remote sensing

MDPI

Article

Efficient Object Detection Framework and Hardware Architecture for Remote Sensing Images

Lin Li [1,2,*], Shengbing Zhang [1] and Juan Wu [3]

1 School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710072, China; zhangsb@nwpu.edu.cn
2 Fourth Design Department, Beijing Institute of Microelectronics Technology, Beijing 100076, China
3 School of Animation and Software, Xi'an Vocational and Technical College, Xi'an 710077, China; wujuan0226@outlook.com
* Correspondence: lilin0106@mail.nwpu.edu.cn; Tel.: +86-29-65685192

Received: 18 August 2019; Accepted: 11 October 2019; Published: 13 October 2019

Abstract: Object detection in remote sensing images on a satellite or aircraft has important economic and military significance and is full of challenges. This task requires not only accurate and efficient algorithms, but also high-performance and low power hardware architecture. However, existing deep learning based object detection algorithms require further optimization in small objects detection, reduced computational complexity and parameter size. Meanwhile, the general-purpose processor cannot achieve better power efficiency, and the previous design of deep learning processor has still potential for mining parallelism. To address these issues, we propose an efficient context-based feature fusion single shot multi-box detector (CBFF-SSD) framework, using lightweight MobileNet as the backbone network to reduce parameters and computational complexity, adding feature fusion units and detecting feature maps to enhance the recognition of small objects and improve detection accuracy. Based on the analysis and optimization of the calculation of each layer in the algorithm, we propose efficient hardware architecture of deep learning processor with multiple neural processing units (NPUs) composed of 2-D processing elements (PEs), which can simultaneously calculate multiple output feature maps. The parallel architecture, hierarchical on-chip storage organization, and the local register are used to achieve parallel processing, sharing and reuse of data, and make the calculation of processor more efficient. Extensive experiments and comprehensive evaluations on the public NWPU VHR-10 dataset and comparisons with some state-of-the-art approaches demonstrate the effectiveness and superiority of the proposed framework. Moreover, for evaluating the performance of proposed hardware architecture, we implement it on Xilinx XC7Z100 field programmable gate array (FPGA) and test on the proposed CBFF-SSD and VGG16 models. Experimental results show that our processor are more power efficient than general purpose central processing units (CPUs) and graphics processing units (GPUs), and have better performance density than other state-of-the-art FPGA-based designs.

Keywords: object detection; remote sensing image; deep learning; convolutional neural networks (CNNs); hardware architecture; processor

1. Introduction

Object detection in high resolution optical remote sensing images is to determine if a given aerial or satellite image contains one or more objects belonging to the class of user focused and locate the position of each predicted object in the image [1]. As an important research topic of remote sensing images analysis, object detection in remote sensing images is widely applied to military reconnaissance, intelligent transportation, urban planning, and other domains [2–5]. In recent years,

with the development of optical remote sensing technology and space-borne intelligent information processing system, it is a trend to construct a system that combines remote sensing detection with information processing on satellite or aircraft. Efficient object detection in a remote sensing image and processing on a satellite or aircraft can not only reduce the amount of communication data, but also achieve efficient, flexible, and fast earth observation tasks. However, there are many challenges to detect the user-concerned objects quickly and accurately from the massive remote sensing data. Firstly, remote sensing images have ultra-high spatial resolution, which usually contains tens or hundreds of millions of pixels. Quickly and accurately detecting the user-focused objects from massive amounts of data is a challenging task. Secondly, objects in remote sensing images have multi-scale features. For example, objects such as a ground track field, bridge, etc. have hundreds of pixels, while small objects such as a vehicle, ship, etc. may only contain a few pixels. This feature makes accurate object detection in remote sensing images more difficult, especially for small objects. Thirdly, objects in a remote sensing image viewed from overhead have any orientation, while natural image sets are typically acquired horizontally. Therefore, models trained on natural image sets cannot be directly applied to remote sensing image object detection. In addition, the general-purpose processor that carries out the algorithm cannot meet the requirements of space-borne or airborne information processing with high performance and low energy consumption. Therefore, the design of efficient object detection algorithm framework and hardware architecture for remote sensing images has become an urgent problem to be solved for space-borne or airborne information processing.

There are many methods for object detection in remote sensing images after years of research and development. We summarize these methods into traditional object detection methods based on prior information and manual features and deep learning based object detection methods. The traditional object detection methods based on prior information and manual designed features regard object detection as a classification problem composed of feature extraction and object classification, which includes template matching-based object detection methods, knowledge-based object detection methods, object-based image analysis-based (OBIA-based) object detection methods, and machine learning object detection methods based on prior information and manual designed features [1]. The template matching-based object detection methods are divided into two steps. Firstly, the template is trained from existing data by hand-crafting or statistical methods, and then the similarity measurement is performed on the pre-processed input image to complete the detection. The template matching-based object detection approaches are usually divided into a rigid template and deformable template according to the template type selected by the user [1,6,7]. The knowledge-based object detection methods setup knowledge and rules based on geometric information and context information, and generate hypotheses on the input image and convert the object detection problem into a hypothesis testing problem [8–10]. The OBIA-based object detection methods accomplish the detection task by segmenting the input image and classifying the object, wherein the scale of the image segmentation directly affects the detection result [11–13]. The machine learning object detection methods based on prior information and manual designed features are typically divided into two stages: feature extraction stage and object classification and recognition stage. In the feature extraction stage, selective search is usually used to extract handcrafted features, such as scale-invariant feature transform (SIFT) [14], histogram of oriented gradients (HOG) [15], bag-of-words (BoW) feature [16], texture features [17], and so on. In the stage of object classification and recognition, classifiers often include: support vector machine (SVM) [18], AdaBoost [19], deformable parts model (DPM) [20], condition random field (CRF) [21], sparse coding-based classifier [22] and artificial neural network (ANN) [23], and so on. However most of these methods mentioned above rely on the prior information and manual designed features, and it is difficult to efficiently achieve object detection tasks under massive remote sensing data.

In recent years, deep learning technology has achieved great success in computer vision applications, and the deep learning based object detection methods have become the mainstream in the field of image recognition. The deep convolutional neural network AlexNet proposed by Krizhevsky, A. et al. not only won the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVR 2012),

but also set off a wave of deep learning research [24]. Since then, many researchers have proposed a variety of excellent deep learning algorithms, and deep convolutional neural networks have become the best deep learning algorithm in the field of image recognition. Researchers have also begun to apply deep learning technology to object recognition in remote sensing images. For example, Cheng, G. et al. applied the rotation-invariant convolutional neural networks (RICNN) to object detection in very high resolution (VHR) optical remote sensing images [25]. Wang, G. et al. proposed the infrastructure target detection of remote sensing image based on residual networks [26]. Although these object detection methods based on convolutional neural networks perform well in remote sensing image object detection, they do not form a unified and efficient object framework. Currently, the deep learning object detection algorithm framework based on convolutional neural network has made great progress. These mainstream algorithm frameworks include region proposal-based two-stage object detection algorithm and regression-based one stage object detection algorithm [27]. The region proposal-based algorithm generates a series of region proposals according to selective search (SS), Bing or edge boxes methods firstly, and then extracts the features by the deep neural networks, and implements object classification and boundary regression based on these features. For example, Girshick, R. et al. proposed region-based convolutional neural networks (R-CNN) [28], which combines object candidate regions and deep learning for object detection. Then he proposed the more efficient fast R-CNN algorithm [29], which overcomes the shortcomings of R-CNN's redundant operation when extracting features. Subsequently, Ren, S. et al. proposed a faster R-CNN algorithm [30], which uses region proposal networks (RPN) to extract object candidate regions, and integrates the entire object candidate region extraction, feature extraction, object recognition, and detection into a deep neural network framework. In order to solve the multi-scale detection problem, Lin, T.Y. et al. introduced the feature pyramid network (FPN) to improve the recognition efficiency of small objects [31]. However, the object detection algorithm framework based on the region proposal is not very efficient because it takes more time to extract the candidate region. The regression-based object detection algorithm has no candidate region extraction step, which combines all recognition and detection steps in a deep neural network, and has high detection and recognition efficiency. For this type of algorithm framework, the you only look once (YOLO) algorithm framework proposed by Redmon, J. et al. requires only a single network to evaluate the entire image to obtain the target bounding box and category [32]. The single shot multi-box detector (SSD) algorithm framework proposed by Liu, W. et al. introduces an anchor mechanism based on YOLO, which detects the object on the different scales and improves accuracy without affecting process speed [33]. In general, the deep learning based object detection methods have made great progress in the accuracy and efficiency of object detection compared to the traditional methods. At present, these algorithms are widely used in remote sensing image object detection. However, these algorithms are not satisfactory for the detection of small objects in the images, and they need to be improved. Meanwhile, for the airborne or space-borne application environment, not only the object detection accuracy but also the computation complexity and model size need to be considered.

The rapid development of deep learning technology is inseparable from the support of high performance hardware computing systems. The performance of deep learning algorithms is not only related to its own structure, but also depends on the hardware architecture of computing system that carries out the algorithm. Currently, the training and inference of deep learning algorithms mainly depends on general-purpose processors, such as central processing units (CPUs) and graphics processing units (GPUs). Although the general-purpose CPUs have higher flexibility and better parallel computing power, the deep learning algorithm does not achieve better execution efficiency. GPUs are widely used in training and inference of deep learning algorithms because of its unique many-core architecture and superior parallel computing power, but it cannot obtain good performance-power ratio due to high power consumption. In some specific applications, not only high processing performance but also stricter power consumption requirements are required. For example, in embedded application scenarios or on satellite or aircraft information processing systems, general-purpose CPUs and GPUs will not be able to accommodate the application requirements in such situations. Therefore,

application-oriented domain-specific architecture (DSA) is the solution to overcome such problems currently [34]. In recent years, many researchers have proposed different hardware architectures for their respective application scenarios. Farabet, C. et al. proposed CNP [35] with parallel vector computing architecture for low-power lightweight unmanned aircraft vehicles (UAVs) or robots, and a scalable dataflow 2-D grid hardware architecture Neuflow [36] optimized for the computation of general-purpose vision algorithms. Peemem, M. et al. proposed hierarchical memory-centric accelerator architecture to improve the performance of convolutional operations and reduce the overhead of memory access [37]. Alwani, M. et al. reduced the transfer of off-chip feature map data by modifying the order of input data and fusing multiple continuous convolutional layer processing [38]. Chen, T. et al. presented a high-throughput algorithm accelerator DianNao based on adder tree structure for large-scale convolutional neural network (CNN) and deep neural network (DNN) [39]. Du, Z. et al. proposed ShiDianNao based on 2-D mesh topology structure for image recognition applications near to sensors, and reduced memory usage through weight sharing [40]. Zhang, C. et al. designed a CNN accelerator based on the adder tree structure by quantitative analysis of memory bandwidth required for throughput [41]. Google has introduced a high-performance tensor processing unit (TPU) for data centers based on 2-D systolic array architecture [42]. Li, L. et al. designed a co-processor with 2-D mesh topology structure for image recognition by optimization calculation of algorithm [43]. Chang, J.W. et al. proposed a deconvolutional neural networks accelerator (DCNN) with 2-D mesh architecture for super-resolution images [44]. These hardware architectures are designed for a specific application scenario and are mainly used to accelerate the calculation of the deep learning algorithms. The processing elements (PEs) in these processors are typically organized in 1-D or 2-D topology structure. These processors only implement parallel computation of synapses and neurons, which compute each feature map one by one. However, the feature map in the image object detection is 3-D, but the current design do not consider the parallel calculation of feature map, so there is still potential for mining parallelism. Meanwhile, the storage organization and data reuse need to be considered in the architecture to adapt to large-scale algorithms and parameters, which is very important for computing and storage dual-intensive remote sensing images object detection applications.

