




Article

Hardware Acceleration of Satellite Remote Sensing Image Object Detection Based on Channel Pruning

Yonghui Zhao , Yong Lv  and Chao Li * 

College of Computer and Control Engineering, Northeast Forestry University, Harbin 150040, China; hero9968@nefu.edu.cn (Y.Z.); 2021116215@nefu.edu.cn (Y.L.)

* Correspondence: lichaonefuzyz@nefu.edu.cn; Tel.: +86-158-4650-7221

Abstract: Real-time detection of satellite remote sensing images is one of the key technologies in the field of remote sensing, which requires not only high-efficiency algorithms, but also low-power and high-performance hardware deployment platforms. At present, the image processing hardware acceleration platform mainly uses an image processing unit (GPU), but the GPU has the problem of large power consumption, and it is difficult to apply to micro-nano satellites and other devices with limited volume, weight, computing power, and power consumption. At the same time, the deep learning algorithm model has the problem of too many parameters, and it is difficult to directly deploy it on embedded devices. In order to solve the above problems, we propose a YOLOv4-MobileNetv3 field programmable gate array (FPGA) deployment scheme based on channel layer pruning. Experiments show that the acceleration strategy proposed by us can reduce the number of model parameters by 91.11%, and on the aerial remote sensing dataset DIOR, the average accuracy of the design scheme in this paper reaches 82.61%, the FPS reaches 48.14, and the average power consumption is 7.2 W, which is 317.88% FPS higher than the CPU and reduces the power consumption by 81.91%. Compared to the GPU, it reduces power consumption by 91.85% and improves FPS by 8.50%. Compared with CPUs and GPUs, our proposed lightweight algorithm model is more energy-efficient and more real-time, and is suitable for application in spaceborne remote sensing image processing systems.



Citation: Zhao, Y.; Lv, Y.; Li, C.

Hardware Acceleration of Satellite Remote Sensing Image Object Detection Based on Channel Pruning. *Appl. Sci.* **2023**, *13*, 10111. <https://doi.org/10.3390/app131810111>

Academic Editors: Feng Gao, Jin Zheng and Qizhi Xu

Received: 14 August 2023

Revised: 2 September 2023

Accepted: 5 September 2023

Published: 8 September 2023



Copyright: © 2023 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 (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: satellite remote sensing; FPGA; YOLOv4; pruning; quantify

1. Introduction

Remote sensing image target detection has been extensively applied in various fields, including land resources planning [1], glacier change monitoring [2], disaster prevention and relief [3], military national defense [4], urban safety supervision [5], forestry vegetation monitoring [6], and many others. It holds significant scientific research value and offers broad application prospects. Remote sensing image object detection can be broadly categorized into traditional human-based feature extraction methods and deep learning-based methods [7]. Traditional methods involve manually designing effective feature extractors to perform target detection. On the other hand, deep learning methods employ convolutional neural networks (CNNs) to extract features in a data-driven manner, allowing for continuous learning and achieving object detection tasks. With the continuous advancement and maturation of deep learning theory and technology, object detection performance based on deep learning has surpassed traditional methods by a significant margin. It has also gained broader applications in various domains, including image classification and object detection tasks. Deep learning-based general object detection algorithms include anchor-based methods. Prominent examples of anchor-based algorithms are one-stage detection algorithms like SSD [8]; the YOLO series [9]; and two-stage detection methods like Faster R-CNN [10] and Mask R-CNN [11]. Each of these algorithms has its own strengths and offers distinct advantages in practical engineering applications. The two-stage detection algorithm divides the object detection process into two stages. First, it generates

candidate regions and then refines their positions before classifying them. The advantages of the two-stage detection algorithm are its low detection and recognition error rates, as well as a low rate of false positive detections. However, it has the drawback of slower detection speed, making it less suitable for real-time object detection tasks in videos or images. Therefore, two-stage detection algorithms are commonly used in applications that require high-precision object detection, such as face recognition, medical image analysis, and so on. In contrast, the one-stage detection algorithm does not involve a separate stage for generating candidate regions. Instead, it directly generates the category probability and position coordinate values of the objects. The final detection result can be obtained after a single pass, resulting in faster detection compared to the two-stage algorithm. Therefore, single-stage detection algorithms are commonly used in scenarios that require fast processing, such as real-time video monitoring, autonomous driving, and so on. In recent years, there have been significant advancements in anchor-free object detection algorithms, such as CenterNet [12] and FCO (fully convolutional one-stage object detection) [13]. These algorithms revolutionize the traditional approach by eliminating the dependence on predefined anchors.

Target detection in remote sensing images has garnered significant attention in the field. The advancement of remote sensing satellite technology in recent years has spurred research in spaceborne remote sensing image target detection systems. When applying target detection in spaceborne remote sensing images to practical engineering, it becomes crucial to consider not only detection accuracy but also real-time capabilities of the network model and power consumption of the deployment platform. These factors play a vital role in ensuring efficient and effective target detection in the context of remote sensing imagery. To illustrate, let us consider the power constraints of small remote sensing satellites like CubeSats. These satellites typically operate within a power budget of only 2–8 W [14]. In contrast, GPU platforms such as A100, RTX 2080Ti, RTX 3090, Tesla V100, and others have power consumption levels that far exceed the carrying capacity of the satellite. This stark difference in power requirements poses a significant challenge when it comes to deploying these GPU platforms directly on board. Therefore, it becomes essential to explore alternative approaches or optimize existing algorithms to ensure efficient and effective target detection within the power limitations of small remote sensing satellites. As a result, the conventional approach in satellite remote sensing image processing involves transmitting the captured images from satellites or aerial drones to GPU platforms on the ground for further processing. Due to the power limitations of small remote sensing satellites, it is more practical to offload the computational tasks to powerful GPU platforms that are readily available on the ground. This approach allows for the utilization of high-performance computing resources while overcoming the power constraints of the satellite. By transmitting the images to a GPU platform on the ground, researchers can leverage the capabilities of these platforms to process the remote sensing data efficiently and accurately. However, the process of downloading image data to the ground for processing introduces significant delays in data processing, resulting in reduced system efficiency. This limitation poses challenges in meeting real-time detection requirements [15]. To address this issue, researchers have begun exploring the deployment of convolutional neural network (CNN) models on application-specific integrated circuits (ASICs) and field programmable gate arrays (FPGAs) [16]. These specialized hardware platforms offer the potential for accelerating and optimizing the processing of image data, thereby improving overall system efficiency. By leveraging the parallel computing capabilities of ASICs and FPGAs, researchers aim to overcome the latency associated with transmitting data to the ground, enabling more efficient real-time detection. This research direction can overcome the challenges posed by traditional GPU processing and meet the stringent requirements of real-time detection in remote sensing applications. Among them, FPGAs have emerged as an alternative to GPUs in spaceborne scenarios due to their advantages such as low power consumption, high performance, parallel computing capabilities, programmability, and

customization. They possess the characteristics of high flexibility and lower cost compared to ASICs [17].

When deploying CNN models on embedded devices, it is common to trade off some accuracy in order to improve detection speed and reduce the number of model parameters. Currently, commonly used methods include model compression [18] and the utilization of more lightweight network models [19] to decrease the model size. These techniques enable the deployment of CNN models on embedded devices with reduced computational complexity and memory footprint, while still maintaining acceptable performance levels.

To cater to the real-time target detection needs of remote sensing images in spaceborne applications, while addressing challenges related to deep learning model parameters, computational complexity, and hardware platform deployment, this paper presents an efficient lightweight algorithm called YOLOv4-MobileNetv3. Additionally, channel pruning is performed on this algorithm. Furthermore, the Xilinx Vitis AI tool chain is employed to quantize, compile, and deploy the pruned network model at the edge. The primary contributions of this paper can be summarized as follows:

- **Significant Reduction in Model Parameters:** The acceleration strategy we proposed achieves a remarkable reduction in the number of model parameters, with the parameter size being only 0.09 times that of the original YOLOv4. This reduction not only enhances the efficiency of model storage but also contributes to faster inference times.
- **Proposed Deployment Scheme:** We have developed a novel deployment scheme for YOLOv4-MobileNetv3 on an FPGA using channel layer pruning. The proposed deployment scheme addresses the challenges of high power consumption and real-time performance in hardware deployment platforms for object detection, enabling efficient implementation on resource-constrained hardware.
- **Superiority in Energy Efficiency and Real-Time Processing:** Our lightweight algorithm model outperforms CPUs and GPUs in terms of both energy efficiency and real-time performance. This makes our proposed solution highly suitable for spaceborne remote sensing image processing systems, where the efficient utilization of resources is crucial.

The remaining sections of this paper are structured as follows: Section 2 provides an overview of related work, including common lightweight network model structures, compression techniques for network models, and FPGA deployment of CNN models. Section 3 presents the experimental design for this study. It begins by introducing image enhancement techniques, followed by the detailed description of the improved YOLOv4-MobileNetv3 network model structure, then introduces the related experimental work of sparse training and channel pruning for YOLOv4-MobileNetv3. Finally, it covers the process of utilizing Vitis AI for quantizing the pruned network model and compiling it for deployment on FPGA platforms. Section 4 outlines the experimental setup and analyzes the results. It evaluates and compares the performance of the improved network model against the mainstream network model. Additionally, it analyzes and compares the performance of the network model deployed on different hardware platforms. Lastly, Section 5 provides a summary of the entire paper. It discusses the key findings and future research directions in this field.

2. Related Work

2.1. Lightweight Network

Using a lightweight network as the backbone feature extraction network can better deploy CNN models to embedded devices. At present, the mainstream lightweight networks mainly include the ShuffleNet series [20], NasNet [21], GhostNet [19], SqueezeNet, WeightNet, MicroNet, the EfficientNet series, the MobileNet series, etc. SqueezeNet reduces the number of input channels for pooling and images in the network model by employing 1×1 convolution and packet convolution [22]; The SqueezeNext network structure borrows from the residual structure and uses the separation convolution reduction operation [23]. Figure 1 shows the network architecture diagram of ShuffleNet, MobileNetv1, MobileNetv2, and WeightNet. In the ShuffleNet series of networks, the ShuffleNetv1 network proposes

a channel shuffle operation, which can not only reduce the amount of network computation, but also increase the convolution dimension; The ShuffleNet2 network proposes a channel split operation, which divides the input features into two parts, which can achieve the effect of feature reuse while achieving the effect of accelerating the network [24]; The WeightNet network integrates CondConv and SENet on the weight space, and directly generates convolution kernel weights by adding a packet fully connected layer after the activation vector, thereby improving the computational efficiency [25]; MobileNetv1 is a lightweight network developed by Google for the deployment of neural networks on the mobile terminal, using depthwise convolution (DWC) and pointwise convolution (PWC) to build deep separable convolution greatly improves the running speed, but also bringing a large amount of information loss of images [26]; MobileNetv2 network is a lightweight convolutional neural network launched by Google in 2018, which has two innovations compared to the MobileNetv1 network: inverted residuals (inverted residual structure) and linear bottlenecks. The inverted residual structure is first on the input feature matrix, through 1×1 convolution to upgrade the dimension and increase the channel size. Then, the DW convolution kernel of 3×3 is used for convolution processing, and finally the dimensionality reduction is carried out by the convolution kernel of 1×1 [27].

