Article

Efficient Object Detection in SAR Images Based on Computation-Aware Neural Architecture Search

Chuanyou Li [1,2,3,*], Yifan Li [2,3,4], Huanyun Hu [2,3], Jiangwei Shang [3,4,5], Kun Zhang [3,*], Lei Qian [3] and Kexiang Wang [6]

1. School of Computer Science and Engineering, Southeast University, Nanjing 211189, China
2. MOE Key Laboratory of Computer Network and Information Integration, Southeast University, Nanjing 211189, China
3. State Key Laboratory of Mathematical Engineering and Advanced Computing, Wuxi 214128, China
4. School of Cyber Science and Engineering, Southeast University, Wuxi 214128, China
5. School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China
6. Chinese Aeronautical Establishment, Beijing 100029, China

* Correspondence: cyli@seu.edu.cn (C.L.); zhang.kun@meac-skl.cn (K.Z.)

Abstract: Remote sensing techniques are becoming more sophisticated as radar imaging techniques mature. Synthetic aperture radar (SAR) can now provide high-resolution images for day-and-night earth observation. Detecting objects in SAR images is increasingly playing a significant role in a series of applications. In this paper, we address an edge detection problem that applies to scenarios with ship-like objects, where the detection accuracy and efficiency must be considered together. The key to ship detection lies in feature extraction. To efficiently extract features, many existing studies have proposed lightweight neural networks by pruning well-known models in the computer vision field. We found that although different baseline models have been tailored, a large amount of computation is still required. In order to achieve a lighter neural network-based ship detector, we propose Darts_Tiny, a novel differentiable neural architecture search model, to design dedicated convolutional neural networks automatically. Darts_Tiny is customized from Darts. It prunes superfluous operations to simplify the search model and adopts a computation-aware search process to enhance the detection efficiency. The computation-aware search process not only integrates a scheme cutting down the number of channels on purpose but also adopts a synthetic loss function combining the cross-entropy loss and the amount of computation. Comprehensive experiments are conducted to evaluate Darts_Tiny on two open datasets, HRSID and SSDD. Experimental results demonstrate that our neural networks win by at least an order of magnitude in terms of model complexity compared with SOTA lightweight models. A representative model obtained from Darts_Tiny (158 KB model volume, 28 K parameters and 0.58 G computations) yields a faster detection speed such that more than 750 frames per second (800 $\times$ 800 SAR images) could be achieved when testing on a platform equipped with an Nvidia Tesla V100 and an Intel Xeon Platinum 8260. The lightweight neural networks generated by Darts_Tiny are still competitive in detection accuracy: the F1 score can still reach more than 83 and 90, respectively, on HRSID and SSDD.

Keywords: ship detection; detection efficiency; neural architecture search; pruning; computation awareness

check for updates

Citation: Li, C.; Li, Y.; Hu, H.; Shang, J.; Zhang, K.; Qian, L.; Wang, K. Efficient Object Detection in SAR Images Based on Computation-Aware Neural Architecture Search. *Appl. Sci.* **2021**, *12*, 10978. https://doi.org/10.3390/app122110978

Academic Editors: Rui Yao and Hancheng Zhu

Received: 16 September 2022
Accepted: 26 October 2022
Published: 29 October 2022

1. Introduction

Synthetic aperture radar (SAR) can provide high-resolution images in day-and-night and all-weather earth observation. SAR techniques have played a significant role in applications of environmental monitoring, maritime management and resource exploration. With the rapid growth of SAR imaging techniques, an increasing number of studies have considered object detection in SAR images. In this paper, we study ship detection in SAR images towards edge scenarios, where detection efficiency should be taken into account along with detection accuracy. For example, SAR is typically mounted on battery-powered devices [1]. Due to limited energy and computation resources, any ship detector should not be computation intensive.

To detect ship targets in SAR images, many algorithms taking the advantages of neural networks in feature extraction have been proposed. In early studies, different dedicated neural networks were designed towards having better detection accuracy. Recent studies noticed that efficiency is important as well as accuracy, especially in edge scenarios. A series of lightweight neural networks were then proposed, where detection accuracy and efficiency were considered together.

According to our knowledge, existing lightweight neural networks for ship detection are mainly designed by pruning manually designed neural networks such as YOLO [2], Fast R-CNN [3], and MobileNet [4]. It is known that the performance of a neural network depends on its structure and the operations (e.g., standard convolution, depth-wise convolution, max pooling, average pooling, etc.) used. Thus, designs based on a well-known neural network are often not expected to have bad preliminary results. However, this methodology relies on artificial expertise and is subject to the existing neural architectures. Such a problem becomes more intractable when meeting two inconsistent targets: accuracy and efficiency. In many existing studies, although different baseline models are reduced on the premise of maintaining detection accuracy, a large amount of computation is still required. For example, in two recent YOLO-based lightweight ship detectors [5,6], the FLOPs (floating-point operations) required are, respectively, 1.94 G and 3.53 G. In this paper, we follow a different methodology, neural architecture search (NAS), to design neural networks. After a search space is specified, the neural network design process can be done automatically, and meanwhile, more possibilities in the network design space could be explored. In addition, detection accuracy and efficiency are combined together in our neural architecture search model. Hence, it is expected to generate a neural network that is more attractive in detection efficiency and is still competitive in detection accuracy.

The main body of our ship detector is composed of a backbone and a detection head. The backbone is used to extract features, and the detection head is used to identify targets. Note that most computations are required by the backbone. Clearly, the lighter the backbone is, the higher the detection efficiency that could be achieved. In order to generate a satisfactory backbone, techniques of Darts [7,8] are adopted. We first followed the recommended settings [7,8] to construct a ship detector. However, after testing on HRSID [9] and SSDD [10], we obtained no good results, only serious overfitting. Furthermore, we found that first conducting an architecture search by Darts and then pruning the neural network generated can hardly obtain sufficient improvements. Aiming at the problem, we customized a new architecture search model named Darts_Tiny. Compared with Darts, Darts_Tiny first prunes the super neural network used for the architecture search. In addition, a channel-pruning scheme is carried out along with the search process, and a synthetic loss function is used such that not only is the overfitting restrained, but also, the detection efficiency is enhanced.

Comprehensive experiments are conducted on two datasets, HRSID [9] and SSDD [10]. Experimental results demonstrate that the neural networks generated by Darts_Tiny can save at least an order of magnitude in model volume, the amount of parameter, and the amount of computation. A lighter model yields a faster detection speed: by inputting 800×800 SAR images, more than 750 frames per second (FPS) can be achieved.

The main contributions are summarized as follows:

1. We customized a new differentiable neural architecture search model, Darts_Tiny, to automatically generate neural network-based ship detectors. Compared with manual design, our methodology can explore more possibilities in the network design space and thus can handle well both the ship detection accuracy and efficiency.
2. The neural architecture search process of Darts_Tiny is computation aware, where a channel-pruning scheme and a synthetic loss function are adopted. The channel-pruning scheme suppresses excessive channels, and the synthetic loss function combines cross-entropy loss and the amount of computation.

3. Compared with existing models, Darts_Tiny can generate remarkably lightweight neural networks that not only have significant advantages in terms of model scale, computation complexity and detection speed but are also competitive in detection accuracy.

The rest of this paper is organized as follows. Section 2 sketches out the related work. Section 3 introduces the ship detection problem and necessary evaluation metrics. In Section 4, we introduce Darts and apply it to ship detection. By conducting experiments, we show that directly applying Darts can hardly obtain good results. In Section 5, we propose a new model, Darts_Tiny. The experimental setup and results are given in Section 6. Finally, Section 7 brings concluding remarks and discusses future work.

2. Related Work

2.1. Ship Detection via Neural Networks

Convolutional neural networks are well-known for their remarkable ability of feature extraction. Since J. Li et al. [10] proposed a labeled open dataset, SSDD, applying convolutional neural networks to ship detection in SAR images quickly received intense attention. At the beginning, studies mainly focused on designing a neural network having better detection accuracy. Based on SSDD, J. Li et al. [10] first proposed a ship detector based on Faster R-CNN, and then they conducted follow-up studies for further improvements [11,12]. To achieve better accuracy, many other solutions have been proposed. For example, H. Guo et al. [13] leveraged feature pyramids, Z. Zhang et al. [14] designed a multitask learning-based object detector, X. Geng et al. [15] presented boundary box optimization techniques, and Z. Sun et al. [16] proposed an anchor-free method aiming at improving ship detection in high-resolution SAR images. However, all these studies did not pay enough attention to the detection efficiency, such that they might not be adaptive in an environment with limited computation resources.