In order to adapt to the characteristics of object detection in remote sensing images and tackle the problems of the algorithm framework and hardware architecture mentioned above, we propose an efficient context-based feature fusion SSD (CBFF-SSD) algorithm framework. Subsequently, we have designed hardware architecture of deep learning processor with multiple 2-D mesh architecture supporting feature maps parallel processing by analyzing and optimizing the calculation of each layer in the deep learning algorithm framework. Finally, the efficiency and performance of the algorithm framework and processor are evaluated by multiple experiments and indicators. The main contributions of this paper are summarized as follows:

1. We propose a context-based feature fusion SSD (CBFF-SSD) framework for object detection in remote sensing images. MobileNet is used as the backbone network in the algorithm framework to reduce the amount of calculation and parameters, which makes the algorithm more efficient. Two feature fusion units and seven feature maps are used in the algorithm to enhance the detection of multi-scale objects and small objects, and improve the detection accuracy.
2. We analyze and optimize the calculation of each layer in the algorithm framework, which makes it easy to implement in hardware, and lays a foundation for the design of time-division multiplexing processing unit in the subsequent hardware architecture of deep learning processor.
3. We propose efficient hardware architecture of deep learning processor with multiple 2-D mesh topology oriented to image object recognition, which can simultaneously calculate multiple output feature maps. Hierarchical on-chip storage organization makes the neurons and weights to be efficiently delivered to the neural processing units (NPUs). The parallel architecture, the hierarchical storage organization, and the register designed in the PE effectively realize the sharing and reuse of the calculation data.

4. We evaluate the performance and efficiency of the algorithm framework and hardware architecture based on several experiments and evaluation indicators. The performance of the proposed algorithm framework is compared with the several popular algorithms on the NWPU-VHR-10 dataset. We realize the proposed hardware architecture of the deep learning processor on the field programmable gate array (FPGA), and then evaluate the processing performance and compared with the CPU, GPU, and the current popular deep learning processors. The experimental results confirmed the effectiveness and superiority of the proposed algorithm framework and hardware architecture of deep learning processor.

The rest of this paper is organized as follows. Section 2 proposes the context-based feature fusion SSD (CBFF-SSD) framework. The calculation of each layer in deep learning algorithm framework and optimization are described in Section 3. Section 4 introduces the details of the hardware architecture of deep learning processor. Section 5 presents the experimental results and analysis. The experimental results are discussed in the Section 6. Finally, the conclusions are drawn in Section 7.

2. Context-Based Feature Fusion SSD Framework

2.1. Related Works

The deep learning algorithm based on the convolutional neural network model has achieved excellent results in image object detection applications, which not only improves the accuracy of recognition, but also improves the efficiency of recognition. In particular, the recent rapid development of the region proposal-based objects detection algorithm framework and the regression-based objects detection algorithm framework are particularly outstanding. Since remote sensing images have ultra-high resolution, diverse object sizes and directions, including small targets, and diverse shooting angles, it is full of challenges to quickly and accurately detect the user-focused object from massive remote sensing data. For the application of remote sensing image object detection, many researchers have conducted a lot of research based on the two popular algorithm frameworks.

For examples, Han, X. et al. proposed a highly efficient and robust integrated geospatial object detect framework based on the faster region-based convolutional neural network (Faster R-CNN), which realized the integrated procedure by sharing features between the region proposal generation stage and the object detection stage [45]. Zhu, M. et al. proposed an effective airplane detection method in remote sensing images based on Faster R-CNN and multiplayer feature fusion, which solved the problem of insufficient representation ability of weak and small objects and overlapping detection boxes in airplane object detection [3]. Etten, A.V. presented a rapid multi-scale object detection method based on YOLO and DarkNet for the detection of small objects in large satellite imagery [46]. In order to solve the detection of small and dense objects, Zhang, X. et al. proposed an effective region-based VHR remote sensing imagery object detection framework named double multi-scale feature pyramid network (DM-FPN) [47].

In summary, the above algorithms are improved for the small objects detection in remote sensing images, and have achieved good object detection results. However, efficient on-board remote sensing image object detection not only needs to consider the accuracy of detection, but also needs to think about the efficiency of calculations such as computational complexity and the number of parameters.

2.2. CBFF-SSD Framework

Based on the in-depth analysis of the characteristics and challenges of object detection in a remote sensing image on a satellite or aircraft, we proposed a context-based feature fusion SSD (CBFF-SSD) algorithm framework shown in Figure 1. The whole algorithm framework was designed based on the SSD framework [33]. This is because on the one hand, the regression-based object detection framework is considered to be more efficient in image object detection, and the other is because the algorithm framework is more suitable for multi-scale object detection. Different from the SSD framework, the backbone network uses the MobileNet [48] instead of VGGNet. The lightweight MobileNet uses

depth-wise separable convolution to effectively reduce the amount of calculation and parameters of the algorithm, which is more conducive to efficient object detection in embedded applications, especially in a space-borne or airborne application environment. We compared the number of parameters and the amount of calculations of the proposed CBFF-SSD algorithm framework and the SSD algorithm in Table 1. It can be seen that the number of parameters of the proposed CBFF-SSD algorithm framework was 56.09% of the SSD algorithm, and the calculation amount was only 17.56% of the SSD algorithm. By reducing the size of the model parameters and computational complexity, the proposed algorithm framework was more effective in object detection.

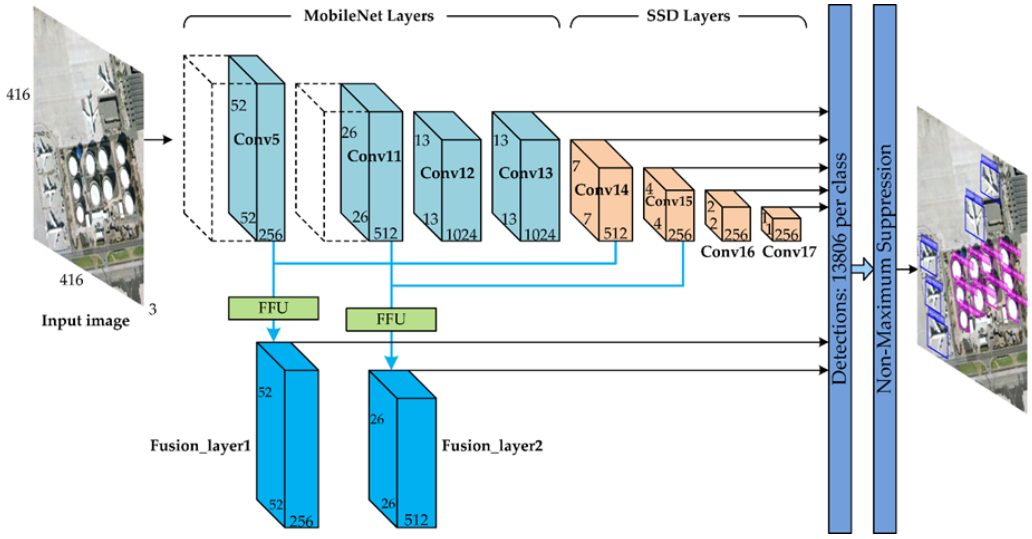

Figure 1. The overall structure of the proposed context-based feature fusion single shot multi-box detector (CBFF-SSD) algorithm framework.

Table 1. Comparison of the number of parameters and calculation amount of the SSD and the proposed CBFF-SSD framework.

Framework	Input Size	Parameters	Mult_Adds
SSD	300 × 300 × 3	26.28 Million	31.37 Billion
CBFF-SSD	416 × 416 × 3	14.74 Million	5.51 Billion

The high-level features of the convolutional network were rich in semantics and suitable for detecting large objects. After layer-by-layer down-sampling, the features lose too much detail information, and it is often important to small object detection. The feature rich in semantic information was mapped back to the lower layer features with larger resolution and richer detail information, and they were fused in an appropriate way to improve the effect of small object detection. Therefore, in the algorithm framework, the low-level feature map Conv5 (52 × 52 × 256) was added, and the Conv14 and Conv15 were respectively up-sampled and fused with Conv5 and Conv11 layers to improve the precision of small object detection.

Figure 1 shows the architecture details of the CBFF-SSD algorithm framework. The Fusion_layer1, Fusion_layer2, Conv13, Conv14, Conv15, Conv16, and Conv17 were used to predict both location and confidences. In the deep learning algorithm, the resolution of the input image and the number of detection boxes for each class of object affected the accuracy of the detection. Based on the SSD framework, we adjusted the input resolution from 300 × 300 to 416 × 416, and increased the number of detection boxes from 8732 to 13806 to ensure the recognition accuracy of the proposed algorithm framework.

2.3. Feature Fusion Unit (FFU)

There are two feature fusion units in the proposed algorithm framework as shown in Figure 1. The structure of the feature fusion unit is shown in Figure 2. The design of feature fusion unit is inspired by the design of the deconvolution module in the deconvolutional single shot detector (DSSD) [49]. The high-level feature maps are processed by deconvolution to the same size and channel as the lower-level feature maps. Then they are fused by element wise addition.

Taking the feature fusion layer 1 as an example, in order to fuse the feature maps of the Conv14 and Conv5, it is necessary to up-sample the resolution of the Conv14 layer by eight times. Specifically, for the Conv14 layer, we designed three deconvolution layers with stride 2 to achieve up-sampling. Since the feature maps in the feature fusion unit are computationally intensive, we also applied depth-wise separable convolutions here to reduce parameters and computational complexity. The deconvolution layer was followed by depth-wise separable convolution. The depth-wise separable convolution was composed of 3×3 depth-wise convolutional layer, batch normalization, rectified linear unit (ReLU) layer, 1×1 point-wise convolutional layer, batch normalization layer, and ReLU layer. The Conv5 layer underwent a depth-wise separable convolution module. After the normalization layer, we fused them by element-wise addition, and finally passed the ReLU to complete the fusion.

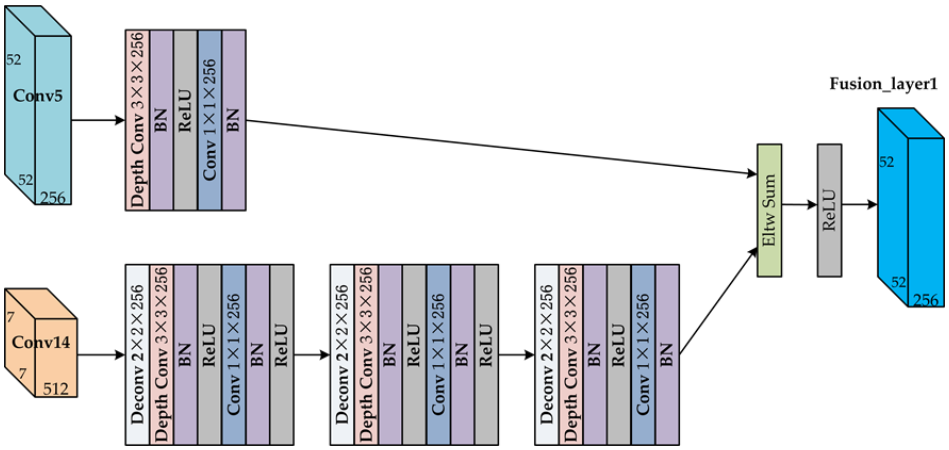

Figure 2. The structure of feature fusion unit (FFU).

Fusion layer 2 used the same calculation method, and only the channel was adjusted. Only minor modifications were required for models with different input resolutions.

2.4. Training

We used the same training strategy as SSD [33]. During training, a set of default boxes was matched to the ground truth boxes. For each ground truth box, we matched it to the default box with the Jaccard overlap higher than a threshold (e.g., 0.5). This was more conducive to predicting multiple bounding boxes with high confidence for overlapped objects. We selected the non-matched default boxes with top loss value as the negative samples so that the ratio of negative and positive was 3:1.

The training objective was for multiple object categories. We set x to be an indicator for matching the default box to the ground truth box of category p, which equaled to 1 or 0. The c, l, and g represent confidences, the predicted box, and ground truth box respectively. The overall objective loss function is a weighted sum of the localization loss L_{loc} and the confidence loss L_{conf}, which is defined as:

$$L(x, c, l, g) = \frac{1}{N}(L_{\text{conf}}(x, c) + \alpha L_{\text{loc}}(x, l, g)), \quad (1)$$

where N is the number of matched default boxes, and the weight term α is set to 1 by cross validation. The localization loss L_{loc} is a smooth L1 loss between the predicted box (l) and the ground truth box

(g) parameters. The confidence loss L_{conf} is the Softmax loss over multiple classes' confidences (c). The confidence loss L_{conf} and the localization loss L_{loc} are defined the same as SSD [33].

