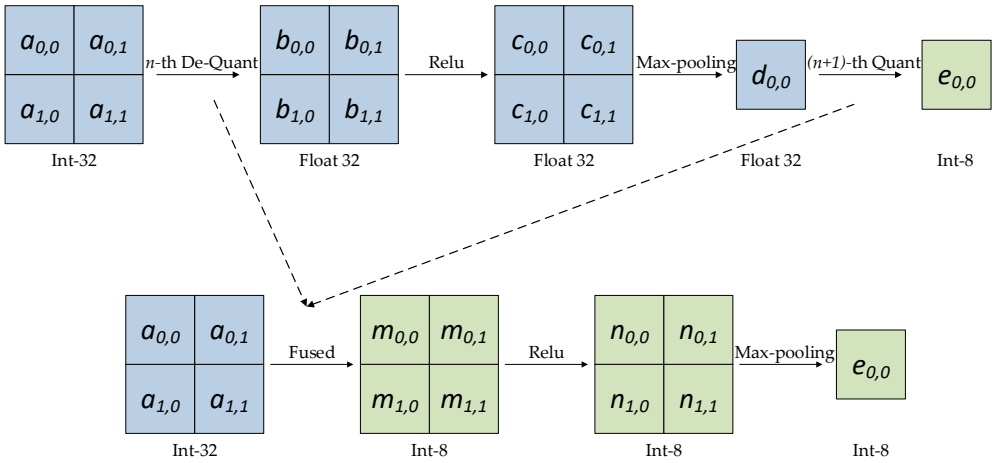

Figure 12. The process of generating element $e_{0,0}$ in general network inference and optimized network inference. $a_{i,j}$ is the output of the convolution module. $b_{i,j}$, $c_{i,j}$, and $d_{i,j}$ are the outputs of the de-quantization module (the n-th layer), ReLU module and Max-pooling module in general network inference, respectively. $m_{i,j}$ and $n_{i,j}$ are the outputs of the fusion module and ReLU module in optimized network inference, respectively. 'Quant' and 'De-Quant' refer to 'quantized' and 'de-quantized', respectively.

5. Experiments and Results

In this section, extensive experiments were conducted to evaluate the performance of the proposed Q-IORN and hardware accelerator. The evaluation experiments were divided into two parts. First, the proposed Q-IORN was trained and tested on a publicly available remote sensing image scene dataset to evaluate its classification accuracy and obtain the integer-only model for FPGA implementation. Then, we implemented the proposed hardware accelerator on FPGA and tested its processing performance. The experimental settings and detailed experimental results are provided below.

5.1. Experimental Settings

5.1.1. Dataset Description

The proposed Q-IORN and hardware accelerator were evaluated on the NWPU-RESISC45 dataset [1], which is a large-scale remote sensing scene classification dataset. This dataset covers 45 scene classes and contains 700 images per class. The images in the dataset have a uniform size of 256 × 256. Sample images of each class are shown in Figure 13. In this work, we randomly selected 10% of the images in the NWPU-RESISC45 dataset for training, and the rest were utilized for testing. Data augmentation tricks, including random cropping, random horizontal flip and normalization, were applied in training phase. As for the testing phase, we just applied center cropping and normalization. The size of the cropped image was 224 × 224.

Figure 13. Sample images of each class in NWPU-RESISC45 dataset.

5.1.2. Experimental Procedure

To test the performance of the proposed algorithm and obtain the integer-only model for FPGA implementation, we trained the Q-IORN with the quantization-awareness training method on a NVIDIA Titan Xp GPU with PyTorch 0.4. The 8-bit quantized network Q-IORN was trained for 200 epochs with a starting learning rate of 0.002. The learning rate was divided by 0.5 at every 50 epochs. The weight parameters were trained by a stochastic gradient descent (SGD) optimizer with a weight decay of 5×10^{-4} and a momentum of 0.9. The batch size was set to 64.