Recent studies on ship detection have noticed the importance of detection efficiency as well as accuracy. Aiming at the two inconsistent targets, current studies are often conducted based on existing manually designed neural networks, expecting to achieve a preliminary detection result. To obtain higher detection efficiency, either a lighter neural network backbone is used [17–20] or the baseline network is tailored [5,6,21,22]. Lightweight techniques sometimes decreasethe detection accuracy. Additional techniques are adopted as long as the detection accuracy needs to be compensated for [5,6].

Y. Li et al. [17] proposed a lightweight model based on Faster R-CNN. T. Zhang et al. [18] and S. Liu et al. [19] proposed neural networks that took the advantages of MobileNet. In the above work, all the new designed models can achieve faster inference speed than the baseline models, such as Faster R-CNN and YOLO. More recent studies were interested in designing a lightweight ship detector based on different YOLO versions. Z. Sun et al. [23] proposed an arbitrary-oriented ship detector based on the YOLO detection framework using bi-directional feature fusion and angular classification. The amount of the parameter was 19.57 M, and the model volume was 39.4 MB. M. Alkhaleefah et al. [21] proposed Accelereated-YOLOv3 for ship detection. Compared with YOLOv3 [24], a higher detection speed is achieved by using fewer channels and convolutional layers. S. Chen et al. [5] proposed an efficient SAR ship detector named Tiny YOLO-Lite, which was also designed based on YOLOv3. Tiny YOLO-Lite combines network pruning and knowledge distillation techniques to guarantee both efficiency and accuracy. The amount of computation is reduced to 1.94 G FLOPs, which is a fifteenth of that of YOLOv3. J. Jiang et al. [6] proposed YOLO-v4-light. Aiming at a faster detection speed, YOLOv4 [25] was tailored to reduce the model size and the number of parameters. To compensate for the loss of accuracy, YOLO-v4-light was refined for three-channel images. By cutting the amount of computation to 3.53 G FLOPs, the detection speed of YOLO-v4-light achieves rates three times faster than YOLOv4. X. Ma et al. [26] proposed Light-YOLOv4, where the model volume, parameter size, and FLOPs can been reduced by 98.63%, 98.66%, and 91.30% compared with YOLOv4. X. Xu et al. [22] proposed a model named Light-YOLOv5, where the amount of computation is 4.44 G FLOPs. In addition to standard YOLO versions, YOLOX [27], an anchor-free version, also received attention. Y. Feng et al. [20] proposed LPEDet based on YOLOX. To ensure higher detection efficiency, the original backbone Darknet-53 was replaced with a lighter network LCNet.

Existing studies show the effectiveness of designing based on a known neural network. However, this methodology heavily depends on artificial experiences: first selecting a reasonable baseline network and then performing delicate remolding to obtain both high efficiency and accuracy. On the other hand, the design space is also limited by the selected baseline network. Aiming at these problems, we present a different idea for ship detection. By adopting a neural architecture search, the neural network itself is designed automatically: the design process turns out to be a form of optimization. In this way, we could combine detection accuracy and efficiency together in the optimization and explore more possibilities in the network design space. Compared with existing studies, we succeed in moving a step further towards lightweighting: for example, the amount of computation can be decreased to less than 0.6 G FLOPs while still keeping competitive detection accuracy.

2.2. Neural Architecture Search

Neural architecture search (NAS) is a recently proposed technique that aims at designing aneural network automatically. Early NAS methods are time costly [28,29] due to the huge discrete architecture-searching space. In the past two years, Darts [7] and its follow-up studies, e.g., PC-Darts [8], Darts+ [30], Darts- [31], etc. transformed the discrete searching space to a continuous space, in which the gradient descent method was carried out to decrease the searching costs.

The methodology introduced in this paper is derived from Darts. However, there are two main differences compared with the Darts series [7,8,30,31]. First, for the specified ship detection task, we refine the search process, propose a tiny backbone to restrain overfitting, and thus improve the detection accuracy. Second, we took into account the number of channels and the amount of computation to implement computation awareness in the search process.

Adopting NAS for ship detection in SAR images is still in a start-up stage. Y. Wang et al. [32] proposed a lightweight neural network based on FBNet [33] where the network is determined by NAS. FBNet has a simple search space in which different operations are selected to fill in a chain-like network structure. In our work, we set a different search space that provides more resiliency in the network structure. On the other hand, our search process is refined to be computation aware such that the detection efficiency can be greatly improved.

3. Problem Description

We focus on efficient ship detection in SAR images. The open datasets used are HRSID [9] and SSDD [10]. Both include SAR images with different resolutions, polarizations, sea conditions, sea areas, and coastal ports. Figure 1 provides SAR images obtained in different scenarios. Our target is to demarcate each ship target accurately and efficiently (e.g., the green rectangular boxes in Figure 1 represent ships detected). By comparing the area with the ground truth (identified via the red rectangular box in Figure 1), we can calculate *Precision* (P), *Recall* (R), $F1$ score and *Average Precision* (AP) to evaluate detection accuracy [34]. On the other hand, to show that the neural network we generated is efficient in computation, we carry out comparisons by the number of computations (represented by floating-point operations—FLOPs) and frames per second (FPS). Detailed definitions of evaluation metrics are given below:

$$P = \frac{TP}{TP + FP}; \tag{1}$$

$$R = \frac{TP}{TP + FN}; \tag{2}$$

$$F1 = 2 \times \frac{P \times R}{P + R}; \tag{3}$$

$$AP = \int_0^1 P(R)dR; \tag{4}$$

TP (true positives), FP (false positives) and FN (false negatives) in Equations (1) and (2) are, respectively, the number of ship detected correctly, false alarms, and missing ships. To consider precision and recall together, we also adopt the F1 score in Equation (3) and AP in Equation (4). AP is defined by an integral equation, where P represents precision and R represents recall. $P(R)$ refers to a set of $\langle P, R \rangle$ pairs that are obtained by setting different confidence thresholds.

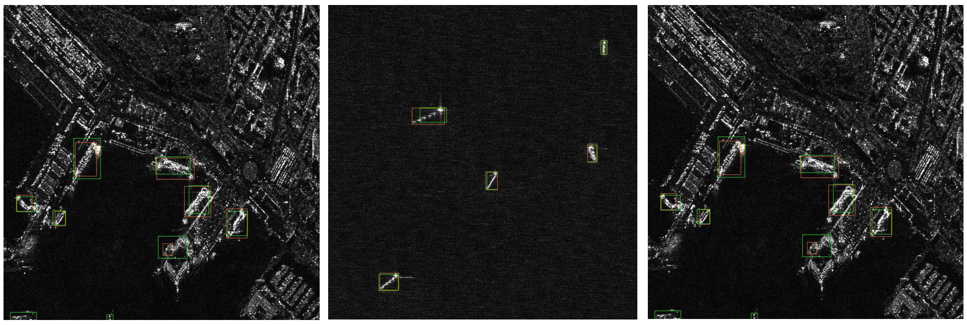


Figure 1. The problem of ship object detection.

4. Darts vs. Ship Detection

Figure 2 provides an overview of the target detection components (for ships). There are three parts: a backbone, a detection head and post-processing. A backbone is a neural network taking charge of the feature extraction. In this section, the backbone is designed by Darts techniques [7,8] that will be specified later. By considering computation efficiency, we introduce a simple detection head that is composed of Relu6 and a 1×1 convolution. Once the detection head process finishes, target identification occurs next. Note that at this moment, there exists more than one candidate rectangular box (each candidate has a confidence level) for each target. To output one final rectangular box for each target detected depends on post-processing, where non-maximum suppression is performed.

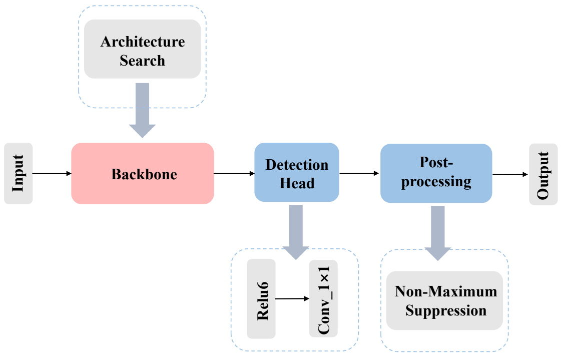


Figure 2. Overview: ship target detection.