Seven feature maps were used to predict both location and confidences in the proposed CBFF-SSD framework. Specifically, we adopted four default boxes at each feature map for Fusion_layer1, Conv16, and Conv17, and used six default boxes at each feature map location for all other layers. We made a minor change on the scale of default boxes, the lowest layer had a scale of 0.15 and the highest layer had a scale of 0.9. The aspect ratio setting of default boxes were selected as 1, 2, 3, 1/2, and 1/3.

We also used the data augmentation strategy that was consistent with SSD to make the framework more robust to various input object sizes and shapes. These methods included random cropping, flipping, photometric distortion, and random expansion augmentation trick, which are very helpful for detecting small objects.

3. Calculation of Each Layer in the Deep Learning Algorithm Framework and Optimization

Based on the analysis of many deep learning algorithms used in image object detection and the framework proposed in Section 2, it can be seen that although the structures of these algorithms were different, they were all designed based on the deep convolutional neural network. These object detection algorithms are composed of some basic calculation layers, including the convolutional layer, deconvolutional layer, pooling layer, nonlinear activation function layer, normalization layer, element-wise sum layer, full connection layer, and Softmax layer. The hardware architecture suitable for algorithm computing was abstracted based on the analysis and optimization of each layer in the following.

3.1. Convolutional Layer and Deconvolutional Layer

The convolutional layer is composed of several convolution kernels, which is used to extract various features from the input feature maps. When calculating, the size of input feature maps f_{in} is defined as $W \times H \times C$; the kernels are expressed as $K_x \times K_y \times C \times M$ (M is the number of kernels, which is also equal to the number of output feature maps).

The output neuron N at position (x, y) of output feature map f_{out} is computed with:

$$N_{x,y}^{f_{out}} = \sum_{f_{in}=0}^{C-1} \sum_{j=0}^{K_y-1} \sum_{i=0}^{K_x-1} W_{i,j}^{f_{in},f_{out}} * N_{x*S_x+i,y*S_y+j}^{f_{in},f_{out}} + Bias^{f_{in},f_{out}}, \quad (2)$$

where W and $Bias$ represent the kernels and bias parameters between input feature map f_{in} and output feature map f_{out} respectively, and S_x and S_y are the sliding steps when the image convoluted in the x-direction and y-direction.

The standard convolutional formula is shown as Equation (2), which can be transformed into depth-wise convolution and 1×1 point-wise convolution by modifying the kernels. The depth-wise convolution and point-wise convolution are the two key computation layers of depth-wise separable convolution, which is widely used in my proposed framework. When the kernel channels C are equal to 1, and the number of kernels M are equal to the number of channels of input feature maps, the kernels can be expressed as $K_x \times K_y \times M$, the standard convolution has been transformed into depth-wise convolution. When the kernel size K_x and K_y are both equal to 1, the standard convolution has been transformed into 1×1 point-wise convolution. It can be seen from the above Equation that the basic calculations of the convolutional layer are multiplication and addition.

Deconvolutional in image object detection typically refer to transposed convolution or dilated convolution, which is used to up-sample the result of the convolutional layer back to the resolution of the original image. The calculations of deconvolutional are similar to the convolutional layer. The output neuron can be calculated according to the Equation (2), and its basic calculation is also composed of multiplication and addition.

3.2. Pooling Layer

The pooling layer is also called the down-sampling layer, which reduces the amount of data onto feature maps by maximizing or averaging the neuron in each pooling window. The calculation of pooling not only retains the main features, simplifies the computational complexity of network, but also effectively controls the risk of over-fitting of deep neural networks. Maximum pooling and average pooling are two commonly used pooling methods. In the pooling calculation of two-dimensional image feature map, the pooling window is defined as $P_x \times P_y$, and the input feature map f_{in} and output feature f_{out} are one-to-one correspondence.

The maximum pooling formula for the output neuron N at position (x, y) of output feature f_{out} is:

$$N_{x,y}^{f_{out}} = \max_{0 \le i \le P_x - 1, 0 \le j \le \mathrm{Py}-1} N_{x+i,y+j}^{f_{in}}, \tag{3}$$

where the maximum pooling is done by successively comparing the maximum values of neuron in the pooling windows, which can be achieved by a hardware comparator. The average pooling formula for the output neuron N at position (x, y) of output feature f_{out} is:

$$N_{x,y}^{f_{out}} = \frac{\sum_{i=0}^{P_x-1} \sum_{j=0}^{P_y-1} N_{x+i,y+j}^{f_{in}}}{P_x * P_y}. \tag{4}$$

The calculation of average pooling is realized by the accumulating the neurons in the pooling window and dividing by the size of the pooling window. Since the structure of the neural network is determined, the $1/(P_x \times P_y)$ in formula (4) can be regarded as a coefficient, that multiplied by each neuron in the pooling window and accumulated to realize the calculation of average pooling. This method saves hardware resources by eliminating divisions in average pooling calculations and converting them into multiplication and addition operations.

3.3. Nonlinear Activation Function Layer

The nonlinear activation functions are widely used in neural networks, which not only make the neural network have nonlinear learning and an expression ability by layered nonlinear mapping compound, but also enhance the ability of the network to represent the high-level semantics of data. Commonly used activation functions include Sigmoid, Tanh, ReLU, and so on. The Sigmoid and Tanh functions are usually approximated by piecewise linear interpolation method, which is composed of multiplication and addition.

Recently, the ReLU has been widely used in many current deep learning algorithms and neural networks because it can effectively alleviate over-fitting and is less prone to gradient loss. Its calculation formula is:

$$ReLU(N) = \begin{cases} N, N > 0 \\ 0, N \le 0 \end{cases}, \tag{5}$$

where N is the neuron. Since the signed fixed-point number is adopted in the hardware architecture design of deep learning processor, the calculation for ReLU can be completed directly by judging the sign bit of neuron.

In this paper, we drew on an efficient hardware pipeline architecture proposed by Li, L. et al. to realize the calculation of the activation function, which is another result of our work [50].

3.4. Normalization Layer

The normalization operation in the neural network is to solve the problem that the distribution of the data in the middle layer changes during the training process to prevent the gradient from disappearing or exploding and speed up the training. Krizhevsky, A. et al. used a local response normalization (LRN) operation in AlexNet to reduce the error rates of top-1 and top-5 by 1.4% and

1.2% [24]. Du, Z. et al. added local response normalization (LRN) and local contrast normalization (LCN) in the design of ShiDianNao, which improved the recognition accuracy, but increased the computational and hardware complexity [40]. The batch normalization proposed by Ioffe, S. et al. in 2015 is widely used in the deep neural network, which effectively accelerates the speed of training and convergence [51]. The batch normalization is calculated as a separate layer in the neural network forward inference process. The batch normalization formula for the neuron N located at the (x, y) position of output feature map f_{out} is as follows:

$$N_{x,y}^{f_{out}} = (N_{x,y}^{f_{in}} - \frac{mean_{x,y}^{f_{in}}}{scalefactor}) / \sqrt{\frac{Variance_{x,y}^{f_{in}}}{scalefactor} + \varepsilon}, \quad (6)$$

where *mean* and *Variance* are the mean and variance of the input feature map fin respectively, and *scalefactor* is the scaling factor. These three parameters are learned by network training. The ε is a small constant, usually taken as 0.00001. It can be intuitively seen that the batch normalization calculation includes complex operations such as division and square root, and we will optimize the calculation using the parameter preprocessing strategy. Therefore, the Equation (6) can be converted into the Equation (7) shown below,

$$\begin{cases} N_{x,y}^{f_{out}} = BN\_a_{x,y}^{f_{in}} * N_{x,y}^{f_{in}} + BN\_b_{x,y}^{f_{in}} \\ BN\_a_{x,y}^{f_{in}} = 1 / \sqrt{\frac{Variance_{x,y}^{f_{in}}}{scalefactor} + \varepsilon} \\ BN\_b_{x,y}^{f_{in}} = -\frac{mean_{x,y}^{f_{in}}}{scalefactor} * BN\_a_{x,y}^{f_{in}} \end{cases}, \quad (7)$$

where *BN_a* and *BN_b* are new parameters obtained by parameter preprocessing. Thus, complex batch normalization calculations are converted into multiplication and addition operations, thereby simplifying the complexity of the hardware structure.

3.5. Element-Wise Sum Layer

The calculation of element-wise sum is introduced when performing feature fusion, and this layer fuses feature maps of the same size produced on different paths. The main calculation of this layer is the addition of neurons at the corresponding positions of input feature maps, which can be implemented by hardware adder.

3.6. Full Connection Layer

The full connected layer is used to combine the features extracted from previous layers of the network, which is usually at the top of the neural network and used as a classifier. Although the full connect layer is no longer used in some current object detection algorithms, in order not to lose generality, the calculations of this layer are listed here. The input feature maps f_{in} is a vector of $1 \times 1 \times C$, and the output feature maps f_{out} is a vector of $1 \times 1 \times M$. The output neuron N at position (1, 1) of output feature map f_{out} is computed with:

$$N_{1,1}^{f_{out}} = \sum_{f_{in}=0}^{C-1} W_{1,1}^{f_{in},f_{out}} * N_{1,1}^{f_{in}} + Bias^{f_{in},f_{out}}, \quad (8)$$

where W and $Bias$ represent the kernels and bias parameters between input feature map f_{in} and output feature map f_{out} respectively. The calculation of the full connect layer is similar to that of the convolutional layer, which is composed of multiplication and addition.

3.7. Softmax Layer

The Softmax layer is used for the output of the multi-classification neural network, which maps the M-dimensional vector V into an M-dimensional vector S with a range between (0,1) and a cumulative sum equal to 1. The calculation of the Softmax layer is:

$$S_i = \frac{e^{V_i}}{\sum_{j=1}^{M} e^{V_j}} (i = 1, 2, \ldots, M), \tag{9}$$

where V and S are both M-dimensional vectors. The Softmax layer also includes exponential calculation. Consequently, we drew on an efficient hardware pipeline architecture based on piecewise linear interpolation proposed by Li, L. et al. [50], which shares the same hardware as the calculation of the activation function.

Based on the characteristics of neural network layer-by-layer calculation, this section analyzed and optimized the calculation of each layer, and summarized the basic calculation of deep learning processor. The algorithm also includes some calculations that are difficult to implement in hardware. We would implement them in software, such as the calculation of the default boxes, non-maximum suppression (NMS), and so on. By considering the complexity of the hardware implementation, the same or a similar calculation were used at each layer as much as possible, laying the foundation for the design of the time-division multiplex hardware architecture in the next section.

4. Hardware Architecture of Deep Learning Processor

In order to adapt to the high performance and low power consumption environment on the satellite or aircraft, we used the dedicated deep learning processor to replace the traditional CPUs or GPUs for object detection algorithm processing. Due to various considerations, the deep learning hardware architecture we designed only supports the calculation of the algorithm inference stage, and the training and the acquisition of parameters are realized by the offline mode.

Based on the characteristics of algorithmic hierarchical computing and the analysis and optimization of the calculation of each layer in deep learning algorithm in Section 3, we designed the hardware architecture of deep learning processor shown in Figure 3. Our deep learning processor consists of the following main components: memory controller interface, feature maps buffer, parameters buffer, instruction memory, decoder, buffer controller and processing data setup module, neural processing engine, and activation pipeline module.

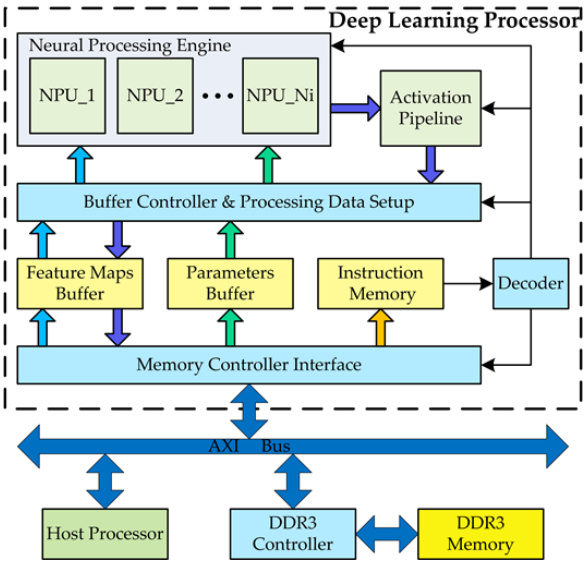

Figure 3. The overall diagram of the designed hardware architecture of the deep learning processor.