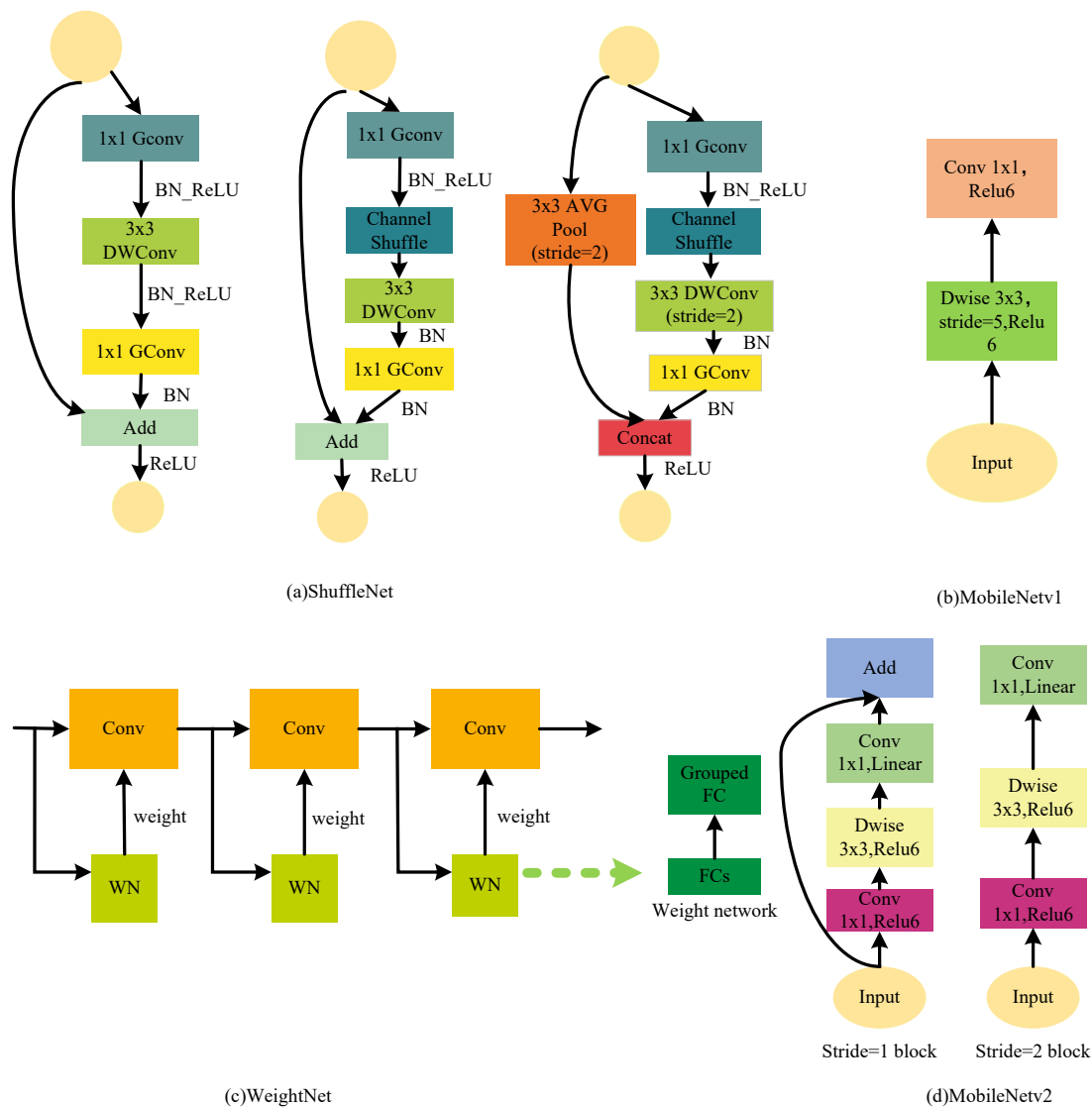


Figure 1. Comparison of convolutional blocks with different network structures.

2.2. Model Pruning and Quantization

Since the popularization of deep convolutional neural networks, convolutional neural networks have gradually developed in the direction of making deeper and more complex network structures. By increasing the number of layers of the deep convolutional data network, stronger representation ability is obtained, which improves the accuracy and generalization ability of the model [28]. However, the complex network structure also brings problems such as large number of parameters, high computational complexity, and slow detection speed, and such deep models are difficult to deploy from bulky servers to embedded and mobile devices with limited computing power and memory space.

Some researchers have discovered through studies of the human brain that neuronal synapses initially increase and subsequently decrease in individuals as they continue to develop. This phenomenon has given rise to the concept of pruning. Neural network pruning has demonstrated the ability to reduce model parameters by over 80% while significantly enhancing inference performance, with minimal impact on accuracy. Jonathan et al. proposed the lottery hypothesis to find out the winning lottery tickets of large networks by pruning experiments on the weights of the smallest magnitude, and the winning ticket weights will return to the initial value before training, thus proving that the luck of the winning ticket comes from initialization [29]. Li et al. proposed a pruning method based on the L1 norm, and used the L1 norm to prune unimportant weights. This approach does not introduce additional regularization, but has limitations due to the L1 norm and is not applicable to some special weights [30]. Han et al. [31] propose a method to use model training to learn the connection and prune the model small weight join, which reduces the amount of storage required by AlexNet by 233.1 M without loss of accuracy. Liu et al. proposed a simple and efficient pruning scheme that advances the scale factor in the batch normalized (BN) layer towards 0 by L1 regularization, and prunes unimportant channels by assessing the importance of the λ parameters of the BN layer. This method has the characteristics of generalization accuracy, and a certain accuracy can be restored after fine-tuning the pruned model [32]. Molchanov et al. proposed a method to use Taylor expansion to evaluate the influence of neurons on loss values, and then prune small neurons through continuous iterative pruning [33].

Model quantification came into interest in the 2010s. Model quantization can convert the floating-point algorithm of the neural network model into fixed-point numbers, and taking 8-bit quantization as an example, the 32-bit floating-point number model can be compressed by 75% without reducing the model accuracy [34]. Quantization can effectively accelerate model inference and is important for model deployment. Rastegari et al. [35] approximated and simplified the model weights by binary representation of CNN model weights, and achieved comparable accuracy with standard networks in ImageNet classification while reducing floating-point number operations. Han et al. [31] proposed a three-stage model compression method in the technology of the predecessor technique: pruning, training quantization, and Hoffman coding, which reduced the number of model weight parameters by tens of times. In the network quantization and weight sharing section, the network model is compressed by reducing the number of bits required for weights. Their method reduces the total number of weights by connecting the same weights and fine-tuning the weights being shared. The 4×4 matrices in the upper left and lower right corners of Figure 2 represent the weight matrix and gradient matrix, respectively, and the four colors green, light green, yellow, and dark blue represent four different bins where the weights are quantified, and they share the same values for the weights of the same bins. At the time of update, all gradients are added in groups by bin, then multiplied by the learning rate (lr) and subtracted from the last shared centroid. This allows the COTV layer in the model to be quantized by 8 bits, reducing memory while maintaining accuracy. Courbariaux et al. [36] proposed a training quantization method for binary networks, quantized the BinaryConnect activation value to 1 bit, and replaced addition and multiplication operations with bitwise operations, which effectively reduced the memory consumption of memory. In 2022, Wang et al. [37] proposed a quantitative

improvement strategy based on learnable lookup tables (LLT), which greatly reduced the computational cost by converting quantification into a lookup process.

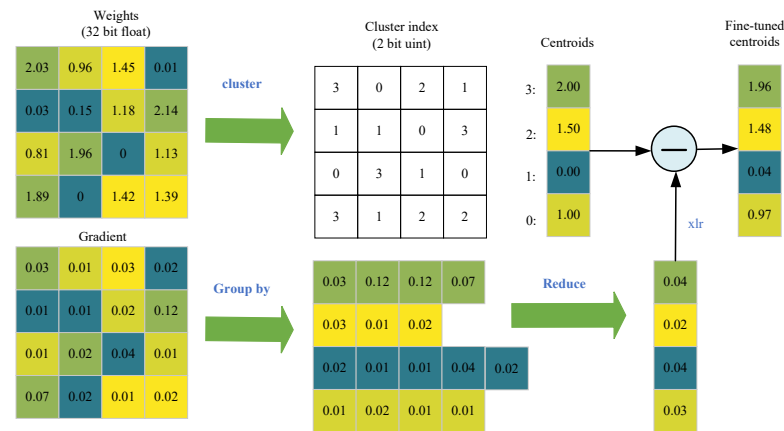


Figure 2. Weight sharing.