To verify the performance of the proposed hardware accelerator, we described the proposed hardware architecture with Very-High-Speed Integrated Circuit Hardware Description Language (VHDL) and synthesized it with Xilinx Vivado 2017.2. The Xilinx VC709 development board with Xilinx XC7VX690T FPGA and two DDR3 memory modules was utilized as the implementation platform. In the proposed Q-IORN, the channel numbers of the feature map in each layer are multiples of 64, and the maximum feature size is 224×224. Thus, the main modules of the convolutional processing engine were configured as follows: the convolution calculation module placed 64 convolution PEs, the input buffer was comprised of six BRAMs (three BRAMs with a depth of 512 were used to cache original image, the rest with a depth of 224 were utilized to cache input feature maps), the weight buffer was composed of 64 BRAMs with a depth of 1024, and the output buffer consisted of a BRAM with a depth of 224. The maximum number of output neurons in the FC layers is 4096. Thus, a FC PE and a data buffer composed of a First Input First Output (FIFO) memory with a depth of 4096 were employed in the FC processing engine. The performance of the proposed hardware accelerator was compared with CPU, GPU and several recent advanced FPGA-based implementations.

5.2. Performance Evaluation of the Q-IORN

In this section, we aimed to evaluate the performance of the proposed Q-IORN. We took the original IORN as a baseline and the comparison results are summarized in Table 1. As illustrated in

Table 1, the classification accuracy of the proposed Q-IORN is 88.31%. Compared with the baseline, the Q-IORN has almost no classification accuracy loss. The result shows that symmetric quantization and S-ORAlign operation have negligible effects on classification accuracy. Moreover, the model size of the proposed Q-IORN is almost 4× smaller than that of the baseline. In general, the proposed Q-IORN can efficiently reduce the scale of the network while maintaining approximate classification accuracy.

Table 1. The classification accuracy of the proposed quantized improved oriented response network (Q-IORN) and baseline.

Network.	Accuracy	Model Size (MB)
IORN	88.49%	485.88
Q-IORN	88.31%	121.51

5.3. Performance Evaluation of the Proposed Accelerator

We deployed the proposed hardware architecture on FPGA to demonstrate its effectiveness. In the proposed architecture, Digital Signal Processing (DSP) slices were used to implement multiplication operations, including fixed-point multiplication and floating-point multiplication. The detailed occupation of DSPs provided by Vivado design suite is shown in the Table 2. In our accelerator, a total of 65 floating-point multipliers were used by the fusion calculation module, and each floating-point multiplier consisted of two DSPs. A total of 640 fixed-point multipliers were adopted by convolutional processing engine and FC processing engine, and each fixed-point multiplier was composed of one DSP. Thus, a total of $65 \times 2 + 640 \times 1 = 770$ DSPs were used in our accelerator, which is consistent with the DSP utilization shown in Table 2. The Look-Up Tables (LUTs) were used by control units, buffer units, calculation units, data splicing units and so on. The BRAMs were mainly utilized to store offsets, weights and feature maps.

Table 2. The details of Digital Signal Processing (DSP) utilization in our architecture.

Multipliers Type	Floating-Point Multipliers	Fixed-Point Multipliers
Multipliers Number	65	640
DSP Utilization	130	640

The hardware resource utilization of the proposed accelerator is summarized in Table 3. As shown in Table 3, the utilization of DSP, BRAM, Flip Flop (FF), and LUT is 770, 404.5, 116,742 and 73,320, respectively. The utilization percent of each resource is less than 30%. These results illustrate that the proposed hardware architecture can be implemented on the platform with limited resources.

Table 3. Resource utilization of the proposed hardware accelerator.