The memory controller interface interacts with the outside system through the advanced extensible interface (AXI) bus. It stores the received image data, parameters, and instructions into the feature maps buffer, the parameters buffer, and instruction memory respectively, and returns the calculation result of the deep learning processor to the system. The feature maps buffer is used to store input and output feature maps. The parameters buffer is used to store the weight and bias parameters obtained by offline training. The instructions of deep learning processor are stored in the instruction memory. After the instruction is decoded, the control signals are respectively transmitted to the respective functional modules. The buffer controller and processing data setup module readout feature maps data and parameters, and then sends them to the neural processing engine after organization, and stores the calculated output feature maps data of the activation pipeline module into the feature maps buffer. The neural processing engine performs basic neuron operations such as multiplication, addition and comparison. The activation pipeline module is used to implement an approximate calculation of the nonlinear activation function.

4.1. Parallel Computing Architecture

In the image object detection application, the input and output feature maps are both three-dimensional. According to the characteristics of the feature map, we naturally thought of parallel computing for three different dimensions in calculation. However, in the current research, most designs were performed in parallel for a two-dimensional neuron of an output feature map until this one was finished and next one began [35–44]. Therefore, we designed multiple neural processing units (NPUs) in the neural processing engine to realize parallel computing of multiple output feature maps. The hardware architecture of neural processing engine we designed is shown in Figure 4.

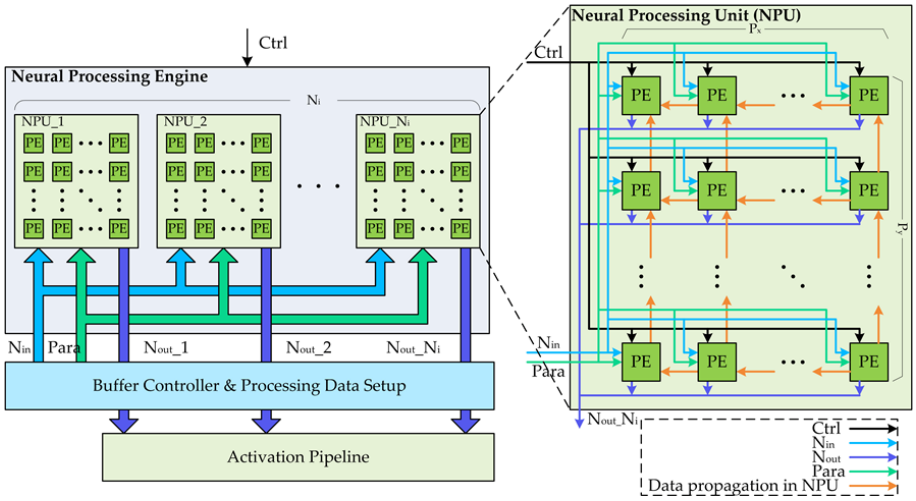

Figure 4. The hardware architecture of the neural processing engine. The left part is the neural processing engine consisting of multiple neural processing units (NPUs) and its data flow. The right part is the structure and data flow of a neural processing unit (NPU) composed of multiple processing elements (PEs).

We designed N_i neural processing units (NPUs) in the neural processing engine, which can simultaneously calculate the N_i output feature maps. The buffer controller and processing data setup module provides neurons and synapses for each neural processing unit for calculation. It reads data from the feature maps buffer and parameters buffer and reorganizes it for distribution to the neural processing units. After the calculation is completed, the neural processing engine outputs to the activation pipeline module for processing, and then writes the results to the feature maps buffer-by-buffer controller. The neural processing unit (NPU) consists of $P_x \times P_y$ processing elements (PEs) arranged in 2-D format. As can be seen from the right part of Figure 4, the neuron data calculated

by the PEs could come not only from the buffer controller but also from right or bottom PEs. The data propagation between the PEs achieved data reuse, and reduced the bandwidth of reading the feature maps buffer.

The data propagation path and the computation path were designed in the PE structure shown in Figure 5 to implement data reuse and neuron calculation functions respectively. The data propagation path was used for local transfer of neuron data between PEs, thereby realizing data reuse. Inspired by the calculation process of the sliding window during image convolutional operations, the input of this path included three ways of reading from the buffer controller and reading from the right or bottom PE. We set two sets of shifter registers in this path, which were row shift register (Row_Shifter) and column shift register (Col_Shifter). These registers were used to implement temporary storage and propagation of the reused neuron data. Based on the analysis of many current mainstream deep learning algorithms and the object detection algorithm proposed in Section 2, the stride of convolutional operations generally did not exceed 2. Therefore, in each set of shift registers we set up two serially connected 16-bit registers to form a queue. The specific use of these two registers was detailed in the Section 4.3 data sharing and reuse. The hardware structure of the computation path included a multiplexer, multiplier, adder, comparator, and register. The computation path mainly completed the calculation of the PE, which supported multiplication, addition and comparison operations, and implemented all the calculation of the algorithm layers except for activation function and Softmax. When the computation path worked, the neuron data came from the PE_Reg, and the parameters were selected according to the calculation of each layer. For example, weight and bias were used for the calculation of convolutional layer, and two pre-processed parameters BN_a and BN_b for the computation of the normalization layer.

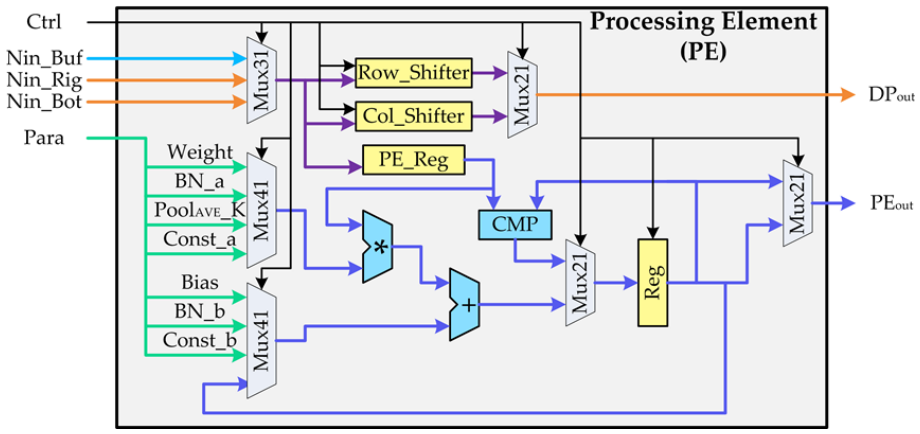

Figure 5. The structure of the processing element.

4.2. Hierarchical Storage Organization

The application of object detection in a remote sensing image is not only computation intensive, but also storage-intensive. Therefore, the efficient calculation of the neural processing engine is inseparable from the efficient organization and timely delivery of neuron data. The hierarchical storage organization structure as show in Figure 6 was adopted to store the feature maps in our design.

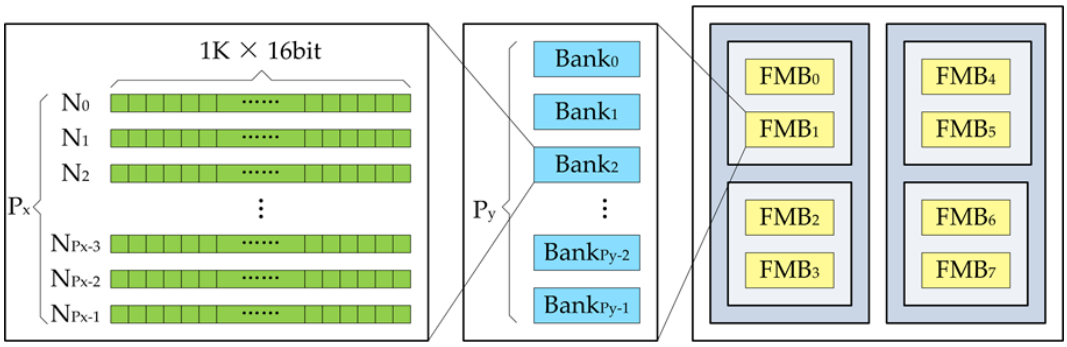

Figure 6. The hierarchical storage organization of the feature maps buffer.

We can see from the Figure 6, the structure of the feature map buffer was related to the number of PEs in the NPU. We designed P_x independent 16-bit width memory in one bank, where P_x is the number of PEs in a row of the NPU. Designing with independent memory made it easy for us to address a single neuron. In the upward level of storage, we designed P_y banks, where P_y is the number of PEs in a column of the NPU. This design allowed us to not only address a single neuron, but also read $P_x \times P_y$ neurons in one clock period. In order to achieve greater memory bandwidth, we could also design the separate memory similar to the multi-port format of the register file. The feature map buffer was divided into eight blocks, each of which contained P_y banks. Each of the two memory blocks constituted a set of Ping-Pong memories for alternately storing input or output feature maps. When a set of memory (e.g., FMB_0 and FMB_1) is the input feature map buffer, another set (e.g., FMB_2 and FMB_3) is used to store the output feature map. The input and output characteristics of the buffer were alternately changed, and the current output feature map buffer was the input feature map buffer calculated by the next layer of the algorithm. The other half of the buffer was used as a temporary buffer to handle data interactions of large algorithm when the feature maps size was larger than the on-chip buffer capacity. At this time, the two halves of the buffer acted as Ping-Pong memory to achieve alternate storage of the feature maps.

In order to effectively read out neuron data and reorganize it, the buffer controller reads the feature map buffer in four ways:

- Read $P_x \times P_y$ neurons from P_y banks.
- Read P_x neurons from 1 bank.
- Read P_y neurons from P_y banks with given stride.
- Read a single neuron.

These four reading modes can complete the reading of the neuron data in each layer of the deep learning algorithm.

The parameters buffer mainly stores the weights and bias data of the convolutional layer and the preprocessing parameters of the normalization layer. Due to the parameter sharing feature in the convolutional network, the structure of the parameters buffer is slightly different from the feature map buffer. The parameters buffer contains two levels storage structure. We designed N_i independent memories in its first level storage for a bank, where N_i is the number of NPUs. The second level storage contains four independent banks, which form two sets of Ping-Pong memory. In order to adapt to the large-scale model, the two sets of buffer were alternately used for the storage of current parameters and the interaction of subsequent parameters. The parameters are sequentially stored in the first-level independent memory according to the dimensions of the output feature map. The instruction buffer was designed as a FIFO (first input first output) with a width of 128 bits and a depth of 2 k. The very long instruction word (VLIW) instructions are sequentially stored in it. Since the structure of the instruction buffer is simple, it will not be described more here.

4.3. Data Sharing and Reuse

In order to reduce the bandwidth of the read buffer and perform calculations efficiently, and to make full use of the weight sharing of the convolutional neural network, we took a lot of work in the design of the deep learning processor. Taking the convolutional operation with the largest amount of calculation in the object detection algorithm as an example, since the N_i NPUs share the input neurons, only the $P_x \times P_y$ neurons data are read out and broadcast to the N_i NPUs for calculation in the first cycle. At the same time, the weights are shared within the NPU, and only the N_i weights are read from the parameters buffer and sent to the corresponding NPU for calculation. In the subsequent cycles of convolutional operation, not only data sharing but also data reuse based on inter-PE neuron propagation was adopted. The specific process of data reuse based on inter-PE neuron propagation in the convolutional operation is show in Figures 7 and 8.

We can see from Figure 7 that when stride was equal to 1, neuron reuse used only one row register and one column register for 3×3 convolutional operations. The row and column shift registers in the PE could be configured by the control signal to write neuron data directly to the output stage registers while masking the first stage registers. When Stride was equal to 1, the data could be reused for all cycles of the convolutional operation except for the first cycle. Specifically, $(K_x - 1) \times K_y$ row propagations were performed, and $(K_y - 1)$ column propagations were performed, where K_x and K_y was the kernel size.