2.3. FPGA Acceleration of CNN

In recent years, researchers have focused their attention on how to deploy CNNs on hardware acceleration platforms. On the one hand, CNN has a large amount of calculation and a large amount of data. How to optimize the network algorithm for hardware deployment has become a research hotspot. Some researchers have optimized the computational data flow of CNNs to take advantage of the parallelism of the algorithm. Li et al. [38] proposed an algorithm framework of CBFF-SSD, based on SSD, using lightweight MobileNet as the feature extraction network of the algorithm model. At the same time, a deep learning processing hardware architecture supporting parallel processing of feature maps was designed, and the deployment of the model was completed on Xilinx XC7Z100. Caba [39] and others developed a highly optimized new HyperLCA algorithm for the limited power consumption budget and low-bandwidth downlink of remote sensing platforms such as drones, and used Xilinx (San Jose, CA, USA) Zynq-7000 series chips to complete the FPGA Heterogeneous acceleration. This design scheme can achieve real-time detection effect in real-life hyperspectral application engineering. Compared with the three GPUs of Jetson Nano, Jetson TX2, and Jetson XavierNX, their FPGA solution is lower in price and higher in performance, and has the lowest power consumption, being just over 3 W. Fan H. [40] proposed the design scheme of deploying SSDLiteM2-MobileNetv2 on Xilinx ZC706, and realized the 8-bit fixed-point quantization of SSDLiteM2-MobileNetv2 through the quantization scheme. The experimental scheme reached a detection speed of 65 frames per second, meeting the requirements of real-time detection need. Zhang et al. [41] completed the deployment of the improved YOLOv2 network model on Xilinx (San Jose, CA, USA) ZYNQ xc7z035, and the power consumption was only 5.96 W. Based on the YOLOX-s network model, Wang et al. optimized its core convolution module. At the same time, they designed a three-way prefetch cache queue to make full use of the on-chip data reuse model to solve the external DDR bandwidth in the multi-level cache process. The cache bottleneck problem is solved and the model inference performance is improved. The design architecture achieves an average throughput of 399.62 GOPS, and the computing efficiency of DSP reaches 97.56%. Compared with other researchers' designs, resource utilization has been greatly improved. Yan T. et al. [42] proposed a scheme to automatically deploy CNN on FPGA. A series of hardware-oriented CNN improvements are proposed to reduce the model complexity. Based on this, a reconfigurable array of processing engines and an efficient convolution computing architecture are designed as hardware accelerators. At the same time, a compilation tool chain is also introduced to realize the automatic conversion of CNN models to hardware instructions, which facilitates the deployment of various network models on hardware. Finally, the deployment of the improved models of VGG16 and YOLOv2 on FPGA was

realized, and the power consumption on Xilinx AC701 (San Jose, CA, USA) is 6.77 W and 6.51 W, respectively. This solves the problem that most FPGA acceleration solutions can only target specific network models, and meet the low power consumption requirements of airborne or spaceborne platforms. Tan [43] proposed an FPGA deployment scheme based on network pruning and subgraph fusion, which reduces the number of model parameters by 11 times and achieves 9–10 times inference acceleration while sacrificing part of the detection accuracy of the model.

On the other hand, numerous researchers are leveraging the Vitis AI development environment offered by Xilinx to accelerate artificial intelligence computations and explore optimal neural network architectures for specific applications. Vitis AI offers a comprehensive development platform for efficiently deploying deep learning models on Xilinx FPGAs and SoCs. Key components of Vitis AI include its compilers and runtime libraries. They facilitate the transformation of models trained using popular deep learning frameworks like TensorFlow, PyTorch, and Caffe into highly efficient hardware accelerators through quantization and compilation techniques. By harnessing the parallel computing capabilities of FPGAs and SoCs, these accelerators enable real-time applications and edge devices to benefit from low-latency, high-throughput inference acceleration. By utilizing Vitis AI, users can perform pruning, quantization, and compilation operations on network models, and deploy them onto the FPGA of the Zynq-ultraScale series, including KV260, ZCU102, ZCU104, and ZCU106. Wang et al. [44] proposed a YOLOv3 convolutional neural network accelerator based on the FPGA+ARM architecture of the AXI bus. The YOLOv3 network model was quantified and pruned through the Vitis AI tool chain, and finally deployed on Xilinx Zynq ZCU104 accelerating convolutional neural networks. The power consumption of this solution is 25 W, and the FPS is 84.5518. Compared with GPU GeForce GTX1080, it has lower power consumption and higher detection real-time performance. Chen [45] and others used the Vitis AI 1.4 framework to complete the deployment of the neural network model on the Xilinx ZCU 104 FPGA, and finally achieved 88.8% mAP and 28 FPS, realizing the real-time detection of the road unevenness detection system.

2.4. Difference from Existing Works

Most prior studies have predominantly focused on using lightweight feature extraction networks or applying pruning techniques independently. Currently, research on the combination of lightweight network architectures with model pruning methods is limited, leaving further scope for reducing model parameters. Furthermore, some outdated CNN models such as SSD, YOLOv2, and YOLOv3 have achieved deployment on the FPGA end, but their detection performance is somewhat inadequate. Conversely, the novel network structure operators of some new models like CenterNet, YOLOv5, and YOLOv7 necessitate substantial modifications to enable deployment on edge devices with restricted operators and instruction sets, significantly impacting development cycles [46]. In summary, we choose YOLOv4 as the reference model and employ MobileNetv3 to lightweight the feature extraction network of YOLOv4. Subsequently, we employ a channel pruning strategy for model pruning and further optimize the model parameters. Finally, to compress the network model further and accomplish deployment on KV260, we utilize Xilinx (San Jose, CA, USA) Vitis AI quantizer based on PTQ (post-training quantization) to quantize the floating-point parameters into fixed-point data before model deployment.

3. Methodology

3.1. GridMask-Mosaic Data Augmentation

Data enhancement is a common method in the field of image processing to improve the robustness of network models. The common methods are to crop, rotate, adjust brightness, flip, and mask the image. Compared with YOLOv3, YOLOv4 uses the Mosaic algorithm in data enhancement, which can better help the model to better learn targets from different angles and locations, thereby improving the generalization ability and robustness of the network model [47]. Mosaic data enhancement is an improved version of CutMix, and

the two have certain similarities in theory. CutMix data augmentation involves splicing two images together, where a random region is selected from one image and cut out, then pasted onto another image to create a new training sample. This helps the model learn to handle boundaries and contextual information between different objects. Mosaic data augmentation, on the other hand, randomly arranges and reorganizes four input images by applying random zooming, random cropping, random flipping, and color gamut changes to generate a new composite image. This provides diverse and varied training samples, aiding the model in learning to adapt to various complex scenes and object combinations. Mosaic data enhances the image processing effect, as shown in Figure 3.

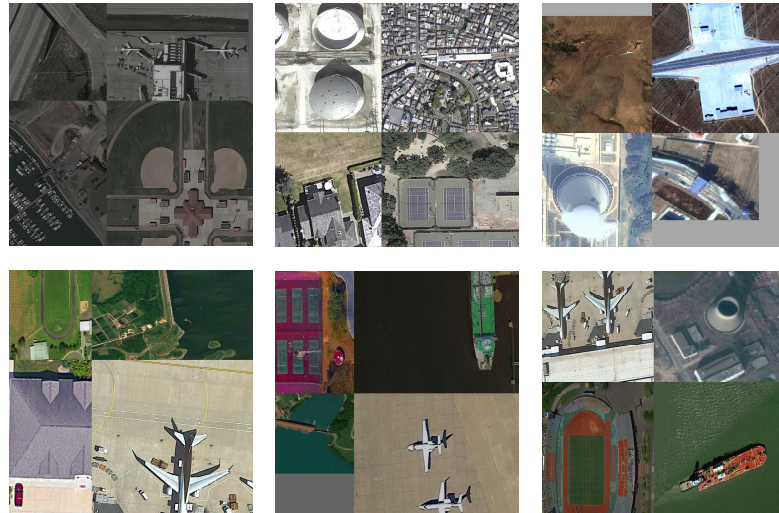


Figure 3. Mosaic data enhancement effect.

In practical applications, due to the complex shooting background environment, it is inevitable that some remote sensing images will be occluded. The existing datasets do not contain a sufficient number of occlusion scenes, and the data augmentation techniques incorporated in the YOLOv4 network model do not support augmentation for occlusion. In order to solve this problem, we use the GridMask algorithm to perform a target random erasure operation on the training set part of the DIOR dataset, so as to achieve the purpose of improving the robustness of the network model.

GridMask belongs to the data enhancement algorithm of the information deletion category, which pays special attention to the degree of grasp of image information deletion. Too much deletion of image information will cause the remaining image information to fail to correctly express the target information, thus becoming noisy data, while too little deletion of image information will cause the target to be unaffected and cannot improve the robustness of the model [48]. The GridMask algorithm is based on the original information deletion algorithm, and uses evenly distributed square areas to delete image information, so that large areas of continuous images will not be deleted. The data enhancement effect of GridMask combined with Mosaic is shown in Figure 4.

For the GridMask data enhancement algorithm, set the output image to $f(x) = x \times M$, in which $x \in R^{H \times C \times W}$ represents the input image and $M \in (0, 1)^{H \times W}$ represents the binary mask. If it is 1, the original image information is retained, otherwise the original image information is deleted, which $f(x)$ represents the output image after the image has been processed by the GM algorithm. Use parameters $(r, d, \delta_x, \delta_y)$ to represent M and M is unique, where r means the ratio of retaining the input image information, d determines the size of the deleted area, and δ_x and δ_y are the distance from the first complete unit to the image boundary.

Where r represents the proportion of information in the input image, and for the parameter r , define the hold ratio t given to M as shown in Equation (1):

$$t = \frac{\text{sum}(M)}{W \times H} \tag{1}$$

where W and H refer to the width and height of the input image, respectively; if t is too small, the image will lose too much information, and the remaining area will become noisy data, which cannot achieve the effect of improving the robustness of the model. The relationship between r and t is shown in Equation (2).

$$t = 2r - r^2 \tag{2}$$

where d determines the size of a single deleted region, and when r remains unchanged, the relationship between the side lengths l and d of a single deleted region is shown in Equation (3).

$$l = d \times r \tag{3}$$

For d , the larger the d , the larger the l , and during training, the proportions are kept unchanged. In addition, randomness is added to expand the diversity of the image, and the value of d is shown in Equation (4).

$$d = \text{random}(d_{\min}, d_{\max}) \tag{4}$$

When d is small, the probability of experimental failure can be reduced, but when it is too small, it will lead to poor experimental results and cannot effectively improve the robustness of the model. The masks of r and d can be determined by moving δ_x and δ_y , and the values of δ_x and δ_y are shown in Equations (5) and (6), respectively.

$$\delta_x = \text{random}(0, d - 1) \tag{5}$$

$$\delta_y = \text{random}(0, d - 1) \tag{6}$$

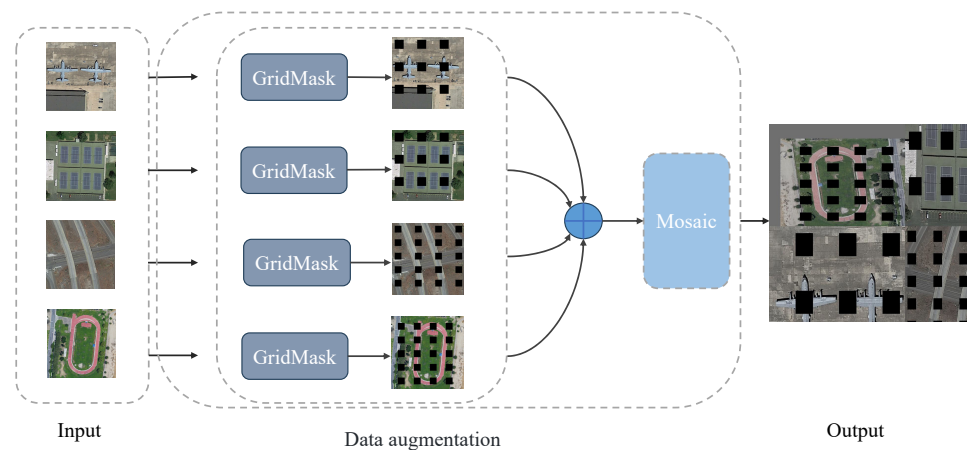


Figure 4. GridMask-Mosaic data enhancement effect.