Resource	DSP	BRAM	FF	LUT
Available	3600	1470	866,400	433,200
Utilization	770	404.5	116,742	73,320
Utilization (%)	21.39	27.52	13.47	16.93

We took the off-the-shelf platforms CPU and GPU for comparison to show the effectiveness of the proposed hardware accelerator. The CPU platform refers to an Intel Xeon E5-2697 v4 CPU @ 2.3 GHz with 224 GB DDR4 DRAM. The GPU platform is a NVIDIA TITAN Xp GPU with 12 GB GDDR5X memory. The performance comparison of CPU, GPU and the proposed hardware accelerator is shown in Table 4. The 'GOP/s' in Table 4 is the abbreviation of Giga-Operations Per Second. Based on the characteristics of data types in Q-IORN, fixed-point arithmetic units were mainly adopted on these platforms when mapping the network. Since the quantization-awareness training method applied in

this paper could effectively simulate the hardware behavior, the proposed accelerator has the same classification accuracy as the CPU and GPU. The Thermal Design Power (TDP) values of the CPU and GPU are 145 W and 250 W, respectively. The power report supplied by Vivado design suite shows that the total power of the proposed accelerator is only 6.32 W. As shown in Table 4, GPU has obvious advantages in terms of throughput, which is 26.49 times that of CPU and 3.44 times that of the proposed accelerator, correspondingly. The proposed accelerator achieves the best energy efficiency performance among all platforms. Compared with CPU and GPU, the energy efficiency of our accelerator is 174.53 times and 11.51 times higher, respectively.

Table 4. Evaluation results on the Central Processing Unit (CPU), Graphic Processing Unit (GPU) and our accelerator.

Platform	CPU	GPU	FPGA
Vendor	Intel Xeon E5-2697 v4	NVIDIA TITAN Xp	Xilinx VC709
Technology (nm)	14	16	28
Frequency (MHz)	2300	1582	200
Power (W)	145	250	6.32
Latency (ms)	1137.62	43.00	147.62
System Performance (fps)	0.88	23.26	6.77
Throughput (GOP/s)	27.20	720.55	209.60
Energy Efficiency (GOP/s/W)	0.19	2.88	33.16

The recent FPGA-based implementations were also compared in this section. The comparison results of the proposed hardware accelerator and several recent FPGA-based implementations are presented in Table 5. The 'conv' and 'all' in Table 5 refer to the performance of the convolutional layers and the overall system, respectively. The experimental results show that the overall throughput of the proposed accelerator reaches 209.60 GOP/s at 200 MHz, and the throughput of the convolutional layers achieves a better performance of 224.43 GOP/s. Since the performance of FPGA-based implementation is closely related to the utilization of on-chip resources, it is partial to evaluate the performance only by throughput. Therefore, the energy efficiency indicator of different FPGA-based implementations was also taken into consideration in this paper. As shown in Table 5, the energy efficiency of the proposed accelerator is 33.16 GOP/s/W. Compared with the previous works, the proposed hardware accelerator achieves a higher energy efficiency. In general, the proposed hardware accelerator could strike a better trade-off between performance and power consumption than previous FPGA-based implementations.

Table 5. Comparisons with previous implementations.

	[30]	[13]	[12]	[23]	Ours
FPGA	Arria-10 GX 1150	Xilinx XC7VX690T	Xilinx XCZU7EV	Xilinx XC7Z100	Xilinx XC7VX690T
Frequency (MHz)	150	200	300	200	200
Precision	8–16 bit fixed	16-bit float	8-bit fixed	16-bit fixed	8-bit fixed
Power (W)	21.2	10.81	17.67	19.52	6.32
Throughput (GOP/s)	645.25	202.03 (conv) 202.42 (all)	290.40 (conv) 14.11 (all)	452.8	224.43 (conv) 209.60 (all)
Energy efficiency (GOP/s/W)	30.44	18.69 (conv) 18.73 (all)	16.44 (conv) 0.80 (all)	23.20	35.51 (conv) 33.16 (all)