In the following, we first apply techniques and settings directly from Darts [7] and PC-Darts [8] to construct the backbone.

4.1. Backbone

Figure 3 depicts a neural network-based backbone that is composed of 2 stems, 12 NCs (normal cells) and 2 RCs (reduction cells). The integer associated with each block is the number of channels. The two stems include 3 standard 3×3 convolutions (1 for stem 0 and 2 for stem 1). All normal cells have the same inner composition. Reduction cells expand the number of channels while reducing the size of the feature maps. Similar to normal cells, the two reduction cells are homogeneous in terms of their inner composition. Both the normal cells and reduction cells are generated via a neural architecture search.

We use the Darts technique to search through the normal and reduction cells to construct the backbone in Figure 3. The search process is implemented over a super neural network depicted in Figure 4. The super neural network is composed of two stems, six normal cells and two reduction cells. The two stems are composed of 3 standard 3×3 convolutions, while the eight cells are defined by a uniform template drawn in Figure 5. In the cell template, different operations are defined by weight sharing, where the discrete architecture search will turn out to be a differentiable weight-tuning process.

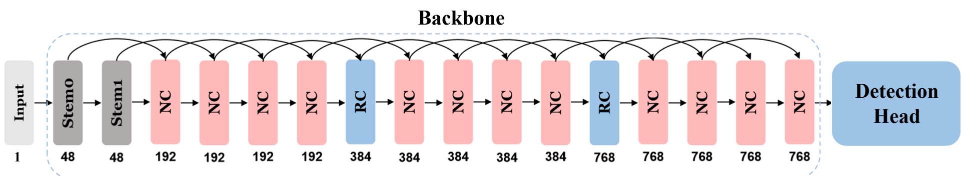


Figure 3. The composition of backbone.

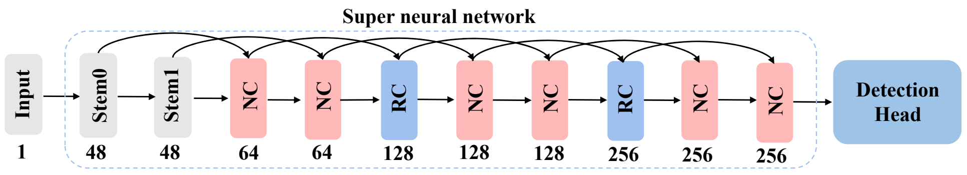


Figure 4. The super network for architecture search.

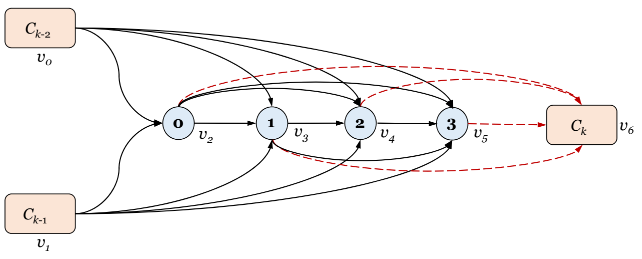


Figure 5. Cell template.

Let us specify the cell template first. It can be seen that the whole cell template is constructed by a directed acyclic graph consisting of 7 nodes $\{v_0, v_1, ..., v_6\}$. Cell C_k takes the outputs of cell C_{k-2} and cell C_{k-1} as inputs. For an easier representation, we also use the notation C_k to represent the output of cell C_k. Inside the cell template, there are four internal states numbered by 0, 1, 2, and 3, each of which can take C_{k-2}, C_{k-1} and the preceding states as inputs. For example, for the internal state 1, its inputs could be C_{k-2}, C_{k-1} or the internal state 0 but never the internal state 2 or 3. Hence, in Figure 5, there are 14 directed edges expressing state transitions, each of which integrates 8 synthetic operations (details are provided in Table 1). The 4 red dash edges in Figure 5 contain no operations. They are used to concatenate the four internal states together as the output of cell C_k.

Table 1. Synthetic operations.

Synthetic Operations	Basic Operations
skip_connect	add two edges
max_pool_3×3	max pooling
avg_pool_3×3	average pooling
sep_conv_3×3	depth-wise separable convolution for 2 times with 3×3 depth-wise kernel
sep_conv_5×5	depth-wise separable convolution for 2 times with 5×5 depth-wise kernel
dil_conv_3×3	3×3 dilation convolution followed by a point-wise convolution
dil_conv_5×5	5×5 dilation convolution followed by a point-wise convolution
zero	no operation

Let $\mathcal{O}$ be the set of 8 synthetic operations. Note that to obtain lightweight models, $\mathcal{O}$ includes depth-wise separable convolution rather than standard convolution. The searching process aims at selecting one synthetic operation from $\mathcal{O}$ for each directed edge (not including the dashed ones). To alleviate performance crash, we shall perform a two-level weight sharing [8]. First, for a directed edge (v_i, v_j), 8 synthetic operations formulate the information propagated from v_i to v_j as a weighted sum. Second, for a node v_i, all its incoming edges (information) are also formulated by a weighted sum.

$$f_{i,j}(\mathbf{x}_i) = \sum_{o \in \mathcal{O}} \frac{exp\{\alpha_{i,j}^{o}\}}{\sum_{o' \in \mathcal{O}} exp\{\alpha_{i,j}^{o'}\}} o(\mathbf{x}_i^1) + \mathbf{x}_i^2 \tag{5}$$

The above Equation (5) represents the first level of weight sharing, where $\mathbf{x}_i$ is the output of node v_i and $\alpha_{i,j}^{o}$ is a hyper-parameter for weighting operation $o(\mathbf{x}_i)$. Note that we also leverage partial channel connections where only $1/K$ channels are selected. This involves a channel sampling mask $\mathbf{S}_{i,j}$ in Equation (5): $\mathbf{x}_i^1 = \mathbf{S}_{i,j} * \mathbf{x}_i$ and $\mathbf{x}_i^2 = (1 - \mathbf{S}_{i,j}) * \mathbf{x}_i$ [8].

$$\mathbf{x}_j = \sum_{i<j} \frac{exp\{\beta_{i,j}\}}{\sum_{i'<j} exp\{\beta_{i',j}\}} f_{i,j}(\mathbf{x}_i) \tag{6}$$

Equation (6) represents the second level of weight sharing, where $\mathbf{x}_j$ is the output of node $v_j \in \{v_2, v_3, v_4, v_5\}$ and $\beta_{i,j}$ is the normalization parameter of (v_i, v_j). The searching process will decide each $\alpha_{i,j}^{o}$ and $\beta_{i,j}$ by gradient descent.

When the search is completed, a neural network is derived based on the weights obtained. Within a cell, for each internal node $v_j \in \{v_2, v_3, v_4, v_5\}$, two synthetic operations are selected, respectively, for v_j's two incoming edges. Specifically, each incoming edge of node v_j is first assigned a synthetic operation with the greatest weight calculated by Formula (7). Then, we select the top two incoming edges of v_j according to the weights assigned.

$$\max_{o \in \mathcal{O}} \frac{exp\{\beta_{i,j}\}}{\sum_{i'<j} exp\{\beta_{i',j}\}} \cdot \frac{exp\{\alpha_{i,j}^{o}\}}{\sum_{o' \in \mathcal{O}} exp\{\alpha_{i,j}^{o'}\}} \tag{7}$$

A brief description of searching is given in Figure 6. To perform the differentiable architecture search, the two-level weight sharing scheme [8] is implemented first (line 1). Then, the neural network parameters are updated by m epochs (line 2), which is a necessary pre-training step by which an architecture search could be carried out more stably. Lines

3–6 are the process of architecture search. Each time the architecture is updated (line 4), an epoch of training is carried out (line 5).

```
1: Implement weight sharing represented by Equations (5) and (6);
2: update neural network parameters for $m$ epochs (by training);
3: while not finished do
4:     update each architecture parameter $\alpha^o_{i,j}$ and $\beta_{i,j}$ by gradient descent;
5:     update neural network parameters (by training);
6: end while
7: Derive the final architecture based on the learned $\alpha^o_{i,j}$ and $\beta_{i,j}$;
```

Figure 6. The algorithm steps of Darts.