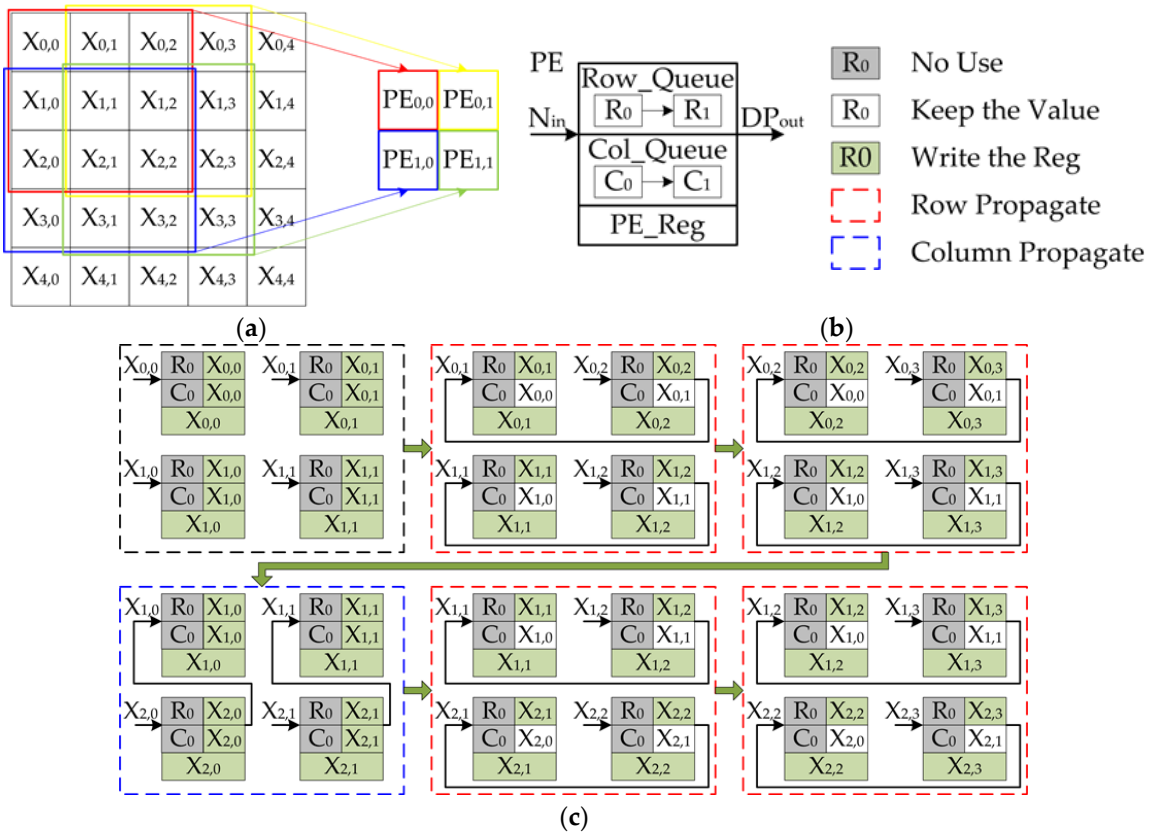

Figure 7. The neuron propagation between PEs and data reuse during the convolutional operation when stride = 1. (**a**) The mapping between neurons and PEs. (**b**) PE model, register operations, and neuron propagation between PEs. (**c**) The neuron propagation and register operation of the neuron between PEs in first six cycles during the convolutional operation.

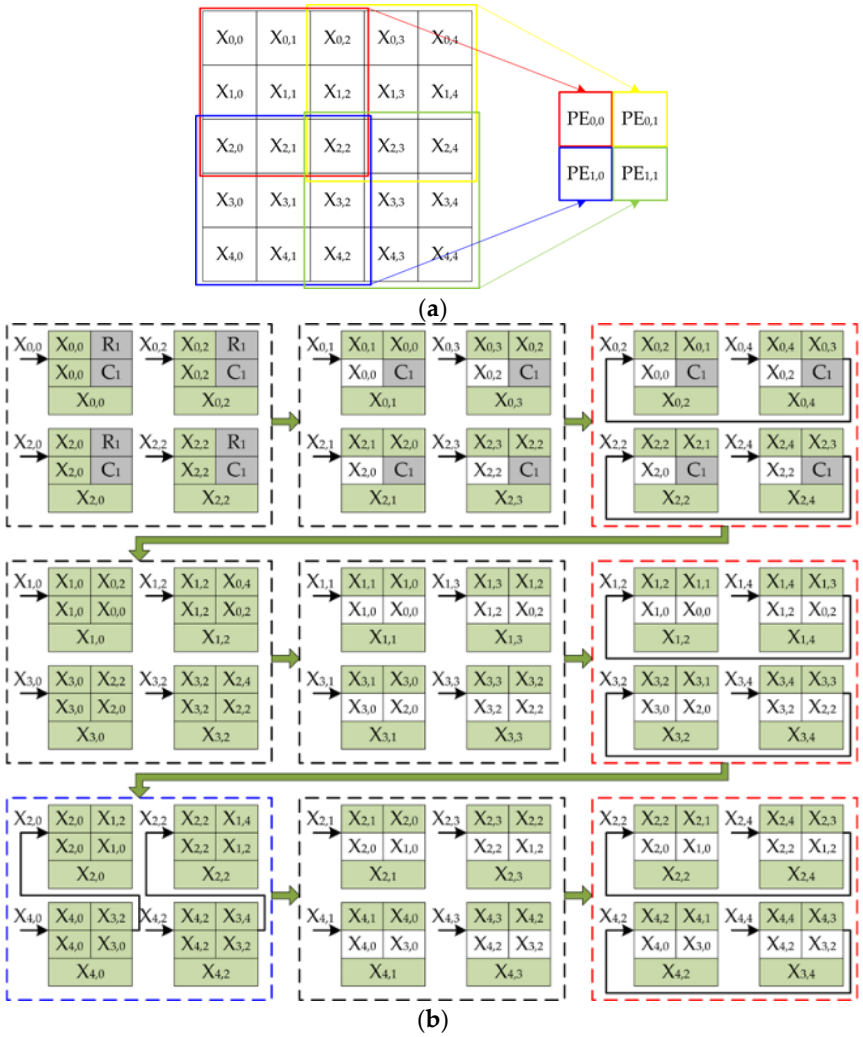

Figure 8. The neuron propagation between PEs and data reuse during the convolutional operation when stride = 2. (**a**) The mapping between neurons and PEs. (**b**) The specific operation of the PE registers and neuron propagation between PEs.

Based on the analysis of many current mainstream deep learning algorithms and the object detection algorithm proposed in Section 2, the stride of convolutional operations generally did not exceed 2. Therefore, in each set of shift registers in PE we set up to two serially connected 16-bit registers to form a queue. We can see from Figure 8, for a convolutional operation with a stride equal to 2, only four 16-bit registers were needed to achieve data reuse, which saved hardware resources compared to the design of FIFO for six 16-bit storage cells in [40]. When stride was equal to 2, taking 3×3 convolution as an example, data reuse could be performed in only four cycles. Fortunately, this situation was a small percentage of the calculation in the algorithm.

Data sharing and reuse were mainly applied to the convolutional layer, the normalized layer, and the fully connected layer. The remaining layers required a larger buffer read bandwidth because there was no overlapping of data.

5. Experiments and Results

In order to evaluate the efficiency and accuracy of our proposed algorithm framework and performance of the deep learning processor hardware architecture, we tested them separately by several experiments and evaluation indicators.

5.1. Experimental Settings

The evaluation experiment was divided into two parts. First, we used the publicly available remote sensing image dataset to evaluate the performance of the proposed algorithm framework in an open source deep learning framework. Then, we would implemented the hardware architecture of the deep learning processor on the FPGA and test its processing performance. The dataset, experiment environment, and test procedure used in the experiments are detailed below.

5.1.1. Dataset Description

We evaluated the performance of the proposed algorithm framework on the Northwestern Polytechnical University very high resolution remote sensing image dataset with 10 classes objects (NWPU VHR-10), which was constructed by Cheng, G. [1,25]. The resolution of these remote sensing images is between 0.5–2 m, and the average size is about 600 × 800 [25,45]. This dataset contained in total 800 VHR remote sensing images, which was divided into two parts, one was the positive image set and the other was the negative image set. The positive image set contained 650 remote sensing images, which were manually annotated with 757 airplanes, 302 ships, 655 storage tanks, 390 baseball diamonds, 524 tennis courts, 150 basketball courts, 163 ground track fields, 224 harbors, 124 bridges, and 477 vehicles. The negative image set was not used in this paper.

In our work, in order to obtain better detection results, we took the data augmentation on the dataset by cropping, flipping, rotating, scaling, and chromatic spatial transformation. During training, we randomly selected 20% as training set, 20% as validation set, and 60% as the test set.

5.1.2. Experiment Environment

To evaluate the performance of the proposed algorithm framework and the hardware architecture of deep learning processor, we leveraged the popular open source Caffe framework [52] and FPGA. For the evaluation of algorithm performance, we implemented the proposed algorithm framework on the Caffe, which executed on a 64-bit Ubuntu 16.04 PC with Intel i7-7700 CPU, 16GB memory and NVIDIA GeForce GTX1070Ti GPU. For the evaluation of the deep learning processor performance, we described the proposed hardware architecture using Verilog HDL and implemented it on Xilinx Zynq-XC7Z7100 FPGA.

5.1.3. Test Procedure

In the performance evaluation of the algorithm, we implemented the proposed CBFF-SSD algorithm framework on the Caffe, and then trained it with GPU. We used the pre-trained MobileNet-SSD model on VOC0712 [53] dataset for CBFF-SSD training, and then fine-tuned our model on NWPU VHR-10 dataset. We set batch size to 8 for 416 × 416 input during training, and set the learning rate at 10^{-3} for the first 60 k iterations, then decreased it to 10^{-4} for the next 40 k iterations, and 10^{-5} for the last 20 k iterations. The momentum and weight decay were set to 0.9 and 0.0005 respectively by using stochastic gradient descent (SGD). The performance of the proposed algorithm framework was compared with the newly trained CNN [25], rotation-invariant CNN (RICNN) [25], R-P-Faster R-CNN [45], and SSD [33]. For a fair and accuracy comparison, the detection accuracy, computational time, and the precision-recall curves (PRCs) were taken as the evaluation indexes.

For the evaluation of the performance of the hardware architecture of the deep learning processor, the proposed hardware architecture was implemented on the Xilinx XC7Z100 FPGA. In the specific implementation of the deep learning processor, we designed 16 NPUs to form the neural processing engine, and designed 8 × 8 PEs in each NPU. The feature map buffer contained eight buffer blocks, each contained eight banks, each bank with a capacity of 8 × 16 bit. The multiplier and adder in the processor were implemented by embedded digital signal processing slice (DSPs) in the FPGA. The feature maps buffer, parameter maps buffer, instruction buffer, and the look up table (LUT) in the activation function

pipeline were all implemented by block random access memory (BRAM). The resource utilization of the deep learning processor is shown in Table 2.

Table 2. Resource utilization of the deep learning processor.

Resource	BRAM (18 k)	DSP	FF	LUT
Available	1510	2020	554,800	277,400
Utilization	912	1152	156,238	136,892
Utilization (%)	60	57	28	49

The performance of the proposed hardware architecture of deep learning processor was compared with CPU and GPU platform, and the other state-of-the-art processor based on FPGA implementations. The technology, frequency, power, performance, power efficiency, and performance density were taken as the evaluation indexes.

5.2. Evalutaion Indicators

In order to quantitatively evaluate the performance of the proposed algorithm framework and the deep learning processor, the widely utilized evaluation indicators of average precision (AP), mean average precision (mAP), average running time per image, and precision–recall curves (PRCs) were adopted for the object detection algorithm framework, and the Giga operations per second (GOP/s), power consumption, power efficiency, and performance density were adopted for the deep learning processor.

5.2.1. Average Precision

The average precision (AP) is the average of the precision over the interval from recall = 0 and recall =1, which is also equal to the area under the precision-recall curve. The higher the AP value, the better the performance of the algorithm. The mean average precision (mAP) is another indicator, which reflects the average of the average precision (AP) of all categories in the dataset. The AP and mAP reflect the performance of the algorithm from the detection accuracy. The average running time per image reflects the execution efficiency of the algorithm, but it depends on the hardware that carries the algorithm.