6. Conclusions

In this paper, we developed a DCNN accelerator for classifying remote sensing scene images under power-limited conditions. First of all, we combined the A-ARF algorithm with the symmetric quantization algorithm to compress the DCNN model. With this scheme, the parameters, feature maps and computation amount of the network are greatly reduced. Moreover, the proposed Q-IORN has almost no classification accuracy loss compared with the IOR4-VGG16 network. Then, an efficient

hardware accelerator was proposed for Q-IORN. In the proposed accelerator, several reasonable storage schemes were presented to reduce the storage resource occupation and time overhead based on the characteristics of off-chip memory and on-chip buffer. Two processing engines with data reuse and computing parallelism were adopted to ensure the high-performance computing requirement of the convolutional layers and FC layers. The proposed accelerator was implemented on a Xilinx VC709 development board. The experimental results show that the throughput and energy efficiency of the proposed accelerator are 209.60 GOP/s and 33.16 GOP/s/W, respectively. Compared with CPU and GPU platforms, the proposed accelerator improves energy efficiency by 174.53 times and 11.51 times, respectively. Several recent advanced FPGA-based implementations were also compared to verify the superiority of the proposed hardware accelerator.

Author Contributions: Conceptualization, X.Z. and X.W.; methodology, X.Z. and X.W.; software, X.Z., X.W. and Q.S.; validation, X.Z. and Q.S.; formal analysis, X.W.; investigation, X.Z. and X.W.; resources, H.C. and Y.X.; writing—original draft preparation, X.Z.; writing—review and editing, X.Z., X.W. and Q.S.; supervision, Y.X.; project administration, H.C.; funding acquisition, H.C. and Y.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key R & D Program of China under Grant No. 2017YFB0502800 and Grant No. 2017YFB0502804, the MYHT Program of China under Grant No. B0201.

Acknowledgments: This work was supported by the Chang Jiang Scholars Program under Grant T2012122 and the Hundred Leading Talent Project of Beijing Science and Technology under Grant Z141101001514005.

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

References

1. Cheng, G.; Han, J.; Lu, X. Remote sensing image scene classification: Benchmark and state of the art. *Proc. IEEE* **2017**, *105*, 1865–1883. [CrossRef]
2. Cheng, G.; Guo, L.; Zhao, T.; Han, J.; Li, H.; Fang, J. Automatic landslide detection from remote-sensing imagery using a scene classification method based on BoVW and pLSA. *Int. J. Remote Sens.* **2013**, *34*, 45–59. [CrossRef]
3. Mishra, N.B.; Crews, K.A. Mapping vegetation morphology types in a dry savanna ecosystem: Integrating hierarchical object-based image analysis with random forest. *Int. J. Remote Sens.* **2014**, *35*, 1175–1198. [CrossRef]
4. Cheng, G.; Han, J.; Guo, L.; Liu, Z.; Bu, S.; Ren, J. Effective and efficient midlevel visual elements-oriented land-use classification using VHR remote sensing images. *IEEE Trans. Geosci. Remote Sens.* **2015**, *53*, 4238–4249. [CrossRef]
5. Xia, G.-S.; Hu, J.; Hu, F.; Shi, B.; Bai, X.; Zhong, Y.; Zhang, L.; Lu, X. AID: A benchmark data set for performance evaluation of aerial scene classification. *IEEE Trans. Geosci. Remote Sens.* **2017**, *55*, 3965–3981. [CrossRef]
6. Hu, F.; Xia, G.-S.; Hu, J.; Zhang, L. Transferring Deep Convolutional Neural Networks for the Scene Classification of High-Resolution Remote Sensing Imagery. *Remote Sens.* **2015**, *7*, 14680–14707. [CrossRef]
7. Zhong, Y.; Fei, F.; Zhang, L. Large patch convolutional neural networks for the scene classification of high spatial resolution imagery. *J. Appl. Remote Sens.* **2016**, *10*, 025006. [CrossRef]
8. Cheng, G.; Li, Z.; Yao, X.; Guo, L.; Wei, Z. Remote sensing image scene classification using bag of convolutional features. *IEEE Geosci. Remote Sens. Lett.* **2017**, *14*, 1735–1739. [CrossRef]
9. Yuan, B.; Li, S.; Li, N. Multiscale deep features learning for land-use scene recognition. *J. Appl. Remote Sens.* **2018**, *12*, 015010. [CrossRef]
10. Zhang, C.; Prasanna, V. Frequency Domain Acceleration of Convolutional Neural Networks on CPU-FPGA Shared Memory System. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; pp. 35–44.
11. Zhang, J.; Li, J. Improving the Performance of OpenCL-based FPGA Accelerator for Convolutional Neural Network. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; pp. 25–34.