4.2. Testing Results and Analysis

To test the performance of Darts, we implement comprehensive evaluations on the dataset HRSID [9], where 90% of SAR images are randomly selected for the architecture search and neural network training, and the remaining 10% of SAR images are left for performance validation. IoU (intersection over union) is set to 0.3, and the threshold of confidence is set to 0.15. We perform a neural architecture search multiple times and select the best three of them (named HArch_1, HArch_2, and HArch_3) to show the results. Cell structures are depicted in Appendix A, Figure A1. According to the Darts settings, 12 normal cells and 2 reduction cells are stacked to construct the backbone (see Figure 3).

Table 2 demonstrates that the detection results are not good: for all three neural networks generated, F1 is below 75, recall is below 69, and precision is around 80. Note that for all the results, there is a big gap between the validation and training results, which is typical overfitting phenomena. To increase the detection accuracy while preventing overfitting, we perform two straightforward methods. One is early-stopping: cutting down the number of training epochs. However, we observe that overfitting appears in the early epochs. When overfitting first appears, the detection results are even worse than those given in Table 2. The second way is to simplify the neural network. What we did was to reduce the number of stacked normal cells. Experimental results are given in Table 3. We can see that stacking 8 or 4 cells to construct the backbone will obtain better results than stacking 14 cells. Surprisingly, even if the backbone is only composed of 2 reductions cells (by suppressing all normal cells), the results are still competitive. Unfortunately, although we cut down the number of cells, overfitting is still significant. We also perform similar experiments on SSDD [10] (Table 4; cell structures are depicted in Appendix A Figure A2). There are similar observations: cutting down the number of cells can obtain benefits, however, overfitting still exists.

According to these experimental results, we have two observations. One is that simplifying the neural network generated by Darts is potentially good for ship detection. On the other hand, performing an architecture search first and then roughly decreasing the number of stacked cells is far from enough. Hence, it is necessary to customize a new model for ship detection.

Table 2. Applying Darts directly for ship detection on HRSID.

Archs	Cells	F1		Recall		Precision	
		Validation	Training	Validation	Training	Validation	Training
HArch_1	14	74.6	90.0	68.4	84.5	82.1	96.2
HArch_2	14	71.0	86.8	63.8	81.2	80.1	93.3
HArch_3	14	68.6	86.4	62.1	80.8	76.7	92.8

Table 3. Detection results on HRSID after reducing the number of cells.

Archs	Cells	F1		Recall		Precision	
		Validation	Training	Validation	Training	Validation	Training
HArch_1	8	76.4	90.9	70.3	85.6	83.5	97.0
	4	76.1	90.4	71.3	85.8	81.7	95.4
	2	73.1	91.1	66.7	85.8	81.0	97.0
HArch_2	8	74.1	89.7	67.7	84.7	81.7	95.3
	4	76.1	91.8	71.1	87.1	81.9	96.7
	2	72.5	90.9	65.7	85.4	80.8	97.1
HArch_3	8	73.7	87.7	69.5	85.0	78.6	90.5
	4	74.7	91.6	68.8	86.5	81.8	97.2
	2	70.7	89.9	64.5	84.2	78.3	96.5

Table 4. Detection results on SSDD after reducing the number of cells.

Archs	Cells	F1		Recall		Precision	
		Validation	Training	Validation	Training	Validation	Training
SArch_1	14	89.01	94.04	86.10	91.12	92.13	97.15
	8	90.78	96.46	88.56	94.02	93.12	99.03
	4	89.88	96.95	88.28	95.12	91.53	98.85
	2	88.70	96.30	85.56	93.93	92.08	98.79
SArch_2	14	86.58	91.40	81.74	88.31	92.03	94.72
	8	89.20	95.87	86.85	93.37	91.91	98.50
	4	91.52	97.14	88.28	95.26	95.02	99.09
	2	88.30	96.98	86.38	95.35	90.31	98.67
SArch_3	14	87.83	92.71	85.56	90.06	90.23	95.51
	8	89.24	96.36	85.83	93.88	92.92	98.98
	4	90.71	97.10	89.10	95.35	92.37	98.90
	2	86.94	96.70	82.56	94.34	91.82	99.18

5. Methodology

In this section, we propose Darts_Tiny, a customized neural architecture search model for edge-based ship detection. According to the knowledge obtained from Section 4.2, Darts_Tiny is designed to remold the architecture search from two aspects. First, the super neural network built by original Darts is pruned. Second, the process of architecture search is refined.

5.1. Pruning the Super Neural Network

Figure 7 depicts the super neural network leveraged by Darts_Tiny. Compared with the one depicted in Figure 4, the super neural network used here is much lighter, where one stem together with three cells constitutes the backbone.

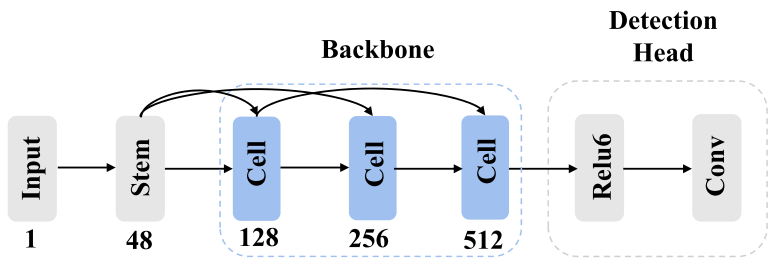


Figure 7. Darts_Tiny.

The customization for ship detection is conducted through dedicated pruning. In the original Darts settings, there are three standard convolution layers that constitute the two stem blocks. According to our analysis, they make part of the overfitting. Thus, in Darts_Tiny, we retain one stem block, where only a standard convolution layer is included. After the stem, we stack three cells. The three cells used here are uniform (no longer divided into normal cells and reduction cells). The cell's inner structure also follows the template depicted in Figure 2, and the concrete structure will be decided by our neural architecture search algorithm. Each of the three cells expands the number of channels while diminishing the size of feature maps. Finally, Relu6 and Conv_1 $\times$ 1 make up the detection head.

5.2. *Computation-Aware Searching*

Based on the pruned super neural network, we propose a computation-aware searching process. Generally, we leverage two schemes. The first one is channel pruning, and the second one is a synthetic computation-aware loss function.

The architecture search process that integrates channel pruning is shown in Figure 8. Generally, the search process flows into the differentiable architecture search. The difference is that channel pruning could be carried out in several epochs (lines 6–11). Line 6 is a test that checks whether it is the right time to prune the channels. To pass the test, two conditions need to be satisfied. The first one focuses on the gap between training and validation precision. When this gap is greater than a predefined threshold T_1, we consider the current overfitting to be noteworthy. Note that, based on this gap, simply pruning a channel is hasty. In the early epochs, because the status is not stable, a large gap could appear between the training and validation precision. To carry out channel pruning in the early stages where the epochs are not stable is not reasonable. Therefore, we add the second condition that requires that the validation precision is greater than another predefined threshold T_2. This condition ensures channel pruning can only be carried out during a stable status. When the test in line 6 is passed, Darts_Tiny cuts off half of the channels (line 7) to restrain overfitting. Note that channel shrinkage is not a differentiable operation, e.g., additional training steps are performed to adjust the neural network parameters (line 9) until the validation precision recovers to the third predefined threshold T_3 (line 8).

```
 1: Implement weight sharing represented by Equations (5) and (6);
 2: update neural network parameters by gradient descent for $m$ epochs;
 3: while not finished do
 4:     update each architecture parameter $\alpha^o_{i,j}$ and $\beta_{i,j}$ by gradient descent;
 5:     update neural network parameters by gradient descent;
 6:     if (training precision—validation precision) > $T_1$ & validation precision > $T_2$ then
 7:         suppress half of the channels;
 8:         while validation precision < $T_3$ do
 9:             update neural network parameters by gradient descent;
10:         end while
11:     end if
12: end while
13: Derive the final architecture based on the learned $\alpha^o_{i,j}$ and $\beta_{i,j}$;
```

Figure 8. The algorithm steps of Darts_Tiny.

The super neural network is learned once the architecture search finishes. The termination condition in line 3 could be set to the number of searching epochs, training precision or validation precision. The method of deriving a neural network from the super network mainly follows the settings introduced in Section 4.1. The difference is that the derived neural network follows the main structure of the super network (depicted in Figure 7) in such a way that only three cells are stacked, i.e., the neural network generated has one stem, then three cells and a detection head.