5.2.2. Precision–Recall Curve

The precision and recall are two important indicators that evaluate the performance of the object detection algorithm from other perspectives. They are formulated as follows:

$$Precision = \frac{TP}{TP + FP}. \tag{10}$$

$$Recall = \frac{TP}{TP + FN}. \tag{11}$$

The precision refers to the proportion of true positive (TP) in all data that are predicted to be positive. The recall reflects to the proportion of data predicted to be true positive (TP) to all positive data. When the area overlap between the predicted box and the ground-truth box exceeds a threshold (e.g., 0.5), the detection map is considered to be a TP. Otherwise, the detection map is considered to be a false positive (FP). In addition, the detection is considered to be a false negative (FN) when the predicted boxes that overlap with ground-truth box but does not have the maximum overlap value. The PRCs describes the relationship between the precision and recall, the larger the area under the curve, the better the performance of the detector.

5.2.3. Processor Performance

The performance evaluation indicators of the deep learning processor mainly include the Giga operations per second (GOP/s), power consumption, power efficiency, and performance density. It reflects the processing performance of the deep learning processor. The power consumption is another indicator of the processor, which is particularly sensitive to embedded systems, especially for the on-board information processing system in my work. The power efficiency is the ratio of processing performance to power consumption, which reflects the processing performance of the same energy consumption. The performance density is defined as the number of arithmetic operations performed by one DSP slice in one cycle, which can better reflect the computing performance of deep learning processors with different hardware architectures based on FPGA implementation [56].

5.3. Test Results of Object Detection Algorithm Framework

Detection examples of proposed CBFF-SSD algorithm framework on the NWPU VHR-10 dataset are shown in Figure 9. The qualitative detection results of proposed CBFF-SSD algorithm framework for the ten categories of aircraft, ship, storage tank, baseball diamond, tennis court, basketball court, ground track field, harbor, bridge, and vehicle are shown in Figure 9, respectively. It can be seen from Figure 9 that the proposed CBFF-SSD algorithm framework shows good detection performance both for large objects such as storage tank, baseball diamond, tennis court, basketball court, ground track field, harbor, and bridge, as well as small objects such as airplane, ship, and vehicle.

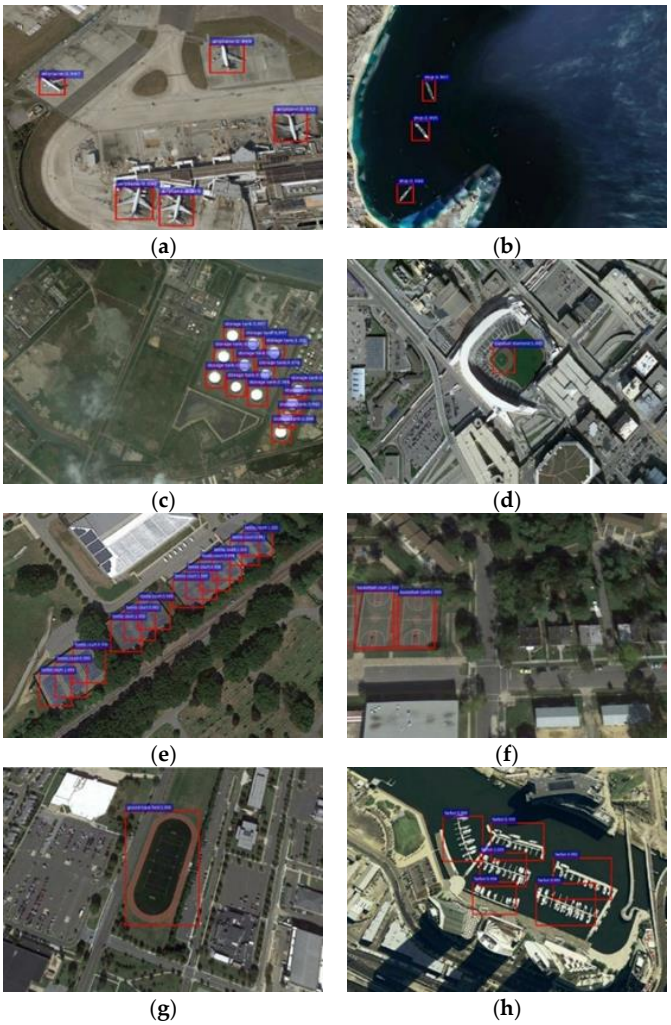

Figure 9. *Cont.*

(**i**) (**j**)

Figure 9. Object detection examples on NWPU VHR-10 dataset with the proposed CBFF-SSD algorithm framework. (**a**) Airplane; (**b**) ship; (**c**) storage tank; (**d**) baseball diamond; (**e**) tennis court; (**f**) basketball court; (**g**) ground track field; (**h**) harbor; (**i**) bridge and (**j**) vehicle.

In Table 3 and Figure 10, we quantitatively compared the performance of five different object detection algorithms by the AP value, mAP value, average running time per image, and PRCs, respectively. In each row of Table 3, the bold number indicates the best test result. It can be seen from the comparison of the test data of the five algorithms in Table 3 that the proposed algorithm had an advantage in the average accuracy of the detection of six classes of objects such as airplane, ship, tennis court, basketball court, ground track field, and vehicle. The detection accuracy of storage tank was lower than the newly trained CNN and RICNN algorithms. The proposed algorithm was slightly inferior to the SSD algorithm in the detection of tennis courts, harbor, and bridge. The average precision of the test data also verified that the proposed CBFF-SSD algorithm framework was more effective for detecting small objects such as an airplane, ship, and vehicle. In Table 3, it can be seen that the proposed CBFF-SSD algorithm obtained the best mean AP value of 0.9142 among all the object detection algorithms. It can also be seen from Table 3 that the proposed CBFF-SSD algorithm framework detected an image with an average running time of 0.0133s, which was faster than other algorithms. This is due to the efficient of regression-based SSD object detection framework and the reduced computational complexity of lightweight MobileNet backbone network. In addition, it was also inseparable from the support of high-performance GPU evaluation platform.

Table 3. Performance comparison of different algorithms.

	Newly Trained CNN [25]	**RICNN [25]**	**R-P-Faster R-CNN [45]**	**SSD [33]**	**CBFF-SSD**
Airplane	0.7014	0.8835	0.9060	0.9565	**0.9693**
Ship	0.6370	0.7734	0.7620	0.9356	**0.9426**
Storage tank	0.8433	**0.8527**	0.4030	0.6087	0.8095
Baseball diamond	0.8361	0.8812	0.9080	**0.9939**	0.9909
Tennis court	0.3546	0.4083	0.7970	0.8765	**0.9150**
Basketball court	0.4680	0.5845	0.7740	0.9200	**0.9264**
Ground track field	0.8120	0.8673	0.8800	0.9864	**0.9882**
Harbor	0.6228	0.6860	0.7620	**0.9460**	0.9159
Bridge	0.4538	0.6151	0.5750	**0.9704**	0.8968
Vehicle	0.4480	0.7110	0.6660	0.7447	**0.7878**
Mean AP	0.6177	0.7263	0.7430	0.8939	**0.9142**
Average running time per image (second)	8.770	8.770	0.150	0.0217	**0.0133**

As can be seen form Figure 10, the proposed CBFF-SSD algorithm framework shows better detection performance in most classes, especially in the classes of airplane, ship, tennis court, basketball court, ground track field, and vehicle. However, it was slightly inferior to the other algorithms in the classes of storage tank, tennis courts, harbor, and bridge.

By jointly analyzing the AP values, mAP value, average running time per image, the recall rate, and the PRCs, it can be seen that the proposed CBFF-SSD algorithm was superior to other algorithms in most classes of detection accuracy and detection efficiency.

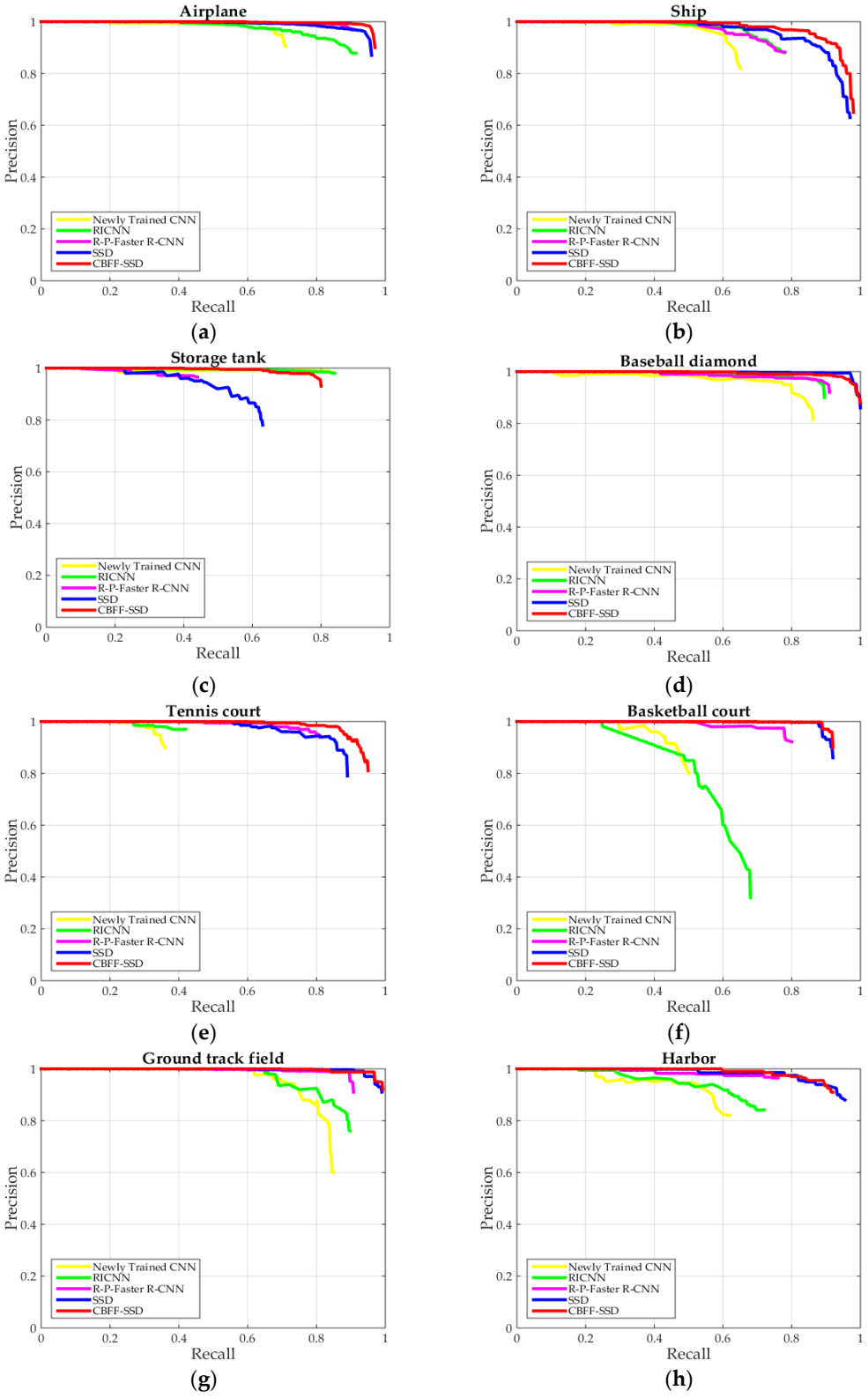

Figure 10. *Cont.*

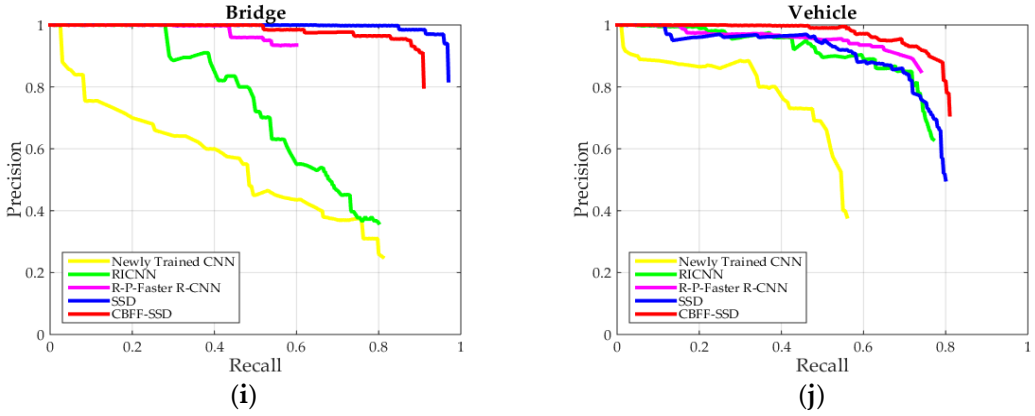

Figure 10. The precision–recall curves (PRCs) of the proposed method and other state-of-the-art methods for 10 classes object in NWPU VHR-10 dataset. (**a**) Airplane; (**b**) ship; (**c**) storage tank; (**d**) baseball diamond; (**e**) tennis court; (**f**) basketball court; (**g**) ground track field; (**h**) harbor; (**i**) bridge; and (**j**) vehicle.