12. 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]
13. Mei, C.; Liu, Z.; Niu, Y.; Ji, X.; Zhou, W.; Wang, D. A 200MHZ 202.4GFLOPS@10.8W VGG16 accelerator in Xilinx VX690T. In Proceedings of the 2017 IEEE Global Conference on Signal and Information Processing, Montreal, QC, Canada, 14–16 November 2017; pp. 784–788.
14. Gysel, P.; Motamedi, M.; Ghiasi, S. Hardware-oriented approximation of convolutional neural networks. *arXiv* **2016**, arXiv:1604.03168.
15. Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *J. Mach. Learn. Res.* **2017**, *18*, 6869–6898.
16. Han, S.; Pool, J.; Tran, J.; Dally, W. Learning both weights and connections for efficient neural network. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; pp. 1135–1143.
17. Liang, S.; Yin, S.; Liu, L.; Luk, W.; Wei, S. FP-BNN: Binarized Neural Network on FPGA. *Neurocomputing* **2018**, *275*, 1072–1086. [CrossRef]
18. Zhao, R.; Song, W.; Zhang, W.; Xing, T.; Lin, J.H.; Srivastava, M.; Gupta, R.; Zhang, Z. Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; pp. 15–24.
19. 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.
20. Liu, B.; Zou, D.; Feng, L.; Feng, S.; Fu, P.; Li, J. An FPGA-Based CNN Accelerator Integrating Depthwise Separable Convolution. *Electronics* **2019**, *8*, 281. [CrossRef]
21. Alwani, M.; Chen, H.; Ferdman, M.; Milder, P. Fused-layer CNN accelerators. In Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture, Taipei, Taiwan, 15–19 October 2016; pp. 1–12.
22. Sun, F.; Wang, C.; Gong, L.; Xu, C.; Zhou, X. A high-performance accelerator for large-scale convolutional neural networks. In Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications, Lausanne, Switzerland, 29 August–2 September 2016; pp. 1–9.
23. Li, L.; Zhang, S.; Wu, J. Efficient Object Detection Framework and Hardware Architecture for Remote Sensing Images. *Remote Sens.* **2019**, *11*, 2376. [CrossRef]
24. Wang, J.; Liu, W.; Ma, L.; Chen, H.; Chen, L. IORN: An Effective Remote Sensing Image Scene Classification Framework. *IEEE Geosci. Remote Sens. Lett.* **2018**, *15*, 1695–1699. [CrossRef]
25. Wei, X.; Liu, W.; Chen, L.; Ma, L.; Chen, H.; Zhuang, Y. FPGA-Based Hybrid-Type Implementation of Quantized Neural Networks for Remote Sensing Applications. *Sensors* **2019**, *19*, 924. [CrossRef] [PubMed]
26. Lu, X.; Wang, B.; Zheng, X.; Li, X. Exploring models and data for remote sensing image caption generation. *IEEE Trans. Geosci. Remote Sens.* **2018**, *56*, 2183–2195. [CrossRef]
27. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. In Proceedings of the 2015 International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015; pp. 1–14.
28. Zhou, Y.; Ye, Q.; Qiu, Q.; Jiao, J. Oriented response networks. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4961–4970.
29. Liu, Z.; Chow, P.; Xu, J.; Jiang, J.; Dou, Y.; Zhou, J. A Uniform Architecture Design for Accelerating 2D and 3D CNNs on FPGAs. *Electronics* **2019**, *8*, 65. [CrossRef]
30. Ma, Y.; Cao, Y.; Vrudhula, S.; Seo, J.S. Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; pp. 45–54.