3.2. YOLOv4-MobileNetv3 Framework

Through the analysis of the network structure of YOLOv4, it can be seen that using CSPDarkNet53 as the backbone extraction network can achieve better feature extraction capabilities, but the network model still has the problem of a large number of parameters and high computational complexity, which will consume a lot of computing resources and is not suitable for deployment to embedded devices and FPGAs. We propose the YOLOv4-MobileNetv3 network framework to solve the problem of large model parameters and high computational complexity. The YOLOv4-MobileNetv3 network framework is shown in

Figure 5. By adopting the lightweight network MobileNetV3 as the backbone extraction network, the depthwise separable convolution is used to improve the network convolution layer to reduce the amount of parameters. MobileNetV3 is a lightweight network that, compared to other lightweight networks like Ghost, achieves higher performance while maintaining its lightweight nature through the introduction of effective design strategies and improved structure. It excels in accuracy and is particularly suitable for tasks that require higher precision. By using lightweight components and operations, MobileNetV3 effectively reduces the number of parameters and computational complexity, resulting in better efficiency in resource-constrained environments such as mobile devices. This means that it can run on lower computational resources and is well suited for embedded devices and mobile applications. In order to make the DPU support the neural network operator of the network model, we also use the LeakyReLU function as the activation function, and the LeakyReLU function is shown in the Equation (7). LeakyReLU is a commonly used activation function in neural networks that enhances the robustness of the network. Unlike the traditional ReLU activation function, LeakyReLU returns a small slope (typically 0.01) on its negative input range instead of simply returning 0. Its main advantage is that it increases the robustness and non-linearity of neural networks while addressing some training issues. The parameter α is a constant smaller than 1, typically set to 0.01. The purpose of this function is to increase the non-linear characteristics of neural networks and prevent the issue of dead neurons in the output layer. Additionally, LeakyReLU can mitigate the problem of vanishing gradients during training, thereby improving the training performance of the neural network. The SPP model pooling sizes of the original YOLOv4 are 9 and 10, but due to compatibility problems with the DPU operator of KV260, maxpool can only be downloaded from 22–88. So, we modified the pooling size of the SPP model of YOLOv4 to 3, 5 and 7. Changing the pool size of the initial SPP model can result in information loss and a decrease in spatial resolution, which may impact detection performance for certain specific tasks. However, reducing the pool size allows us to decrease computational requirements to some extent, thereby improving the inference speed and efficiency of the model.

$$\text{LeakyReLU} = \max(0, x) + \alpha \times \min(0, x) \quad (7)$$

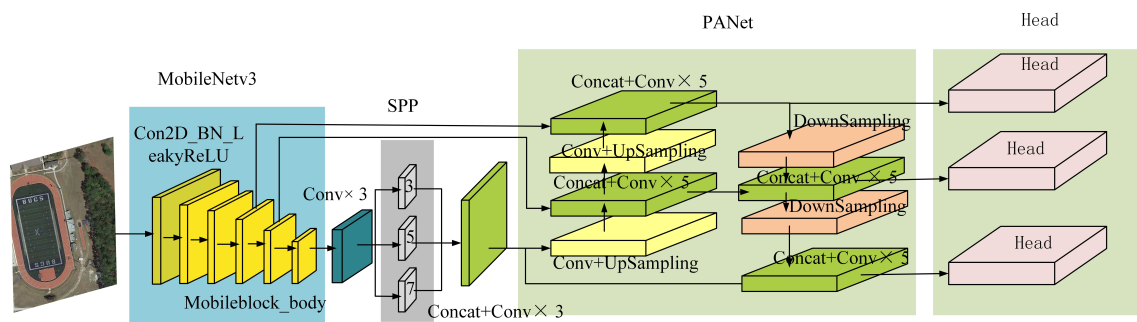


Figure 5. YOLOv4-MobileNetV3 framework.

MobileNetV3 continues the linear bottlenecks and inverted residuals of MobileNetV2, introduces 3×3 depth separable convolution, uses the hswish activation function instead of the swish function to reduce the amount of calculation, and adds the SE attention mechanism module to improve the detection accuracy of the network model.

After modifying the feature extraction network and activation function, you only need to train according to the normal method of the YOLOv4 model. Experiments show that the parameters of the improved YOLOv4-MobileNetV3 model after lightweight processing are 11.47 M, which is 82.09% less than the original YOLOv4 network model.

3.3. YOLOv4-MobileNetv3 Model Channel Pruning

The inference stage of the network model requires a large amount of memory bandwidth, and the YOLOv4-MobileNetv3 model still has redundant weights and activation values after lightweight processing, which leads to inefficient calculation in the model inference stage. In addition, due to the limitation of instruction sets and basic operations on hardware devices, many innovative modules or network layers will fail compilation, which greatly limits the deployment of lightweight networks on the embedded side and FPGA [49]. In view of the above situation, channel pruning is used to further compress the YOLOv4-MobileNetv3 network model.

Compared to pruning methods that remove individual neurons, channel pruning does not introduce sparsity into the initial CNN model architecture, thus requiring no special hardware or software to implement [50]. This makes channel pruning extremely versatile in network model pruning, applicable to almost all model inference platforms, and suitable for hardware acceleration [51]. In channel pruning, the training acceleration of the YOLOv4-MobileNetv3 network model is achieved by using batch normalization (BN) between adjacent convolutional layers. The normalization operation is shown in Equation (8).

$$y = \frac{\gamma(x - \bar{x})}{\sqrt{\theta^2 + \varepsilon}} + \beta \tag{8}$$

where \bar{x} represents the mean of the input feature x , θ^2 represents the variance, β represents the bias, and γ represents the trainable scale factor. During training, γ is used to distinguish between important and non-important channels of the YOLOv4-MobileNetv3 network model. After selecting the pruning rate, non-critical channels are pruned, as shown in Figure 6. Channel sparsity training is performed by using L1 regularization on γ ; sparsity training loss $f(\gamma)$ is shown in Equations (9) and (10).

$$f(\gamma) = L(\gamma) + \mu \|\gamma\|_1 \tag{9}$$

$$\|\gamma\|_1 = |\gamma_1| + |\gamma_2| + |\gamma_3| + |\gamma_4| + \dots + |\gamma_n| \tag{10}$$

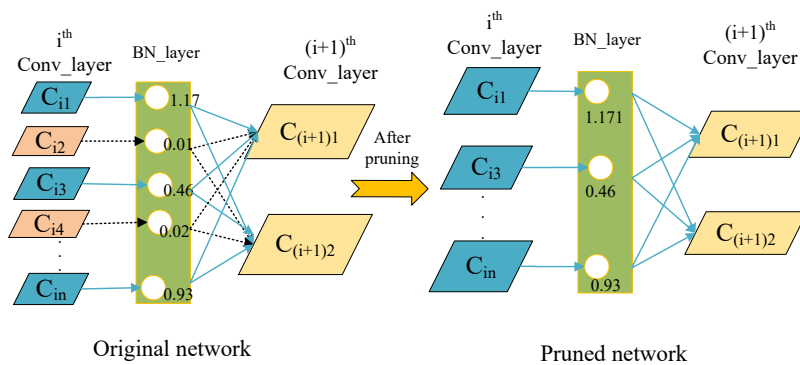


Figure 6. Channel pruning.

The $L(\gamma)$ in the first part of the equation represents the loss of YOLOv4-MobileNetv3; μ represents the penalty factor; $\|\gamma\|_1$ represents the L1 norm of γ . Equation (11) is obtained by expanding Equation (9) at γ^* , where H is represented as a Hessian matrix, and since the trainable scale factors γ in the parameter are independent of each other, H can become a diagonal matrix, as shown in Equation (12).

$$f(\gamma) = L(\gamma^*) + 0.5(\gamma - \gamma^*) \times (\gamma - \gamma^*)^T \times H \tag{11}$$

$$H = \begin{bmatrix} H_{(1,1)} & & \\ & \ddots & \\ & & H_{(n,n)} \end{bmatrix} \tag{12}$$

Then, the original formula $f(\gamma)$ can be expanded into Equation (13).

$$f(\gamma) = L(\gamma^*) + \sum_{i=0} [0.5H_{(i,i)}(\gamma_i - \gamma_i^*)^2 + \mu|\gamma_i|] \tag{13}$$

Combined with the γ assumption of mutual independence, we can obtain Equation (14).

$$f(\gamma_i) = L(\gamma_i) + 0.5H_{(i,i)}(\gamma_i - \gamma_i^*)^2 + \mu|\gamma_i| \tag{14}$$

Equation (15) can be obtained by deriving the above Equation. For the Sgn function, if the input is ω_i^* than 0, the Sgn function will return 1; if it is equal to 0, the Sgn function will return 0; if it is less than 0, the Sgn function will return -1 .

$$H_{(i,i)}(\omega_i - \omega_i^*) + \mu \times \text{Sgn}(\omega_i^*) = 0 \tag{15}$$

In summary, it can be obtained γ_i , as shown in Equation (16).

$$\gamma_i = \begin{cases} \text{Sgn}(\gamma_i^*) (|\gamma_i^* - \frac{\mu}{H_{(i,i)}}), |\gamma_i^*| > \frac{\mu}{H_{(i,i)}} \\ 0, |\gamma_i^*| \leq \frac{\mu}{H_{(i,i)}} \end{cases} \tag{16}$$

The scale factor of each BN layer of the YOLOv4-MobileNetv3 network before and after pruning is shown in Figures 7 and 8. It can be seen that after the sparse training is completed, the γ coefficient is concentrated around 0, which means that the sparse training is saturated. The output of a channel close to 0 approximates a constant and can be clipped.