5.4. Performance Evaluation of the Deep Learning Processor

In order to evaluate the performance of the processor fairly, we chose the proposed CBFF-SSD remote sensing object detection algorithm as a benchmark to compare the performance of the CPU, GPU and our deep learning processor when executing the algorithm. The evaluation results of the proposed processor, CPU, and GPU are shown in Table 3. The performance evaluation of CPU and GPU were based on the open source Caffe framework [52], which could choose the platform (CPU or GPU) to run. The CPU platform was Intel i7-7700 CPU @ 3.60 GHz with 16 GB DRAM. The GPU platform was NVIDIA GTX1070Ti GPU with 8 GB memory. The thermal design power (TDP) values of the CPU and GPU were 65 W and 180 W, separately. The total power of our deep learning processor was 19.52 W according to the power report by the Vivado design suite. We can see from Table 4 that the NVIDIA GTX1070Ti GPU had a great leading advantage in terms of computing performance, and its throughput was 1452 GOP/s. Our deep learning processor achieved the best power efficiency among all the platforms, reaching 23.20 GOP/s/W. The power efficient of our deep learning processor was 29.74 times that of CPU and 2.87 times that of GPU.

Table 4. Evaluation results on CPU, GPU, and our processor.

Platform	CPU	GPU	FPGA
Vendor	Intel i7-7700	NVIDIA GTX1070Ti	Xilinx XC7Z100
Technology (nm)	14	16	28
Frequency (MHz)	3600	1607	200
Power (W)	65	180	19.52
Benchmarks	CBFF-SSD	CBFF-SSD	CBFF-SSD
Latency (ms)	382.15	13.27	42.59
Performance (GOP/s)	51	1452	452.8
Power Efficiency (GOP/s/W)	0.78	8.07	23.20

In order to further evaluate the performance of the proposed deep learning processor, we compared it with the state-of-the-art FPGA-based deep learning processor or algorithm accelerator. In Table 5, we compared the performance of deep learning processors implemented on different FPGA platforms. For the sake of fairness, VGG16 was also selected as a benchmark for evaluating. The performance of the deep learning processor based on FPGA was closely related to the on-chip resources used. Therefore, it was inaccurate to simply evaluate the processor performance by GOP/s. We introduced the performance density indicator, which is related to the GOP/s, the on-chip DSPs resources used, and the operating frequency [56]. Performance density can better reflect the computing performance

of deep learning processors with different hardware architectures based on FPGA implementation. Note that a MAC (multiply and accumulate) was counted as two operations. As can be seen from Table 5, our processor operated at 200 MHz, which was higher than other implements. Our processor used 1152 on-chip DSPs in the implementation, achieving 452.8 GOP/s processing performance, and its performance density was 1.97 OP/DSP/cycle. Although our processor did not achieve the highest value of performance term, its performance density was the best among all the processors compared [54–56].

Table 5. Comparisons with previous implementations.

	Ref. [54]	**Ref. [55]**	**Ref. [56]**	**Ours**
FPGA Chip	Xilinx XC7Z045	Arria10 GX1150	Xilinx XC7VX690T	Xilinx XC7Z100
Frequency (MHz)	150	150	120	200
Precision	16-bit fixed	8–16 fixed	fixed	16-bit fixed
CNN Model	VGG16	VGG16	VGG16	VGG16
DSPs	780	3036	3595	1152
Performance (GOP/s)	187.80	645.25	691.6	452.8
Performance Density (OP/DSP/cycle)	1.61	1.42	1.60	1.97

6. Discussion

We adopted the NWPU VHR-10 dataset to train, validate, and test the proposed CBFF-SSD algorithm framework, which achieved considerable results in the object detection of very high resolution optical remote sensing images. In order to fully verify the effectiveness of the algorithm proposed in this paper and the deep learning hardware architecture, we carried out experiments on small object detection, large-scale remote sensing image object detection, and deep learning processor performance comparison, and analyzed the experimental results.

6.1. Small Objects Detection Result and Analysis

The algorithm proposed in this paper was optimized on the SSD [33] algorithm. The superiority in computational efficiency was verified by the average running time of each image in Table 3. In order to show the effect of the proposed CBFF-SSD algorithm on small object detection, we compared the detection of small objects of different classes by the two algorithm and the results are shown in Figure 11.

By comparing the results of the two algorithms in Figure 11 on the detection of small objects such as the storage tank, airplane, and ship, it can be seen that both SSD algorithm and CBFF-SSD algorithm had achieved good results for the detection of storage tank. For small and dense objects detection such as airplane and ships, the CBFF-SSD algorithm was superior to the SSD algorithm. The experimental results verified the effectiveness of the feature fusion unit that adds context feature fusion to the SSD algorithm structure.

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

Figure 11. *Cont.*

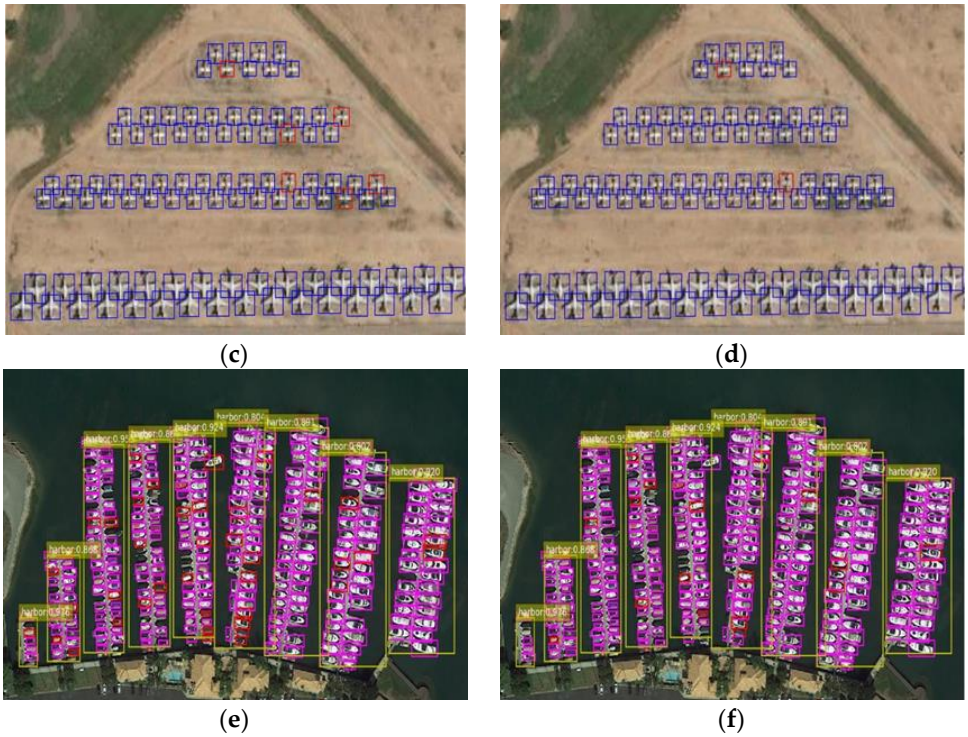

Figure 11. Small object detection results comparison. (**a**,**c**,**e**) are the detection results of the SSD algorithm; (**b**,**d**,**f**) are the detection results of our proposed CBFF-SSD algorithm. The red rectangle indicates incorrect detection.

6.2. Large-Scale Remote Sensing Image Object Detection Results and Analysis

In order to show the overall detection effect, we performed inferences on large-scale remote sensing images and the results are shown in Figure 12.

We tested the proposed algorithm and deep learning processor through large-scale remote sensing images. After the remote sensing image was calculated by the processor, the coordinates, categories and confidence were marked by the software. It should be noted that for the testing of large-scale remote sensing images, we could not directly scale them as input images. We processed the original image by dividing and setting the overlap rate of 10%, and adopted non-maximum suppression (NMS) to process the detection results of the overlap part.

It can be seen from the experimental results that the proposed algorithm and deep learning processor had a good object detection effect for large-scale remote sensing images. However, it can be seen from the experimental results that illumination and shadow had a certain influence on the object detection. Meanwhile, the detection results of objects that did not appear in the training set were unsatisfactory (for example, helicopters without wings). These problems could be improved by training a large number of samples.

6.3. Discussion on Processor Implementation and Model Compression Methods

In the current research, there were many implementations of deep learning processors, such as the general-purpose processor, FPGAs, application specific integrated circuits (ASICs), etc. We compared the different implementations in Table 6.

By comparison we could see that the ASIC-implemented deep learning processors had higher operating frequency, better processing performance, and lower power consumption. This provided us with ideas and directions for the next step in improving the performance of our proposed deep learning processor.

Figure 12. Detection results on large-scale remote sensing images. (**a**) Baseball diamond, ground track filed, tennis court, and vehicle; (**b**) vehicle, and tennis court; (**c**) harbor, and ship; (**d**) storage tank, ship; (**e**) baseball diamond, basketball court, and vehicle; (**f**) vehicle; and (**g**) airplane.

We were currently using a refined model way to compress model parameters and reduce the amount of computation. It can be seen from Table 1 that the number of parameters of the proposed CBFF-SSD algorithm framework was 56.09% of the SSD algorithm, and the calculation amount was

only 17.56% of the SSD algorithm. There are also model compression methods such as weight pruning, weight sparse, processing data quantification, and so on. In particular, the method of processing data quantization can achieve better model compression without changing the model structure. This method includes binary neural networks [57], quantized neural networks (n-bits) [58], integer CNNs [59], and so on. Google's TPU [42] uses 8-bit integer data for superior processing performance and negligible error relative to floating-point production in the engineering applications. Our current model compression strategy could be combined with strategies for processing data quantification to achieve better model compression. This will be the direction of our next step in optimizing the algorithm model.

Table 6. Comparison of deep learning processors with different implementations.

	Jetson AGX Xavier [60]	**TPU [42]**	**ShiDianNao [40]**	**Ours**
Implementations	ASIC (ARM+GPU)	ASIC	ASIC	FPGA
Frequency (MHz)	1370	700	1000	200
Precision	8-bit integer	8-bit integer	16-bit fixed	16-bit fixed
Performance (GOP/s)	22,000	92,000	194	452.8
Power (W)	10/15/30	75	0.320	19.52

7. Conclusions

In this paper, an efficient context-based feature fusion SSD (CBFF-SSD) framework and hardware architecture of deep learning processor with multi-processor clusters were proposed to object detection in remote sensing images on space-borne or airborne. The design of the CBFF-SSD framework fully considers small object detection, detection accuracy, and efficiency. The deep learning processor uses multiple neural processing units (NPUs) composed of 2-D processing elements (PEs) to simultaneously calculate multiple output feature maps. The parallel architecture, hierarchical on-chip storage organization, and the register designed in the PE make the calculation of the processor more efficient.

A comparison test with five algorithms on the NWPU VHR-10 dataset shows that our algorithm framework had an advantage in the average accuracy of the detection of six classes of objects, and it was superior to other algorithms in terms of the mean AP value and average running timer per image. The effectiveness of the proposed CBFF-SSD algorithm was verified by small object detection experiments and large-scale remote sensing image object detection experiments. The proposed processor implemented in FPGA was compared with CPU, GPU, and other FPGA-based deep learning processor. In the comparative test of general-purpose processors, our deep learning processor achieved the best power efficiency, which was 29.74 times that of CPU and 2.87 times that of GPU. In comparison with the state-of-the-art FPGA-based deep learning processors, although our processor did not achieve the highest value of performance term, its performance density was the best among all the processors compared.

In the future, we would improve our algorithm frameworks in terms of identifying the classes of the object and accuracy to realize more effective network for remote sensing images object detection. We would use a strategy for processing data quantification to optimize the proposed algorithm framework. Furthermore, we would implement the proposed hardware architecture of deep learning processor in application specific integrated circuit (ASIC) way to achieve higher computing performance and lower power consumption.