Channel pruning decreases the number of computations, which not only restrains overfitting but also improves computational efficiency. In addition to channel pruning, we also leverage a computation-aware loss function to enhance computation efficiency. The computation-aware loss function used is represented by the Formula (8). We can see that $Loss$ is the summation of two parts, where $Loss\_ce$ is the cross-entropy loss, and Ops is the total number of multiply-and-accumulate calculations involved. For these two parts, $Loss\_ce$ is related to inference accuracy, and Ops is related to computation efficiency. ops_ratio is the parameter used to balance the two parts in the loss function. To decrease $Loss$, both $Loss\_ce$ and Ops should be decreased such that the target neural network searched is expected to perform well in both inference accuracy and computation efficiency.

$$Loss = (1 - ops\_ratio) \times Loss\_ce + ops\_ratio \times log(Ops) \tag{8}$$

Ops is obtained by adding the number of multiply-and-accumulate calculations used in different operations. Formulas (9)–(12) provides our method of counting the number of multiply-and-accumulate calculations. $Ops_{pooling\_3\times3}$, Ops_{depth_wise}, Ops_{point_wise} and Ops_{dil_conv}, respectively, represent the total multiply-and-accumulate calculations of $pooling\_3 \times 3$ ($avg\_pool\_3 \times 3$ and $max\_pool\_3 \times 3$), depth-wise convolution, point-wise convolution and dilated convolution. C stands for the input channel number, M stands for the output channel number, P and Q stand, respectively, for the height and width of the output feature map, and ker_size is the size of the kernel weight.

$$Ops_{pooling\_3\times3} = 3 \times 3 \times C \times P \times Q \tag{9}$$

$$Ops_{depth\_wise} = ker\_size^2 \times C \times P \times Q \tag{10}$$

$$Ops_{point\_wise} = C \times M \times P \times Q \tag{11}$$

$$Ops_{dil\_conv} = ker\_size^2 \times C \times P \times Q \tag{12}$$

6. Experimental Results

In this section, extensive experiments are carried out to evaluate Darts_Tiny. We first present details on datasets and settings. Then, we provide experimental results and our analysis.

6.1. Datasets and Settings

We performed evaluations over two datasets named HRSID [9] and SSDD [10]. HRSID was proposed by using images from 99 Sentinel-1B images, 36 TerraSAR-X, and 1 TanDEM-X image. These images were then cropped to 800 × 800 pixel images. Overall, there are 5604 images, including 16,951 ships. The ship resolution ranges from 1 m to 15 m. SSDD is a multi-size and multi-resolution dataset. It has 1160 SAR images, including 2456 ships. The sizes of the SAR images in SSDD range from 7 × 7 to 211 × 298 pixel images. In order to maintain consistency, we resized the images of the SSDD set to 800 × 800 pixels. All our experiments were implemented by using PyTorch 0.4.1 and CUDA 9.1. Our hardware platform consisted of an Nvidia Tesla V100 (32 G) GPU and an Intel Xeon Platinum 8260 CPU.

For both datasets, HRSID and SSDD, we randomly select 90% of images for the architecture search and training. The remaining 10% of images are left for performance validation. The number of epochs is set to 100, and the batch size is set to 32 for the architecture search. After a neural network is derived by Darts_Tiny, sufficient trainings are performed, while the learning rate is initialized to be 1.

The first 15 epochs of 100 are used to train the super neural network (line 2 in Figure 8). Remember that in the search process of Darts_Tiny (see Figure 8), we have three thresholds, T_1, T_2 and T_3, for performing channel pruning. In our experiments, training and validation precision are expressed by F1. According to our experience, T_1 is set to 3, and T_2 is set to 75 such that only when the gap between training and validation precision is greater than 3 and the current validation precision (F1) is greater than 75, channel pruning is performed. The third parameter T_3 is set to 70 such that after channel pruning, training is performed until F1 of validation recovers to 70. In our experiments, $T_3 = 70$ can be easily reached after a few training epochs. We use the loss function of YOLOv2 [35] to implement $Loss\_ce$ in Formula (8). The detection threshold IoU is set to 0.3. Unless otherwise specified herein, F1 is obtained by setting the threshold of confidence score to 0.15.

6.2. *Results of Channel Pruning*

Channel pruning is an important scheme integrated into Darts_Tiny. In this section, we shall visualize the effect of channel pruning by depicting the variation of F1 through a complete neural architecture search over HRSID and SSDD. Evaluation results are given in Figures 9 and 10. The horizontal axis represents the number of epochs. The vertical axis represents the F1, obtained after each epoch, that reflects the detection accuracy. In the results from HRSID, channel pruning is executed 4 times in epochs 5, 10, 32 and 40 respectively. It is not surprising to see that the F1 validation score falls sharply in these epochs as channel pruning is not differentiable. However, we can see that the F1 validation score recovers quickly after a few epochs of training. Thus, channel pruning does work in restraining overfitting. We can see that after the 41st epoch, the overfitting is stable. As for the results from SSDD, channel pruning is triggered 3 times. Because the scenarios in SSDD are simpler, overfitting here is not as serious as what was achieved in HRSID. However, channel pruning also works in SSDD and can restrain overfilling under a small threshold. For both datasets, the variation in F1 (especially the F1 validation score) via channel pruning does not show a negative impact on detection accuracy.

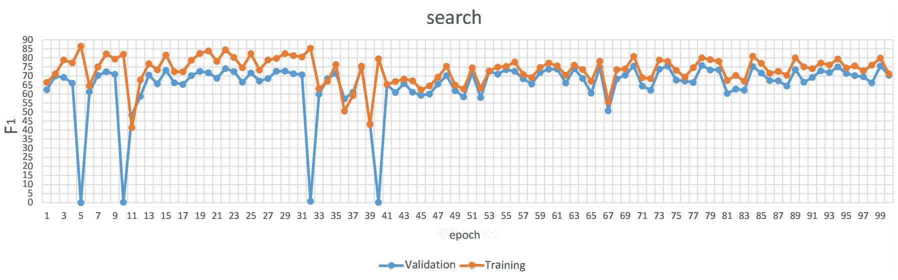


Figure 9. The effect of channel pruning on HRSID: pruning is triggered 4 times. After that, overfitting (the gap between two curves) is restrained.

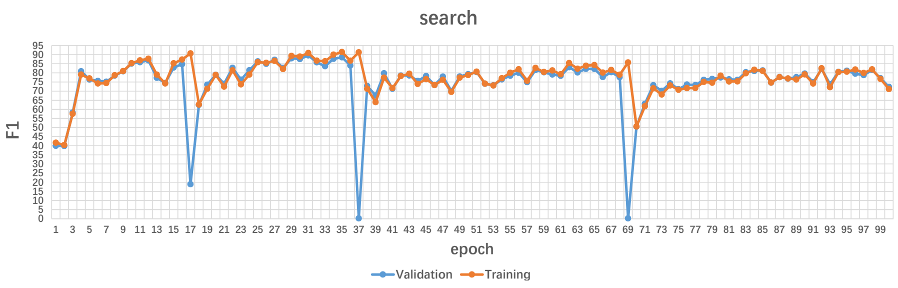


Figure 10. The effect of channel pruning on SSDD: pruning is triggered 3 times. As the scenarios in SSDD are simpler, overfitting is not as serious as what was achieved with HRSID. However, F1 validation score recovers quickly after a few epochs of training.

6.3. Searching Results via Darts_Tiny

In this section, we evaluate Darts_Tiny by using HRSID and SSDD. We leverage the super neural network depicted in Figure 7 to perform a neural architecture search and then derive several neural networks with various numbers of channels. The neural architecture search is fixed to 100 epochs. The time costs are around 36 hours (running on 4 Nvidia Tesla V100 GPUs) on average. Experimental results on HRSID are provided in Table 5.

Table 5. Detection results on HRSID (the first four are obtained by setting $ops\_ratio = 0$, and the last four are obtained by setting $ops\_ratio = 0.012$).