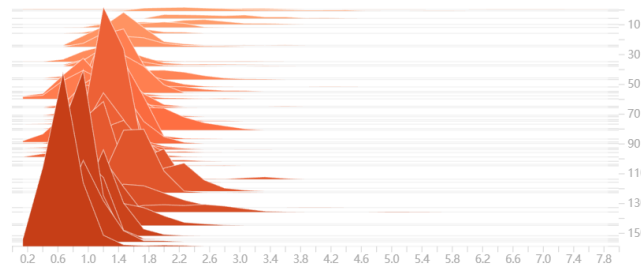


Figure 7. The distribution of γ coefficients in each BN layer before sparse training.

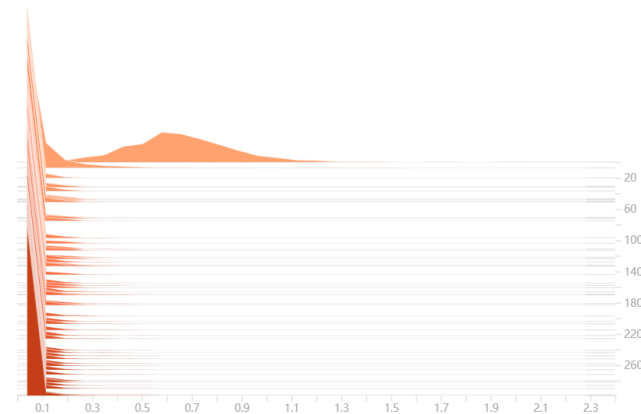


Figure 8. The distribution of γ coefficients in each BN layer after sparse training.

After sparse training at 300 epochs, channel pruning was performed on the sparse YOLOv4-MobileNetv3 network model using a pruning rate of 0.85. The experimental

results show that the parameter size of the model after pruning is 9.42 M, and the parameter size is reduced by 2.05 M.

3.4. Vitis AI Deploys CNN

This paper adopts Xilinx Zynq (AMD Xilinx, San Jose, CA, USA) a software and hardware collaborative acceleration platform combining FPGA and ARM UltraScale + MPSoC realizes the edge deployment of YOLOv4-MobileNetv3. UltraScale + MPSoC is composed of two core devices: programmable logic (PL) and a processing system (PS). It overcomes the lack of real-time computing capability of the ARM processor, and improves the limited algorithm realization capability of the FPGA. UltraScale + MPSoC adopts TSMC's 16 nm FinFET process node, which not only has high performance, but also greatly reduces power consumption.

3.4.1. YOLOv4-MobileNetv3 Model Quantization

It is important to note that quantification methods also need to be used in conjunction with specific hardware platforms. For example, NVIDIA uses TensorRT to quantify network models and deploy them on Nano, TX1, and TX2 devices. Since our FPGA evaluation boards are part of the Zynq UltraScale+ MPSoC series, we can choose Vitis AI's quantizer to quantize the model. The Vitis AI quantizer can reduce the computational complexity of the network model with little loss of inference accuracy. The deep learning frameworks currently supported by the Vitis AI quantizer are PyTorch and TensorFlow, and the quantizer names are `vai_q_pytorch` and `vai_q_tensorflow`, respectively. We use Int8 fixed-point quantization to optimize the algorithm model, and convert the 32-bit floating-point number into an 8-bit fixed-point number format. The quantization algorithm is as Algorithm 1. In the Vitis AI quantizer, the Pytorch quantizer supports two different methods: post-training quantization (PTQ) and quantize aware training (QAT), to quantize deep learning models. This paper selects the PTQ scheme to quantize the YOLOv4-MobileNetv3 model. PTQ is a technology that converts the pretrained floating-point model into a quantized model, and its characteristic model accuracy loss is extremely low. Model quantification generally needs to determine the number of quantized pictures according to the scene. The more detection categories and the more complex the scene, the more pictures required for quantization. Combined with the characteristics of many detection scenes in the DIOR dataset, this experiment uses 4000 images as a representative calibration dataset for calibration to run several batches of inference on the floating-point model to obtain the distribution of activations.

Algorithm 1: Model Quantization

```

Input: input 32-bit floating-point weight
Output: Deployment file
1 Floating point model;
2 pretreatment;
3 Quantify the weight and calibrate the activation;
4 while Calibration data set (No label) do
5   | if and else;
6   | if Qualified don't accuracy then
7   |   | Quantitative fine-tuning;
8   |   | End of quantization.
9   | else
10  |   | End of quantization.
11  | end
12 end
13 Generate the DPU model ;
14 Compile;
15 Deployment.

```

3.4.2. YOLOv4-MobileNetv3 Model Compilation and Deployment

The quantized network model needs to be compiled and mapped into a highly optimized DPU instruction stream before the DPU core can be scheduled on the FPGA to achieve hardware acceleration. The compiler in the Vitis AI suite can perform the compilation operation of the quantized network, which includes an optimizer, parser, and code generator. After the Vitis AI Library compiler (VAL_C) analyzes the topology of the optimized and quantized input model, VAL_C will build an internal calculation graph as an intermediate representation (IR), and build a control flow and data flow based on this. Then, it compiles and optimizes operations based on the calculation graph. Finally, the code generator will map the optimized calculation graph to the DPU instruction stream.

3.4.3. DPU Core Configuration

A deep learning processor (DPU) is a programmable accelerator optimized for neural network models. It consists of a set of parameterizable IP cores that can be implemented through hardware programming without layout and wiring. DPU supports most operations of deep learning, such as convolution, pooling, full connection, activation, etc. Vitis AI provides a series of DPUs for Xilinx's Kria KV260, Versal card, Alveo card, Zynq UltraScale + MPSoC and other embedded devices. The parameters of the DPU can be configured according to the application, so as to improve throughput, latency, and scalability, and unique differences and flexibility are achieved in terms of power consumption. The DPU architecture parameters are shown in Table 1. Combining the logic resources of KV260 and the parameters of the network model, we choose B4096 as the core of the DPU architecture. The DPU 4096 architecture provides high computational power and processing speed, making it suitable for handling complex tasks and large-scale data. The KV260 faces high demands for computational resources, requiring the processing of massive amounts of data and complex calculations. The high-performance computing capability of the DPU 4096 makes it an ideal choice to meet these requirements. Additionally, the DPU 4096 architecture has been widely applied and validated, ensuring its stability and reliability. Choosing a proven architecture like this helps mitigate risks during development and deployment and ensures system stability and consistency.

Table 1. DPU parameter architectures.

DPU Architectur	DSP	LUT	Peak Operand	Channel Parallelism	Pixel
B512	118	27,893	512	8	4
B800	166	30,468	800	10	4
B1024	230	34,471	1024	8	8
B1152	222	33,238	1152	12	4
B1600	326	38,716	1600	10	8
B2304	438	42,842	2304	12	8
B3136	566	47,667	3136	14	8
B4096	710	53,540	4096	16	8

4. Experiments and Evaluation

This section will describe the experimental details. By comparing and analyzing the performance of different network models, we can evaluate the superiority of our design scheme. At the same time, through the comparison of power consumption of different hardware platforms, it is shown that our design scheme has significant advantages compared with GPU and CPU in the application scenario of spaceborne remote sensing images.

4.1. Dataset Description

We use the optical remote sensing image dataset DIOR created by Li et al. of Northwestern Polytechnical University to evaluate the performance of the algorithm model [52]. The

DIOR dataset is a large-scale optical remote sensing image dataset, including 23,463 images with a size of 800×800 and a resolution ranging from 0.5 m to 30 m. There are 192,472 instances in total, covering 20 object categories. The images in the DIOR dataset cover a span of several years, capturing a wide range of imaging conditions, weather variations, and seasons. This diversity ensures that the dataset provides a representative collection of remote sensing data. In our experiments, 80% of the images in the DIOR dataset are selected as the training set, along with 10% for the validation set and 10% for the test set. Figure 9 shows some examples of images from the DIOR dataset. Sample images of golffield, vehicle, trainstation, chimney, groundtrackfield, airplane, stadium, tennis court, storage tank, ship, windmill and airport are shown in (a) to (l), respectively.



Figure 9. DIOR dataset image example: (a) golf field; (b) vehicle; (c) train station; (d) chimney; (e) ground track field; (f) airplane; (g) stadium; (h) tennis court; (i) storage tank; (j) ship; (k) windmill; (l) airport.

4.2. Experiment Environment

The deep learning framework of this paper adopts Pytorch1.8, based on the operating system of Ubuntu18.04; the Python version is 3.8, the integrated development environment is PyCharm professional version, and the Vitis AI version is Vitis AI 2.5. Two NVIDIA A100s are used to train the network model and pruning. The hardware platform CPU is AMD R7-4800H, the GPU is NVIDIA RTX 2060 (6G), and the FPGA is Xilinx KV260. During the training and pruning process, the batch size is set to 32, the input image size is 416×416 , the initial learning rate is 0.001, the optimizer is SGD, and the cosine annealing strategy is used to dynamically adjust the learning rate.

4.3. Evaluation Indicators

In this paper, mean of average precision (mAP) is used to evaluate the target detection effect; the parameter quantity and FPS are used to evaluate the detection speed; the class deployment platform is extremely important, so the power consumption index is used to

evaluate the processing performance of the hardware deployment platform. The relevant formulas are as follows:

$$FPS = \frac{x}{T} \quad (17)$$

$$Precision = \frac{TP}{FP + TP} \quad (18)$$

$$recall = \frac{TP}{FN + TP} \quad (19)$$

$$AP = \int_0^1 P(x)dx \quad (20)$$

$$mAP = \sum_{k=1}^N \frac{AP_k}{N} \quad (21)$$

In Equation (17), x represents the number of detected pictures, T is the time consumed to detect x pictures, FPS represents the number of pictures detected per unit time: the higher the FPS, the faster the detection speed, the higher the real time. In Equations (18) and (19), FP is the detected false sample, which is the sample detected in the target detection task where the target class is inconsistent with the real class; TP is a true sample, indicating that the detected target class is the same as the real class, and FN is a false negative sample, indicating a sample that actually exists but has not been detected. Equation (21) mAP refers to the average of the average accuracy (AP), and all classes detected by the model can be drawn from the accuracy recall to form a curve, and the area enclosed by this curve and the coordinate axis is AP, which is represented by Equation (20). N represents the detection categories.