Author Contributions: Conceptualization, L.L. and S.Z.; methodology, L.L.; software, L.L. and J.W.; validation, L.L. and J.W.; formal analysis, L.L.; investigation, L.L. and J.W.; resources, S.Z.; data curation, L.L.; writing—original draft preparation, L.L.; writing—review and editing, S.Z. and J.W.; supervision, S.Z.

Funding: This research received no external funding.

Acknowledgments: This research is supported by Northwestern Polytechnical University and Beijing Institute of Microelectronics Technology. The authors would like to thank all the teachers and colleagues who provided inspirations and helpful suggestions.

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

References

1. Cheng, G.; Han, J. A survey on object detection in optical remote sensing images. *ISPRS J. Photogramm. Remote Sens.* **2016**, *117*, 11–28. [CrossRef]
2. Xu, Y.; Zhu, M.; Li, S. End-to-end airport detection in remote sensing images combining cascade region proposal networks and multi-threshold detection networks. *Remote Sens.* **2018**, *10*, 1156. [CrossRef]
3. Zhu, M.; Xu, Y.; Ma, S.; Li, S.; Ma, H.; Han, Y. Effective airplane detection in remote sensing images based on multilayer feature fusion and improved nonmaximal suppression algorithm. *Remote Sens.* **2019**, *11*, 1062. [CrossRef]
4. Leitloff, J.; Hinz, S.; Stilla, U. Vehicle detection in very high resolution satellite images of city areas. *IEEE Trans. Geosci. Remote Sens.* **2010**, *48*, 2795–2806. [CrossRef]
5. He, H.; Yang, D.; Wang, S.C.; Wang, S.Y.; Li, Y. Road extraction by using atrous spatial pyramid pooling integrated encoder-decoder network and structural similarity loss. *Remote Sens.* **2019**, *11*, 1015. [CrossRef]
6. Zhang, J.; Lin, X.; Liu, Z.; Shen, J. Semi-automated road tracking by template matching and distance transformation in urban areas. *Int. J. Remote Sens.* **2011**, *32*, 8331–8347. [CrossRef]
7. Liu, G.; Sun, X.; Fu, K.; Wang, H. Interactive geospatial object extraction in high resolution remote sensing images using shape-based global minimization active contour model. *Pattern Recog. Lett.* **2013**, *34*, 1186–1195. [CrossRef]
8. Ok, A.O.; Senaras, C.; Yuksel, B. Automated detection of arbitrarily shaped buildings in complex environments from monocular VHR optical satellite imagery. *IEEE Trans. Geosci. Remote Sens.* **2013**, *51*, 1701–1717. [CrossRef]
9. Leninisha, S.; Vani, K. Water flow based geometric active deformable model for road network. *ISPRS J. Photogramm. Remote Sens.* **2015**, *102*, 140–147. [CrossRef]
10. Peng, J.; Liu, Y. Model and context-driven building extraction in dense urban aerial images. *Int. J. Remote Sens.* **2005**, *26*, 1289–1307. [CrossRef]
11. Hussain, M.; Chen, D.; Cheng, A.; Wei, H.; Stanley, D. Change detection from remotely sensed images: From pixel-based to object-based approaches. *ISPRS J. Photogramm. Remote Sens.* **2013**, *80*, 91–106. [CrossRef]
12. 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]
13. Feizizadeh, B.; Tiede, D.; Rezaei Moghaddam, M.H.; Blaschke, T. Systematic evaluation of fuzzy operators for object-based landslide mapping. *South East. Eur. J. Earth Obs. Geomat.* **2014**, *3*, 219–222.
14. Lowe, D.G. Object recognition from local scale-invariant features. In Proceedings of the 7th IEEE International Conference on Computer Vision, Kerkyra, Greece, 20–27 September 1999; pp. 1150–1157.
15. Dalal, N.; Triggs, B. Histograms of oriented gradients for human detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, San Diego, CA, USA, 21–23 September 2005; pp. 886–893.
16. Sun, H.; Sun, X.; Wang, H.; Li, Y.; Li, X. Automatic target detection in high-resolution remote sensing images using spatial sparse coding bag-of-words model. *IEEE Geosci. Remote Sens. Lett.* **2012**, *9*, 109–113. [CrossRef]
17. Zhu, C.; Zhou, H.; Wang, R.; Guo, J. A novel hierarchical method of ship detection from spaceborne optical image based on shape and texture features. *IEEE Trans. Geosci. Remote Sens.* **2010**, *48*, 3446–3456. [CrossRef]
18. Mountrakis, G.; Im, J.; Ogole, C. Support vector machines in remote sensing: A review. *ISPRS J. Photogramm. Remote Sens.* **2011**, *66*, 247–259. [CrossRef]
19. Collins, M.; Schapire, R.E.; Singer, Y. Logistic regression, adaboost and bregman distances. *Mach. Learn.* **2002**, *48*, 253–285. [CrossRef]
20. Ali, A.; Olaleye, O.G.; Bayoumi, M. Fast region-based DPM object detection for autonomous vehicles. In Proceedings of the 2016 IEEE 59th International Midwest Symposium on Circuits and Systems, Abu Dhabi, United Arab Emirates, 16–19 October 2016; pp. 1–4.
21. Wegner, J.D.; Haensch, R.; Thiele, A.; Soergel, U. Building detection from one orthophoto and high-resolution InSAR data using conditional random fields. *IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens.* **2011**, *4*, 83–91. [CrossRef]

22. Cheng, G.; Han, J.; Zhou, P.; Yao, X.; Zhang, D.; Guo, L. Sparse coding based airport detection from medium resolution Landsat-7 satellite remote sensing images. In Proceedings of the 2014 3rd International Workshop on Earth Observation and Remote Sensing Applications, Changsha, China, 11–14 June 2014; pp. 226–230.
23. Mokhtarzade, M.; Zoej, M.V. Road detection from high-resolution satellite images using artificial neural networks. *Int. J. Appl. Earth Observ. Geoinform.* **2007**, *9*, 32–40. [CrossRef]
24. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems, Lake Tahoe Nevada, NV, USA, 3–6 December 2012; pp. 1097–1105.
25. Cheng, G.; Zhou, P.; Han, J. Learning rotation-invariant convolutional neural networks for object detection in VHR optical remote sensing images. *IEEE Trans. Geosci. Remote Sens.* **2016**, *54*, 7405–7415. [CrossRef]
26. Wang, G.; Chen, J.; Gao, F.; Wu, J. Research on the infrastructure target detection of remote sensing image based on deep learning. *Radio Eng.* **2018**, *48*, 219–224.
27. Jiao, L.; Zhao, J.; Yang, S.; Liu, F. *Deep Learning, Optimization and Recognition*, 1st ed.; Tsinghua University Press: Beijing, China, 2017; pp. 341–367.
28. Girshick, R.; Donahue, J.; Darrelland, T.; Malik, J. Rich feature hierarchies for object detection and semantic segmentation. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, Los Alamitos, CA, USA, 23–28 June 2014; pp. 580–587.
29. Girshick, R. Fast R-CNN. In Proceedings of the 2015 IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1440–1448.
30. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *39*, 1137–1149. [CrossRef] [PubMed]
31. Lin, T.Y.; Dollar, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature pyramid networks for object detection. In Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2016; pp. 936–944.
32. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detetction. *arXiv* **2015**, arXiv:1506.02640.
33. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. SSD: Single Shot MultiBox Detector. In Proceedings of the 14th European Conference on Computer Vision, Amsterdam, The Netherlands, 8–16 October 2016; pp. 21–37.
34. Hennessy, J.L.; Patterson, D.A. *Computer Architecture: A Quantitative Approach*, 6th ed.; Morgan Kaufman: Cambridge, MA, USA, 2019; pp. 539–617.
35. Farabet, C.; Poulet, C.; Han, J.Y.; Lecun, Y. CNP: An FPGA based processor for convolutional networks. In Proceedings of the 2009 International Conference on Field Programmable Logic and Applications, Prague, Czech Republic, 31 August–2 September 2009; pp. 32–37.
36. Farabet, C.; Martini, B.; Corda, B.; Akselrod, P.; Culurciello, E.; Lecun, Y. NeuFlow: A runtime reconfigurable dataflow processor for vision. In Proceedings of the 2011 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, Colorado Springs, CO, USA, 20–25 June 2011; pp. 109–116.
37. Peemen, M.; Setio, A.A.A.; Mesman, B.; Corporaal, H. Memory-centric accelerator design for convolutional neural networks. In Proceedings of the 2013 IEEE 31st International Conference on Computer Design, Asheville, NC, USA, 6–9 October 2013; pp. 13–19.
38. 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.
39. Chen, T.; Du, Z.; Sun, N.; Wang, J.; Wu, C.; Chen, Y.; Temam, O. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM Sigplan Not.* **2014**, *49*, 269–284.
40. Du, Z.; Fasthuber, R.; Chen, T.; Ienne, P.; Li, L.; Luo, T.; Feng, X.B.; Chen, Y.J.; Temam, O. ShiDianNao: Shifting vision processing closer to the sensor. *SIGARCH Comput. Archit. News* **2015**, *43*, 92–104. [CrossRef]
41. 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.
42. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News* **2017**, *45*, 1–12. [CrossRef]

43. Li, L.; Zhang, S.B.; Wu, J. Design and realization of deep learning coprocessor oriented to image recognition. In Proceedings of the 2017 17th IEEE International Conference on Communication Technology, Chengdu, China, 27–30 October 2017; pp. 1553–1559.
44. Chang, J.W.; Kang, K.W.; Kang, S.J. An energy-efficient FPGA-based deconvolutional neural networks accelerator for single image super-resolution. *IEEE Trans. Circuits Sys. Video Tech.* **2018**. [CrossRef]
45. Han, X.; Zhong, Y.; Zhang, L. An efficient and robust integrated geospatial object detection framework for high spatial resolution remote sensing imagery. *Remote Sens.* **2017**, *9*, 666. [CrossRef]
46. Etten, A.V. You Only Look Twice: Rapid Multi-Scale Object Detection in Satellite Imagery. *arXiv* **2018**, arXiv:1805.09512.
47. Zhang, X.; Zhu, K.; Chen, G.; Tan, X.; Zhang, L.; Dai, F.; Liao, P.; Gong, Y. Geospatial object detection on high resolution remote sensing imagery based on double multi-scale feature pyramid network. *Remote Sens.* **2019**, *11*, 755. [CrossRef]
48. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv* **2017**, arXiv:1704.04861.
49. Fu, C.Y.; Liu, W.; Ranga, A.; Tyagi, A.; Berg, A.C. DSSD: Deconvolutional Single Shot Detector. *arXiv* **2017**, arXiv:1701.06659.
50. Li, L.; Zhang, S.B.; Wu, J. An efficient hardware architecture for activation function in deep learning processor. In Proceedings of the 2018 3rd IEEE International Conference on Image, Vision and Computing, Chongqing, China, 27–29 June 2018; pp. 911–918.
51. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 6–11 July 2015; pp. 448–546.
52. Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; Darrell, T. Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM International Conference on Multimedia, Orlando, FL, USA, 3–7 November 2014; pp. 675–678.
53. Everingham, M.; Gool, L.; Williams, L.K.; Winn, J.; Zisserman, A. The pascal visual object classes (VOC) challenge. *IJCV* **2010**, *88*, 303–338. [CrossRef]
54. Qiu, J.; Wang, J.; Yao, S.; Guo, K.; Li, B.; Zhou, E.; Yu, J.; Tang, T.; Xu, N.; Song, S. Going deeper with embedded FPGA platform for convolutional neural network. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 21–23 February 2016; pp. 26–35.
55. 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 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; pp. 45–54.
56. 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]
57. Courbariaux, M.; Bengio, Y.; David, J.P. Binaryconnect: Training deep neural networks with binary weights during propagations. In Proceedings of the 28th International Conference on Neural Information Processing Systems, Montreal, Canada, 7–12 December 2015; pp. 3123–3131.
58. 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.* **2018**, *18*, 1–30.
59. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 2704–2713.
60. Jetson AGX Xavier. Available online: https://developer.nvidia.com/embedded/jetson-agx-xavier (accessed on 12 December 2018).

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).