Architectures	Channels	Paras	FLOPs	AP	F1
H_Arch_32	32×4	280 K	6.89 G	85.09	85.636
H_Arch_16	16×4	79 K	1.84 G	80.76	81.868
H_Arch_8	8×4	28 K	0.62 G	82.03	82.270
H_Arch_4	4×4	10 K	0.21 G	80.38	80.172
H_Arch_C_32	32×4	246 K	5.64 G	79.726	80.515
H_Arch_C_16	16×4	79 K	1.64 G	82.826	83.630
H_Arch_C_8	8×4	28 K	0.58 G	83.93	83.158
H_Arch_C_4	4×4	9 K	0.18 G	79.35	78.744

The given results include the number of channels, the amount of parameters ("Paras"), the amount of computation ("FLOPs"), AP and F1. The number of channels is given in the form of "$x \times 4$", as there are 4 internal states (see Figure 5). For each state, channel pruning will cut down half of its channels. The amount of parameters and the amount of computation reflects the scale of a neural network. Generally, fewer parameters and computations lead to higher computation efficiency. The amount of computation is represented by FLOPs including all multiply-and-accumulate calculations. F1 is a popular criterion in object detection. In Darts_Tiny, we mainly focus on F1 to represent the detection accuracy and take AP as a secondary reference. Both F1 and AP synthetically consider precision and recall.

We provide a sequence of architectures with different channels generated during one search process. In Table 5, there are eight architectures. The first four are obtained by setting $ops\_ratio = 0$, while the last four are obtained by setting $ops\_ratio = 0.012$. Compared with the results in Tables 2 and 3, all the eight architectures derived by Darts_Tiny have a substantial improvement in detection accuracy. On the other hand, when the number of channels is cut down, both the amount of parameters and computations decrease sharply. Clearly, channel pruning enhances efficiency.

From the first four architectures, we can see that as the amount of computation decreases, both AP and F1 present a downward trend. However, by considering the amount of computation saved, an acceptable loss in detection accuracy is reasonable especially in an environment without sufficient computation resources. For example, compared with H_Arch_32, H_Arch_8 only costs 10% of the amount of computation. The last four architectures in Table 5 show something different. H_Arch_C_32 does not have the best performance in terms of F1. When generating the last four architectures, the loss function is composed of the cross-entropy loss and the amount of computations (see Equation (8)). We observe that the architecture search in this case usually needs more epochs to obtain a superior neural network. Therefore, H_Arch_C_32 generated in an early epoch is not as good as H_Arch_C_16 and H_Arch_C_8 generated in later epochs. Finally, by comparing the first four with the last four neural architectures, we can find that integrating Ops in the loss function succeeds in further reducing the number of computations.

We conduct the same evaluations by using SSDD to test whether Darts_Tiny can perform consistently in different datasets. Evaluation results are given in Table 6. Overall, Darts_Tiny also works well on SSDD. As channels are cut down, the number of parameters and the number of computations decrease, which enhances computation efficiency. On the other hand, channel pruning does not bring apparent negative impacts on AP and F1.

Compared with HRSID, SSDD has fewer SAR images and simpler scenarios. Consequently, even Darts_Tiny cuts down the number of channels to 4 × 4, so the detection accuracy is still competitive. Finally, combined with the results given in Tables 5 and 6, we can notice that the detection accuracy and efficiency are balanced when the number of channels is equal to 8 × 4.

Table 6. Detection results over SSDD (the first four are obtained by setting $ops\_ratio = 0$, and the last four are obtained by setting $ops\_ratio = 0.003$).

Architectures	Channels	Paras	FLOPs	AP	F1
S_Arch_32	32 × 4	274 K	5.96 G	91.35	88.46
S_Arch_16	16 × 4	84 K	1.90 G	93.65	91.57
S_Arch_8	8 × 4	30 K	0.59 G	92.28	90.29
S_Arch_4	4 × 4	12 K	0.22 G	90.99	88.68
S_Arch_C_32	32 × 4	251 K	5.99 G	91.68	89.65
S_Arch_C_16	16 × 4	59 K	1.62 G	87.41	88.98
S_Arch_C_8	8 × 4	24 K	0.54 G	89.61	90.24
S_Arch_C_4	4 × 4	9 K	0.21 G	89.14	88.56

6.4. Detection Speed in FPS

In Section 6.3, we used the amount of parameters and the amount of computations to represent the computation efficiency. In this section, we adopted frames per second (FPS) to show the detection speed. Remember that the whole process of ship detection not only includes a stage processed by a neural network but also a post-processing stage used to decide a unique rectangular box. As FPS is decided by both the stages, we separated them in the evaluation results. We used an Nvidia Tesla V100 to conduct the testing on HRSID and SSDD and set $batchsize = 128$. Results are provided in Tables 7 and 8.

Table 7. Computation efficiency in FPS (HRSID).

Architectures	Channels	FLOPs	Net	Post	All	Post$_{mt}$	All$_{mt}$
H_Arch_32	32 × 4	6.89 G	279	492	178	3955	261
H_Arch_16	16 × 4	1.84 G	561	488	261	3923	491
H_Arch_8	8 × 4	0.62 G	1028	481	327	3859	812
H_Arch_4	4 × 4	0.21 G	1855	475	379	3764	1243
H_Arch_C_32	32 × 4	5.64 G	256	563	176	4525	242
H_Arch_C_16	16 × 4	1.64 G	496	551	261	4430	446
H_Arch_C_8	8 × 4	0.58 G	936	560	312	4501	775
H_Arch_C_4	4 × 4	0.18 G	1828	555	377	4452	1297

Table 8. Computation efficiency in FPS (SSDD).

Architectures	Channels	FLOPs	Net	Post	All	Post$_{mt}$	All$_{mt}$
S_Arch_32	32 × 4	6.89 G	245	561	170	4512	232
S_Arch_16	16 × 4	1.84 G	521	576	261	4629	468
S_Arch_8	8 × 4	0.62 G	958	578	360	4645	794
S_Arch_4	4 × 4	0.21 G	1704	529	402	4249	1216
S_Arch_C_32	32 × 4	5.96 G	267	497	173	3992	249
S_Arch_C_16	16 × 4	1.80 G	610	491	269	3945	528
S_Arch_C_8	8 × 4	0.59 G	1048	497	337	3994	830
S_Arch_C_4	4 × 4	0.22 G	1769	497	388	3995	1227

In the two given tables, there are five columns referring to FPS. Let us first focus on the first three columns referring to FPS, where "Net" represents the processing speed of the neural network stage, "Post" represents the speed of the post-processing stage, and

"All" represents the overall detection speed. When the number of channels decreases, the processing speed of the neural network stage ("Net") increases sharply. However, the speed of the post-processing stage ("Post") is more or less stable. Although when combining "Net" and "Post" together, the overall detection speed ("All") increases, the post-processing stage gradually becomes the new bottleneck. Post-processing takes charge of selecting a unique rectangular box for each ship detected, which has a somewhat fixed overhead. Meanwhile, this program was executed on the CPU. In order to handle the new bottleneck, we adopted multi-threads to accelerate the post-processing stage. "Post_{mt}" represents the speed of post-processing by adopting 10 threads. It can be seen that more or less 8 times acceleration for post-processing could be achieved. Aided by multi-threads, post-processing is no more the bottleneck, and the overall detection speed ("All_{mt}") can be generally decided by the part of "Net".

Next, we set different values on *batchsize* to see the variation in FPS. We selected four neural architectures, respectively, from HRSID and SSDD to do the testing. Experimental results are given in Table 9. The entries filled by n/a are due to "out of memory failure". We can see that on both datasets, FPS ("Net") and FPS ("All_{mt}") can benefit from a greater *batchsize*. FPS ("Post_{mt}") is not so sensitive to *batchsize*. When *batchsize* is set to 1, it can be seen that FPS in all the three columns decreases significantly. Running the first batch is time costly due to the fact that GPU has a warm-up time to start the pipeline. Thus, setting a larger *batchsize* can share the costs of the warm-up and thus increase the overall efficiency of inference. On the other hand, the scale of the neural network decides the maximal *batchsize* for a given GPU and a given set of images. Usually, a lightweight neural network wins a greater *batchsize*. Note that this is another advantage of our Darts_Tiny, as it aims at generating lightweight neural networks.

Table 9. FPS with different batchsizes.