Xilinx provides the Maxim PowerTool GUI tool and Uart software (AMD Xilinx, San Jose, CA, USA, version 2.32.03) for developers to perform power testing of FPGAs. According to the actual situation, this paper uses the Maxim PowerTool USB cable with chip and the corresponding GUI software to test the real-time power of FPGA. Although the Maxim PowerTool can display real-time current and voltage levels, it cannot save historical data. We programmed a program in Python language to read out memory values in real time and detect average power by reading power values in continuous time. The calculation formula for the power metric is a fundamental equation used to quantify the rate of energy transfer or consumption in a system. It is expressed as Equation (22), where T is the total time of consumption, and $p(t)$ represents the function of power as a function of time.

$$\bar{P} = \frac{\int_0^T p(t)dt}{T} \quad (22)$$

4.4. Ablation Experiment

In order to verify the effect of algorithm model optimization, the feature extraction network, convolution module, channel pruning, and quantization models in the experiment were ablated experiments, and the experimental results are shown in Figure 10. In Figure 10, A represents the original YOLOv4 network model with 64.04 M parameters and an mAP of 84.47%. B represents the model using the MobileNetv3 feature extraction network without deep separable convolution modules, with a parameter size of 39.99 M and an mAP of 83.69%. C represents the model using the MobileNetv3 feature extraction network with deep separable convolution modules, with a parameter size of 11.47 M and an mAP of 81.54%. D represents the model after data enhancement using GridMask and Mosaic algorithms, with 11.47 M parameters and an mAP of 82.24%. E represents the model after applying channel-level pruning, with 9.42 M parameters and an mAP of 83.03%. F

represents the model after Vitis AI quantization, with 5.69 M parameters and an mAP of 82.61%.

From the data in the figure, it is evident that after replacing the feature extraction network and incorporating deep separable convolution operations, the model accuracy decreased by 2.93%, while the number of model parameters was significantly reduced by 52.57 M. This is because lightweight feature extraction networks and deep separable convolution networks, compared to the original network structure, sacrifice some feature representation capabilities, resulting in lower accuracy. However, this reduction in accuracy comes with significant reductions in the number of model parameters, which leads to improved efficiency when dealing with limited computing resources. Following the data augmentation operation, there was a 0.7% improvement in the accuracy of the algorithm, indicating that data augmentation has a positive effect on enhancing model accuracy. Channel pruning reduces the number of parameters in the model by identifying and removing redundant channels in the network. After pruning, the unused channels no longer contribute to the forward and backpropagation processes, resulting in reduced computational and storage requirements. By applying channel pruning and fine tuning, an additional reduction of 2.05 M parameters is achieved, making the model more lightweight. Moreover, through the fine-tuning and retraining process, the model is further optimized and adapted, leading to improved accuracy and an additional 0.79% performance boost. Channel trimming and fine-tuning effectively compress the model size, reduce parameters, and enhance accuracy through retraining and optimization. This strategy is highly effective for optimizing models, striking a balance between reducing complexity and resource usage while maintaining high performance. Finally, after model quantization, the parameter size of the model decreased by 39.6%, while the accuracy decreased by 0.42%. In conclusion, the model improvement scheme presented in this paper reduces the size of the model parameters to 5.69 M at the expense of 1.86% mAP, which is a reduction of 91.11% compared to the original YOLOv4 model. The impact of each improvement scheme is evident, as they synergistically complement each other and ultimately deliver notable results in terms of model parameter count and detection accuracy metrics.

4.5. Comparative Experiments

Table 2 compares the data of the lightweight model proposed by us after pruning with the current mainstream target detection models Faster-RCNN, SSD, Centernet, YOLOv3, and YOLOv4. From the data in the Table 2, we can see that the mAP of the YOLOv4-MobileNetv3 model is 83.03%, 14.28% higher than Faster-RCNN, 4.85% higher than SSD, 4.05% higher than Centernet, 2.93% higher than YOLOv3, and 1.44% lower than YOLOv4. The parameter size is 9.42 M, which is 93.13% less than Faster-RCNN, 63.98% less than SSD, 71.17% less than Centernet, 84.72% less than YOLOv3, and 85.29% less than YOLOv4. Compared to Faster-RCNN, SSD, Centernet, and YOLOv3, our improved model achieves the best mAP with minimal parameters. The accuracy of the pruned and fine-tuned YOLOv4-MobileNetv3 model remains lower compared to the original YOLOv4 network. This disparity can be attributed to the inadequate feature expression capability resulting from the utilization of a lightweight feature extraction network. Consequently, the model struggles to capture abundant information and target features in the input image, leading to a decrease in the detection accuracy. In contrast, we made a compromise by sacrificing 1.44% of the mAP in order to reduce the model parameters to only 0.15 times the size of the original YOLOv4 network. While this reduction in accuracy is within acceptable limits, our main focus lies in achieving optimal performance on the FPGA platform. This strategic decision holds significant importance for our specific application requirements in terms of computational efficiency and resource utilization.

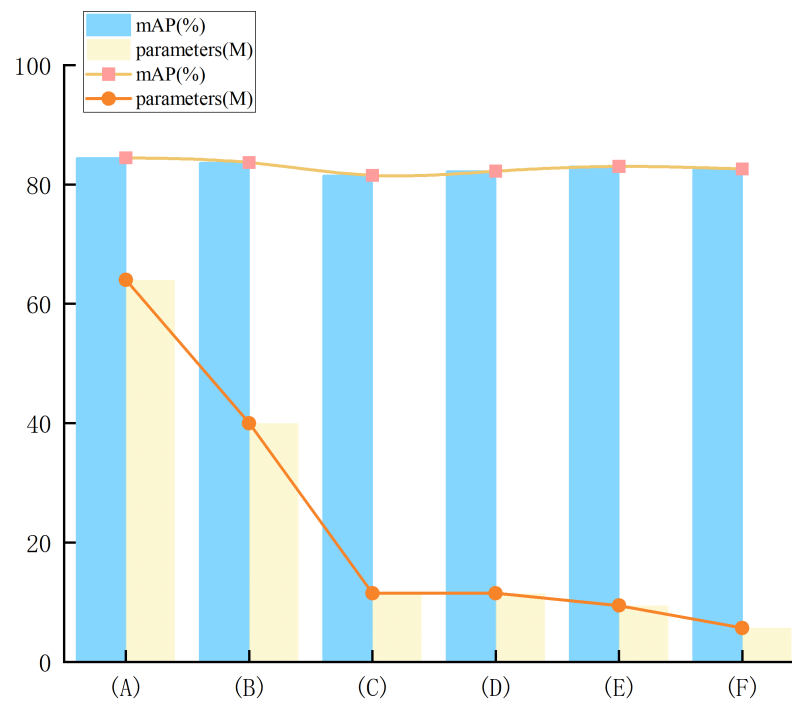


Figure 10. Ablation experiment: parameters and mAP changes. (A) YOLOv4 original network model; (B) model using MobileNetv3 feature extraction network without depth-wise separable convolution modules; (C) model using MobileNetv3 feature extraction network with depth-wise separable convolution modules; (D) model after data augmentation; (E) model after channel pruning; (F) model after quantization.

Table 2. Comparison of mainstream model data.

Network Model	Faster-RCNN	SSD	Centernet	YOLOv4	YOLOv3	Ours
Backbone	VGG16	SSD	Resnet50	CSPDarkNet53	Darknet53	MobileNetv3
airplane	0.78	0.91	0.91	0.95	0.87	0.94
airport	0.87	0.88	0.92	0.93	0.86	0.95
baseballfield	0.89	0.94	0.95	0.97	0.94	0.96
basketballcourt	0.83	0.85	0.83	0.86	0.87	0.83
bridge	0.40	0.45	0.48	0.57	0.50	0.50
chimney	0.78	0.84	0.83	0.83	0.78	0.84
dam	0.91	0.80	0.84	0.87	0.84	0.82
Expressway-toll-station	0.59	0.68	0.71	0.87	0.80	0.77
Expressway-Service-area	0.89	0.92	0.90	0.94	0.88	0.95
golffield	0.83	0.88	0.81	0.82	0.85	0.85
groundtrackfield	0.82	0.87	0.86	0.89	0.79	0.88
harbor	0.63	0.67	0.62	0.71	0.65	0.71
ship	0.27	0.76	0.84	0.93	0.93	0.91
stadium	0.95	0.96	0.88	0.96	0.71	0.94
storagetank	0.45	0.65	0.75	0.86	0.82	0.81
tennis court	0.87	0.94	0.93	0.95	0.93	0.95
trainstation	0.58	0.63	0.70	0.71	0.75	0.71
vehicle	0.21	0.42	0.50	0.63	0.65	0.54
windmill	0.65	0.88	0.85	0.91	0.94	0.90
overpass	0.67	0.69	0.66	0.73	0.72	0.68
mAP (%)	68.75	78.18	78.98	84.47	80.10	83.03
Parameters	137.08 M	26.15 M	32.67 M	64.04 M	61.63 M	9.42 M

The comparison of the partial image detection effect of the pruned YOLOv4-MobileNetv3 algorithm framework proposed in this paper and the mainstream target detection algorithm on the dataset DIOR is shown in Figure 11.

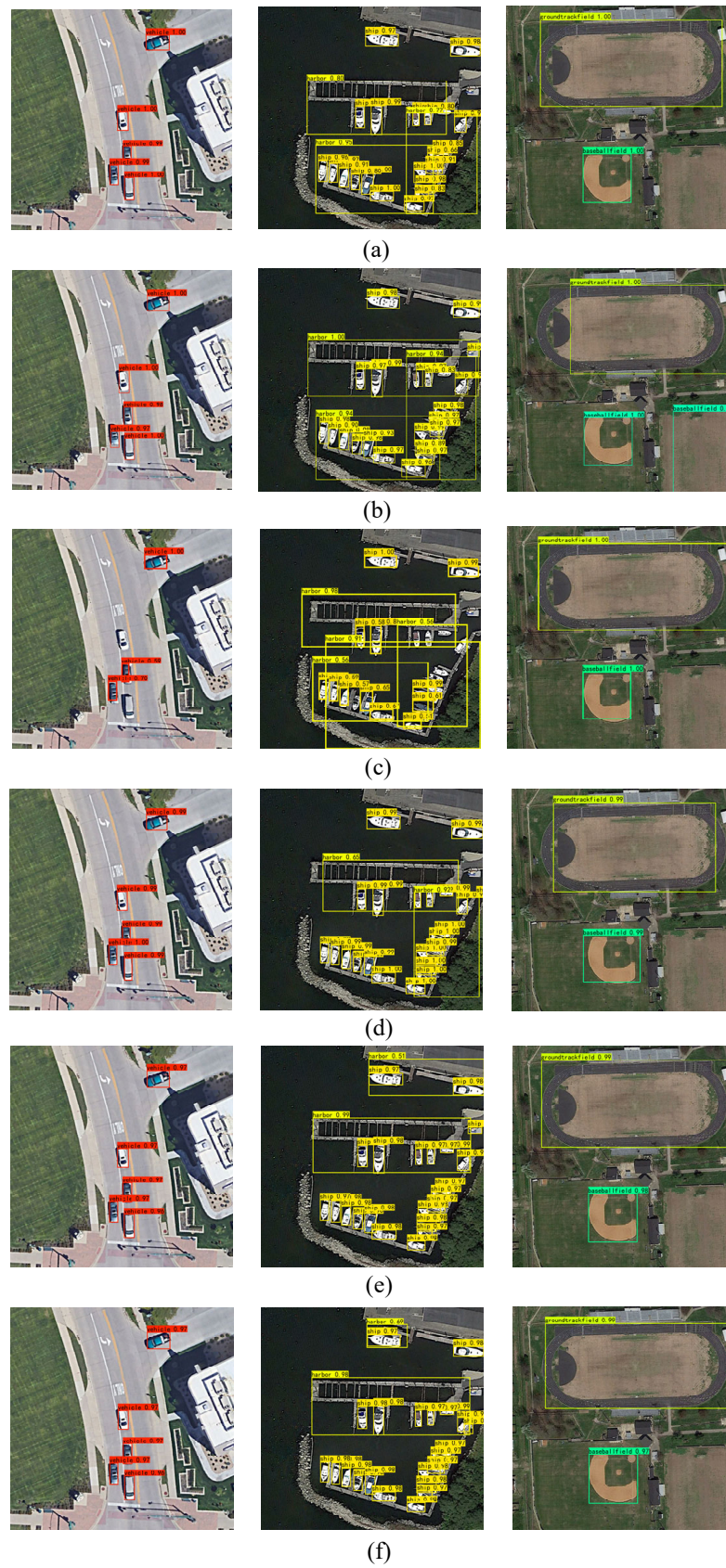


Figure 11. Comparison of mainstream model detection effect. (a) Faster-RCNN; (b) SSD; (c) Centernet; (d) YOLOv3; (e) YOLOv4; (f) ours.

4.6. Hardware Acceleration Platform Comparison

In order to verify the superiority of the proposed deployment scheme, the mainstream deep learning general-purpose processor CPU and GPU were selected for experimental comparison of inference performance. The CPU selected is the AMD R7-4800H and the GPU is NVIDIA RTX 2060 (6G), and the table shows the performance test data comparison of FPGA, GPU, and CPU. As can be seen from the data in Table 3, the FPGA deployment scheme proposed by us reduces power consumption by 81.91% and increases FPS by 317.88% compared with CPU AMD R7-4800H. Compared to the GPU NVIDIA RTX 2060, power consumption is reduced by 91.41% and FPS is increased by 8.50%. Our design solution has a detection speed much higher than the normal human eye persistence time of 24 frames per second, which meets the real-time requirements of object detection, and the power consumption is only 7.2 W, which can meet the low power consumption requirements of airborne and spaceborne aircraft.

Table 3. Hardware platform comparison.

Hardware Platform	Device	Frequency (MHz)	mAP (%)	Power (W)	FPS
CPU	AMD R7-4800H	2900	83.03	39.8	11.52
GPU	NVIDIA RTX 2060	1365	83.03	88.39	44.37
FPGA	XilinxKV260	200	82.61	7.2	48.14

The improved model deployed to Xilinx KV 260 after quantization has an mAP of 82.61%, which is 0.42% lower than that of the model before quantization, which is due to the slight accuracy degradation of floating-point numbers due to quantization. During the quantization process of Vitis AI, precision loss occurs due to the conversion of original floating-point parameters into fixed-point representation. In floating-point representation, parameters can have higher precision and can represent numbers with multiple decimal places, while fixed-point representation can only represent numbers with a fixed number of decimal places or integers. In the quantization process, floating-point parameters are rounded or truncated into fixed-point parameters to accommodate limited computational resources of hardware platforms. This can cause information loss and rounding errors, as the fine details and decimal places of the original model are constrained to fewer bits. Precision loss in the quantization process can potentially impact the performance and accuracy of the model. As the parameters become coarser, the model may fail to capture subtle features and complex relationships present in the original model. As shown in Figure 12, it is evident that despite the accuracy loss resulting from the quantization process, overall, there is minimal impact on the target detection effect. This implies that the quantization process does not hinder the practical application of the model in real-world projects.

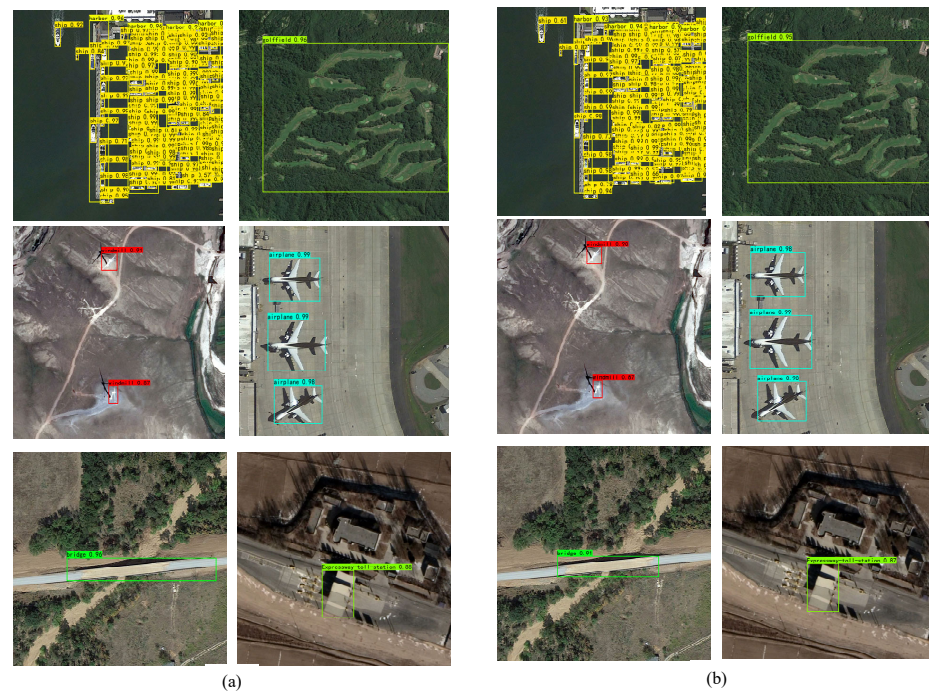


Figure 12. Before and after quantization, different effects were compared. (a) Before quantization; (b) after quantization.

4.7. Comparison of Different Studies

Table 4 provides a comparison between our proposed approach and previous methods. It is noteworthy that our solution has achieved the best results in terms of FPS and power consumption, which can be attributed to the integration of a series of lightweight techniques. By employing a lightweight feature extraction network that combines depth-wise separable convolutions and utilizing two crucial model compression methods, pruning and quantization, we have ultimately enabled our model to exhibit exceptional detection speed performance. Additionally, when comparing Kv260 with ZCU104, Kv260 demonstrates a power advantage.

Table 4. Comparison of different studies.

Method	FPGA Evaluation Board	CNN	FPS	Power (W)
[53]	ZCU104	YOLOv4	27.97	15.82
[54]	ZCU104	YOLOv4	17	20
Ours	KV260	YOLOv4	48.14	7.2

5. Conclusions

This paper proposes an FPGA-based image processing solution for remote sensing images on the aerospace platform, which is based on a lightweight network and network pruning technique applied to YOLOv4-MobileNetv3. The main improvements of this solution include the use of the lightweight feature extraction network MobileNetv3, aiming to reduce the model parameters and improve the detection speed. To further reduce the model parameters and improve computational efficiency, deep separable convolutions are utilized. Additionally, channel pruning techniques are combined to prune redundant parameters, further reducing the model parameters and redundant calculations, thereby improving computational efficiency. In response to the decrease in accuracy after pruning, we fine-tune the model to enhance its accuracy, even surpassing the accuracy before pruning. To adapt to the complex backgrounds of remote sensing images, we apply data augmentation

techniques such as Mosaic and GridMask on the DIOR dataset to enhance the model's robustness and adaptability to various complex backgrounds. For the UltraScale + MPSoC platform used, we utilize the matching quantization tool Vitis AI Quantizer to perform post-training quantization (PTQ) on the pruned YOLOv4-Mobilenetv3 network model, and compile it using Vitis AI compiler, finally deploying it on the Xilinx KV260.

The comparative experimental results on the DIOR dataset demonstrate that our improved network model outperforms mainstream object detection models in terms of mAP and parameter quantity, with mAP slightly lower than the original YOLOv4 model. To validate the superiority of our FPGA acceleration solution, we select the mainstream deep learning hardware deployment platforms, CPU and GPU, for performance comparison. Experimental results show that our FPGA deployment solution has significant advantages in terms of power consumption and FPS. Compared with CPU AMD R7 4800H, FPS improves by 317.88% and power consumption decreases by 81.91%; compared with GPU NVIDIA RTX 2060 (6G), power consumption decreases by 91.85%, and FPS increases by 8.50%. In the scenarios of unmanned aerial and aerospace remote sensing image processing, limited by power consumption, weight, and volume, our design solution meets the requirements of low power consumption and high real-time performance.

The high-performance, low-power image processing hardware acceleration solution proposed in this paper provides some ideas for deploying CNN models on satellite platforms and unmanned aerial platforms. There are still areas for improvement in our experimental approach, as this paper mainly focuses on the post-processing of remote sensing images and only provides a data augmentation scheme for pre-processing. However, in practical engineering, there are usually issues such as noise, distortion, or uneven lighting in the original images captured by satellites. Therefore, pre-processing operations are required before object detection and recognition. Pre-processing steps may include denoising, geometric correction, radiometric calibration, etc., to improve image quality and accuracy. Furthermore, satellite images often have large scales and high resolutions, requiring operations such as cropping, scaling, or segmentation to adapt to specific application scenarios or analysis needs. In future research, we plan to focus on pre-processing and optimize the mAP metric for post-processing object detection tasks, as well as further improve the detection accuracy of the algorithm model through knowledge distillation.