Architectures	Batch Size	Net	Post_{mt}	All_{mt}
H_Arch_32	500	n/a	n/a	n/a
	128	279	3955	261
	16	277	3918	259
	8	274	3863	256
	1	180	3693	172
H_Arch_4	500	1886	3857	1267
	128	1855	3764	1243
	16	1747	3870	1204
	8	1584	3594	1099
	1	202	3638	192
S_Arch_C_32	500	n/a	n/a	n/a
	128	267	3992	249
	16	265	3966	248
	8	262	3935	246
	1	172	3655	164
S_Arch_C_4	500	n/a	n/a	n/a
	128	1769	3994	1226
	16	1667	3613	1141
	8	1561	3653	1085
	1	186	3759	177

6.5. Comparisons

In this section, we conduct comparisons with state-of-the-art methods in the computer vision field: Fast R-CNN [3], YOLOv2 [35], YOLOv2 tiny [36], YOLOv4 [25], YOLOv4 tiny [36], YOLOv5 [37] and MobileNet [4]. Fast R-CNN was the first neural network used to study ship detection [10]. YOLO series are often taken as a baseline to construct lightweight ship detectors [5,6,21,22], and MobileNet is often adopted as a lightweight backbone [18,19].

In the comparisons, we also use the detection head and the cross-entropy loss function of Darts_tiny to construct a MobileNet-based ship detector. In addition, we keep the default parameters for each neural network and do not use special tricks to improve the performance on purpose.

In Figures 11 and 12, we depict two scatter diagrams to generally show our results and the comparison baselines. In both scatter diagrams, the horizontal axis represents F1 score obtained on HRSID. In Figure 11, the vertical axis expresses the detection speed in terms of FPS (log_2). Note that to clearly show each point, we use a logarithmic value rather than the original FPS. Our results are mainly distributed over the top right corner, demonstrating that the neural networks generated by Darts_Tiny are competitive in both accuracy and efficiency. In Figure 12, the horizontal axis represents the model volume (log_2). Similarly, we depict logarithmic values in the diagram. Our results locate over the bottom right corner, implying that the neural networks generated are light and not bad in detection accuracy.

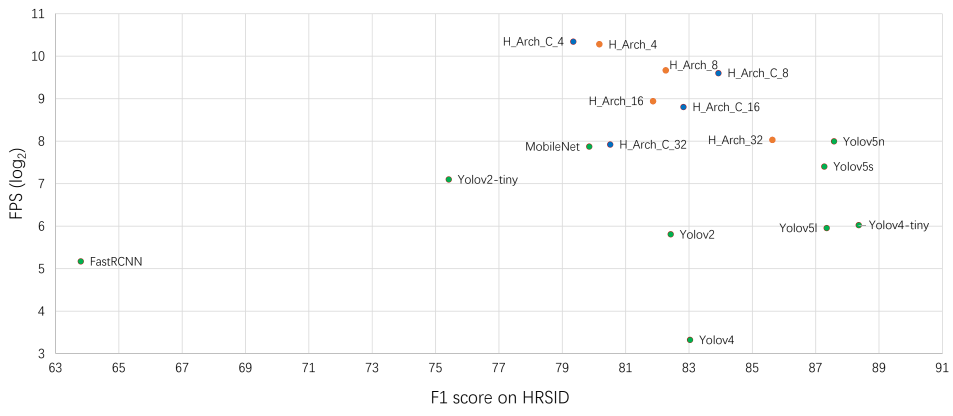


Figure 11. Scatter diagram: FPS vs. F1.

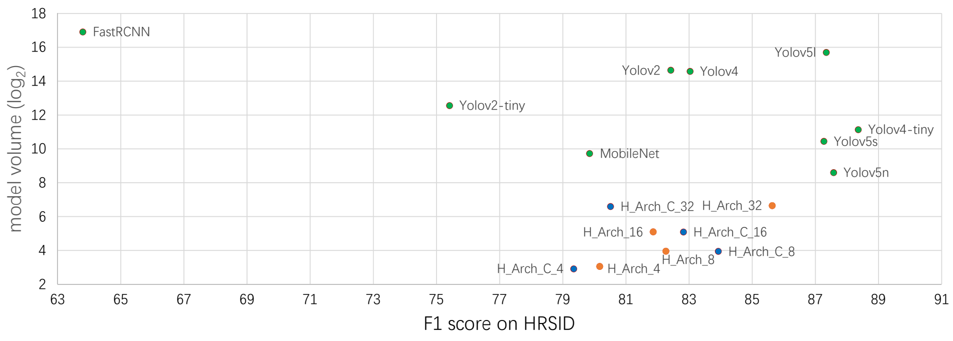


Figure 12. Scatter diagram: model volume vs. F1.

Detailed comparison results are given in Table 10. The amount of parameters, model volume, the amount of computations (represented by FLOPs, 800 × 800 images), epochs of training, time/epoch (many factors, even including coding quality, can affect the absolute training time. For the sake of fairness, we use the training epochs required for convergence and the average time costs per epoch together to represent the training costs. One epoch here is a round that all images in the training set are put into the model for one time.), AP, F1 and FPS are listed. We selected two architectures, H_Arch_32 and H_Arch_C_8, to join the comparison, as the former has the best performance in detection accuracy and the latter demonstrates a good balance in the two criteria. Generally, H_Arch_32 and H_Arch_C_8 achieve a significant advantage in the amount of parameters, model volume and FLOPs. Clearly, H_Arch_C_8 achieves a prominent detection speed in terms of FPS. Furthermore, the two neural networks do not show a distinct weakness in training costs (Fast R-CNN consumes 20 epochs to converge. However, it costs more in each epoch, and on the other hand, the low detection accuracy implies that additional training epochs are useless.), and they are still competitive in detection accuracy.

Table 10. Comparison results on HRSID.

Architectures	Parameters	Volume	FLOPs	Epochs	Time/Epoch	AP	F1	FPS
H_Arch_32	280 K	1.10 MB	6.89 G	80	4 m	85.09	85.636	261
H_Arch_C_8	**28 K**	**158 KB**	**0.58 G**	97	2 m	83.93	83.158	**775**
MobileNet	1.66 M	8.45 MB	7.6 G	62	2 m 20 s	72.252	79.853	234
Fast R-CNN	130 M	1224 MB	195.01 G	**20**	60 m	70.2	63.800	36
YOLOv2	63.7 M	255.81 MB	128.2 G	150	2 m 40 s	85.19	82.424	56
YOLOv2 tiny	15.04 M	60.16 MB	25.3 G	100	2 m 20 s	78.37	75.417	137
YOLOv4	62.47 M	244.15 MB	219.5 G	36	6 m 18 s	93.47	83.036	10
YOLOv4 tiny	6 M	22.43 MB	24.9 G	59	**57** s	**94.33**	**88.362**	65
YOLOv5 l	88.3 M	529.41 MB	170.46 G	120	2 m 30 s	92.75	87.582	62
YOLOv5 s	7.2 M	13.91 MB	25.78 G	70	1 m 48 s	92.34	87.247	169
YOLOv5 n	1.9 M	3.86 MB	7.03 G	60	1 m 20 s	92.12	87.582	255

Compared with MobileNet, Fast R-CNN, YOLOv2 and YOLOv2 tiny, our solutions outperforms these models in both detection accuracy and efficiency. The last five solutions are based on the two latest YOLO versions. Among them, YOLOv5n has the best detection speed. Our result, H_Arch_C_8, can still achieve 3 times faster results with little loss of F1. To evaluate the detection accuracy a step further, we conducted experiments by setting different thresholds of confidence. In Figure 13, when the threshold of confidence increases, the F1 curves of YOLOv4 tiny and YOLOv5n decay more slowly than H_Arch_C_8 and YOLOv2. That is because YOLOv4 tiny and YOLOv5n have more outputs with greater confidence scores. Hence, fewer targets were lost when the bar of confidence increased.

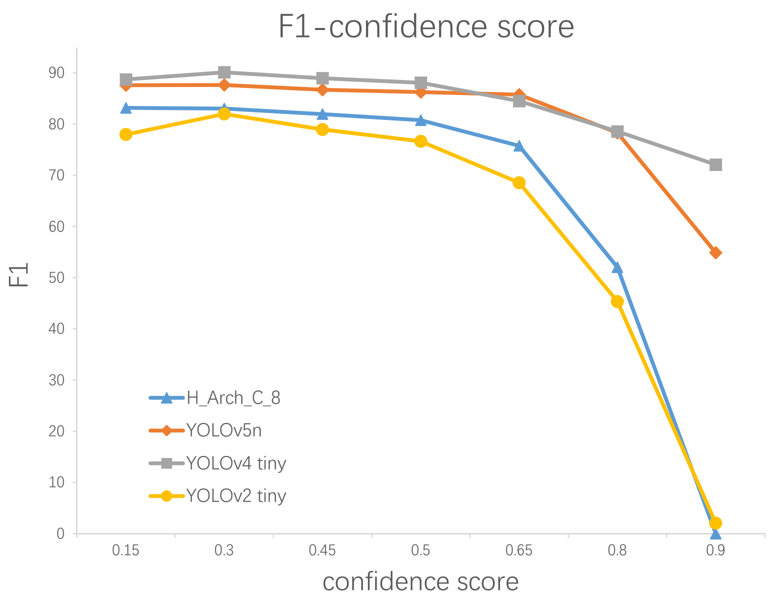


Figure 13. F1-confidence.