Author Contributions: Conceptualization, Y.Z. and Y.L.; methodology, Y.L. and Y.Z.; software, Y.L.; validation, Y.L.; investigation, C.L. and Y.Z.; writing—original draft preparation, Y.Z. and Y.L.; writing—review and editing, C.L., Y.L. and Y.Z.; visualization, Y.L.; supervision, Y.Z. and C.L.; funding acquisition, C.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Natural Science Foundation of Heilongjiang Province (LH2023F003) for financial support.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data required to reproduce these findings cannot be shared at this time as the data also form part of an ongoing study.

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

References

1. Chen, J.; Xia, M.; Wang, D.; Lin, H. Double Branch Parallel Network for Segmentation of Buildings and Waters in Remote Sensing Images. *Remote Sens.* **2023**, *15*, 1536. [[CrossRef](#)]
2. Cook, K.L.; Andermann, C.; Gimbert, F.; Adhikari, B.R.; Hovius, N. Glacial lake outburst floods as drivers of fluvial erosion in the Himalaya. *Science* **2018**, *362*, 53–57. [[CrossRef](#)] [[PubMed](#)]
3. Voudouri, K.A.; Ntona, M.M.; Kazakis, N. Snowfall Variation in Eastern Mediterranean Catchments. *Remote Sens.* **2023**, *15*, 1596. [[CrossRef](#)]
4. 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](#)]

5. Hong, T.; Liang, H.; Yang, Q.; Fang, L.; Kadoch, M.; Cheriet, M. A real-time tracking algorithm for multi-target UAV based on deep learning. *Remote Sens.* **2022**, *15*, 2. [[CrossRef](#)]
6. Wei, W.; Polap, D.; Li, X.; Woźniak, M.; Liu, J. Study on remote sensing image vegetation classification method based on decision tree classifier. In Proceedings of the 2018 IEEE Symposium Series on Computational Intelligence (SSCI), Bangalore, India, 18–21 November 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 2292–2297.
7. Cheng, G.; Han, J. A survey on object detection in optical remote sensing images. *ISPRS J. Photogramm. Remote Sens.* **2016**, *117*, 11–28.
8. 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 Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; Proceedings, Part I 14; Springer: Berlin/Heidelberg, Germany, 2016; pp. 21–37.
9. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
10. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*; NeurIPS: San Diego, CA, USA, 2015.
11. He, K.; Gkioxari, G.; Dollár, P.; Girshick, R. Mask r-cnn. In Proceedings of the IEEE International Conference On Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2961–2969.
12. Duan, K.; Bai, S.; Xie, L.; Qi, H.C.; Huang, Q.; Tian, Q. Keypoint Triplets for Object Detection. In Proceedings of the IEEE International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–2 November 2019; pp. 27–32.
13. Tian, Z.; Shen, C.; Chen, H.; He, T. Fcos: Fully convolutional one-stage object detection. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–2 November 2019; pp. 9627–9636.
14. Arnold, S.S.; Nuzzaci, R.; Gordon-Ross, A. Energy budgeting for CubeSats with an integrated FPGA. In Proceedings of the 2012 IEEE Aerospace Conference, Big Sky, MT, USA, 3–10 March 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 1–14.
15. Yao, Y.; Jiang, Z.; Zhang, H.; Zhou, Y. On-board ship detection in micro-nano satellite based on deep learning and COTS component. *Remote Sens.* **2019**, *11*, 762. [[CrossRef](#)]
16. Amara, A.; Amiel, F.; Ea, T. FPGA vs. ASIC for low power applications. *Microelectron. J.* **2006**, *37*, 669–677. [[CrossRef](#)]
17. Chen, J.; Zhang, J.; Jin, Y.; Yu, H.; Liang, B.; Yang, D.G. Real-time processing of spaceborne SAR data with nonlinear trajectory based on variable PRF. *IEEE Trans. Geosci. Remote Sens.* **2021**, *60*, 1–12. [[CrossRef](#)]
18. Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; Li, H. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems 29 (NIPS 2016)*; NeurIPS: San Diego, CA, USA, 2016.
19. Han, K.; Wang, Y.; Tian, Q.; Guo, J.; Xu, C.; Xu, C. Ghostnet: More features from cheap operations. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 1580–1589.
20. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 6848–6856.
21. Zoph, B.; Vasudevan, V.; Shlens, J.; Le, Q.V. Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 8697–8710.
22. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. *arXiv* **2016**, arXiv:1602.07360.
23. Gholami, A.; Kwon, K.; Wu, B.; Tai, Z.; Yue, X.; Jin, P.; Zhao, S.; Keutzer, K. Squeezenext: Hardware-aware neural network design. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, Salt Lake City, UT, USA, 18–22 June 2018; pp. 1638–1647.
24. Ma, N.; Zhang, X.; Zheng, H.T.; Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 116–131.
25. Ma, N.; Zhang, X.; Huang, J.; Sun, J. Weightnet: Revisiting the design space of weight networks. In Proceedings of the European Conference on Computer Vision, Glasgow, UK, 23–28 August 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 776–792.
26. 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.
27. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520.
28. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
29. Frankle, J.; Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv* **2018**, arXiv:1803.03635.
30. Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; Graf, H.P. Pruning filters for efficient convnets. *arXiv* **2016**, arXiv:1608.08710.
31. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv* **2015**, arXiv:1510.00149.
32. Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; Zhang, C. Learning efficient convolutional networks through network slimming. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2736–2744.
33. Molchanov, P.; Mallya, A.; Tyree, S.; Frosio, I.; Kautz, J. Importance estimation for neural network pruning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 11264–11272.

34. Vanhoucke, V.; Senior, A.; Mao, M.Z. Improving the Speed of neuRAL Networks on CPUs. 2011. Available online: https://research.google.com/pubs/pub37631.html?source=post_page (accessed on 4 September 2023)
35. Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 525–542.
36. Courbariaux, M.; Hubara, I.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or –1. *arXiv* **2016**, arXiv:1602.02830.
37. Wang, L.; Dong, X.; Wang, Y.; Liu, L.; An, W.; Guo, Y. Learnable lookup table for neural network quantization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 12423–12433.
38. Li, L.; Zhang, S.; Wu, J. Efficient object detection framework and hardware architecture for remote sensing images. *Remote Sens.* **2019**, *11*, 2376. [[CrossRef](#)]
39. Caba, J.; Díaz, M.; Barba, J.; Guerra, R.; de la Torre, J.A.; López, S. Fpga-based on-board hyperspectral imaging compression: Benchmarking performance and energy efficiency against gpu implementations. *Remote Sens.* **2020**, *12*, 3741. [[CrossRef](#)]
40. Fan, H.; Liu, S.; Ferianc, M.; Ng, H.C.; Que, Z.; Liu, S.; Niu, X.; Luk, W. A real-time object detection accelerator with compressed SSDLite on FPGA. In Proceedings of the 2018 International Conference on Field-Programmable Technology (FPT), Naha, Japan, 10–14 December 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 14–21.
41. Zhang, N.; Wei, X.; Chen, H.; Liu, W. FPGA implementation for CNN-based optical remote sensing object detection. *Electronics* **2021**, *10*, 282. [[CrossRef](#)]
42. Yan, T.; Zhang, N.; Li, J.; Liu, W.; Chen, H. Automatic Deployment of Convolutional Neural Networks on FPGA for Spaceborne Remote Sensing Application. *Remote Sens.* **2022**, *14*, 3130. [[CrossRef](#)]
43. Tan, S.; Fang, Z.; Liu, Y.; Wu, Z.; Du, H.; Xu, R.; Liu, Y. An SSD-MobileNet Acceleration Strategy for FPGAs Based on Network Compression and Subgraph Fusion. *Forests* **2022**, *14*, 53. [[CrossRef](#)]
44. Wang, J.; Gu, S. Fpga implementation of object detection accelerator based on vitis-ai. In Proceedings of the 2021 11th International Conference on Information Science and Technology (ICIST), Chengdu, China, 21–23 May 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 571–577.
45. Chen, W.H.; Hsu, H.J.; Lin, Y.C. Implementation of a Real-time Uneven Pavement Detection System on FPGA Platforms. In Proceedings of the 2022 IEEE International Conference on Consumer Electronics-Taiwan, Taipei, Taiwan, 6–8 July 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 587–588.
46. Chu, Y.; Li, P.; Bai, Y.; Hu, Z.; Chen, Y.; Lu, J. Group channel pruning and spatial attention distilling for object detection. *Appl. Intell.* **2022**, *52*, 16246–16264. [[CrossRef](#)]
47. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. Yolov4: Optimal speed and accuracy of object detection. *arXiv* **2020**, arXiv:2004.10934.
48. Chen, P.; Liu, S.; Zhao, H.; Jia, J. Gridmask data augmentation. *arXiv* **2020**, arXiv:2001.04086.
49. Wu, D.; Lv, S.; Jiang, M.; Song, H. Using channel pruning-based YOLO v4 deep learning algorithm for the real-time and accurate detection of apple flowers in natural environments. *Comput. Electron. Agric.* **2020**, *178*, 105742. [[CrossRef](#)]
50. Fan, S.; Liang, X.; Huang, W.; Zhang, V.J.; Pang, Q.; He, X.; Li, L.; Zhang, C. Real-time defects detection for apple sorting using NIR cameras with pruning-based YOLOV4 network. *Comput. Electron. Agric.* **2022**, *193*, 106715. [[CrossRef](#)]
51. Liu, H.; Fan, K.; Ouyang, Q.; Li, N. Real-time small drones detection based on pruned yolov4. *Sensors* **2021**, *21*, 3374. [[CrossRef](#)]
52. Li, K.; Wan, G.; Cheng, G.; Meng, L.; Han, J. Object detection in optical remote sensing images: A survey and a new benchmark. *ISPRS J. Photogramm. Remote Sens.* **2020**, *159*, 296–307. [[CrossRef](#)]
53. Li, C.; Xu, R.; Lv, Y.; Zhao, Y.; Jing, W. Edge Real-Time Object Detection and DPU-Based Hardware Implementation for Optical Remote Sensing Images. *Remote Sens.* **2023**, *15*, 3975. [[CrossRef](#)]
54. Lyu, S.; Zhao, Y.; Li, R.; Li, Z.; Fan, R.; Li, Q. Embedded sensing system for recognizing citrus flowers using cascaded fusion YOLOv4-CF+ FPGA. *Sensors* **2022**, *22*, 1255. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.