We then conducted comparisons with referenced studies [5,6,19,26], as they are lightweight models and provide experimental results on the same SSDD dataset as ours. Note that different GPUs are used in different studies. We only address fair and reasonable comparison results in Table 11 to demonstrate the detection efficiency and accuracy. In addition to S_Arch_C_8 and S_Arch_C_4, all data listed are directly extracted from references. Clearly, Darts_tiny can generate lighter and more efficient neural networks while maintaining a competitive detection accuracy (with only a few points lost in the F1 score).

Table 11. Comparison with references on SSDD.

Architectures	Model Volume	Paras	FLOPs	F1	AP
S_Arch_C_8	150 KB	24 K	0.54 G	90.24	89.61
S_Arch_C_4	**75** KB	**9** K	**0.21** G	88.56	89.14
Reference [19]	49.34 MB	10.78 M	4.26 G	n/a	**95.03**
Reference [5]	2.8 MB	600 K	1.94 G	**91.70**	94.60
Reference [6]	30 MB	6.5 M	5.96 G	n/a	90.37
Reference [26]	1.2 MB	290 K	2.86 G	77.7	79.1

By the above comparisons, some shortages of Darts_tiny receive our attention. First, although the neural networks generated are light and efficient, they do not have the best performance in detection accuracy. Recent YOLO versions win better accuracy, as more techniques and tricks are leveraged, e.g., a "backbone + Neck + detection head" structure, an optimized loss function, etc. Integrating successful artificial experiences into the neural architecture search could be meaningful for future accuracy improvement. Second, fewer computations do not always yield a proportionally faster detection speed, as different hardware platforms e.g., GPUs and embedded processing units, have different preferences in the neural architecture and the operations used. It could be the case that a lighter model fails to win an expected faster detection speed when running on a particular hardware. Improving Darts_tiny to be hardware-aware is quite meaningful.

7. Conclusions

In this paper, we addressed the problem of edge-based ship object detection, where the detection accuracy and efficiency need to be considered together. We proposed Darts_Tiny, a novel customized neural architecture search model. By pruning superfluous operations, we leverage computation-aware schemes into the search process. Thus, Darts_Tiny can generate lightweight neural networks that are not only efficient in terms of the amount of computation but also competitive in terms of detection accuracy. In the future, we shall study more techniques (e.g., aiming at targets with different sizes) to improve the detection accuracy of Darts_Tiny. In addition, we would like to study computational efficiency with different hardware platforms and design hardware-aware schemes into the neural architecture search model.

Author Contributions: Formal analysis, C.L., K.Z. and L.Q.; Funding acquisition, C.L.; Methodology, Y.L., H.H. and J.S.; Project administration, C.L.; Validation, C.L., K.Z. and K.W.; Writing—original draft, C.L.; Writing—review & editing, L.Q. and K.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the National Natural Science Foundation of China under Grant 61902063, by the Provincial Natural Science Foundation of Jiangsu, China under Grant BK20190342, and by the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing by Grant 2020A04.

Institutional Review Board Statement: Not applicable.

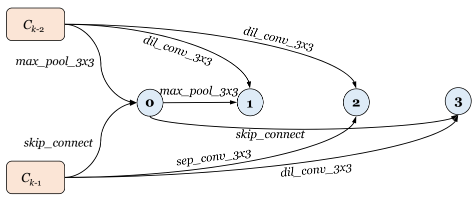
Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

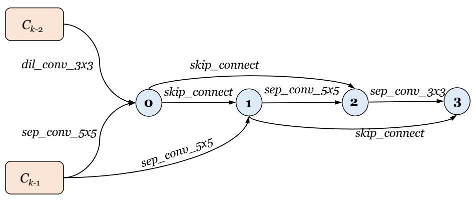
Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Neural Architectures

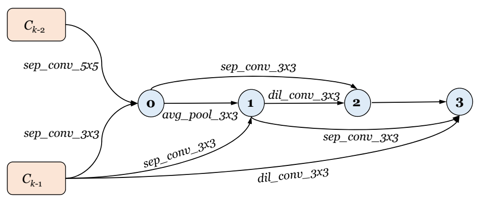
In the last two pages, we provide all neural architectures that appeared in the main text for readers who have special interests (Figures A1–A6).



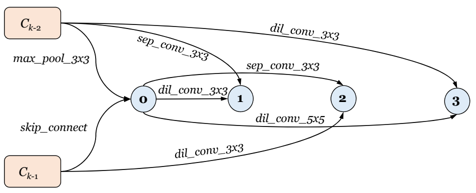
(**a**) HArch_1: Normal Cell



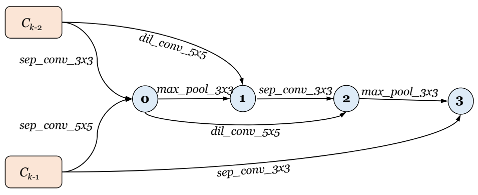
(**b**) HArch_1: Reduction Cell



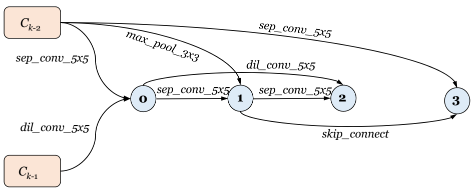
(**c**) HArch_2: Normal Cell



(**d**) HArch_2: Reduction Cell

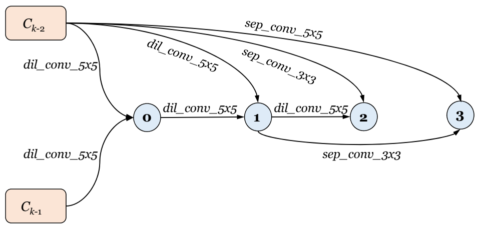


(**e**) HArch_3: Normal Cell

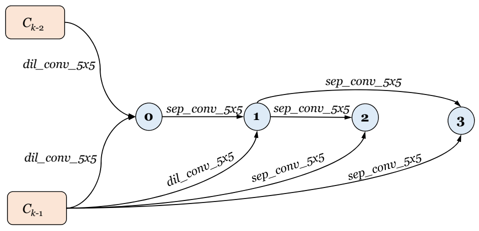


(**f**) HArch_3: Reduction Cell

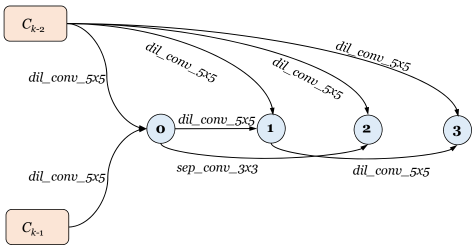
Figure A1. Cell structures of HArch_1, HArch_2 and HArch_3.



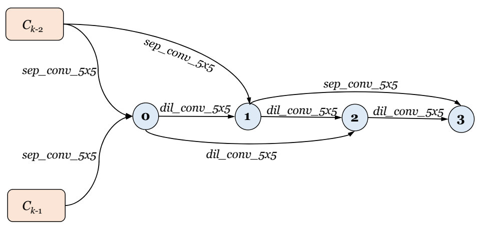
(**a**) SArch_1: Normal Cell



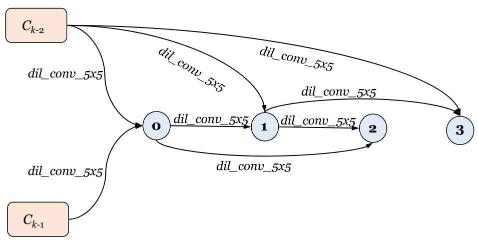
(**b**) SArch_1: Reduction Cell



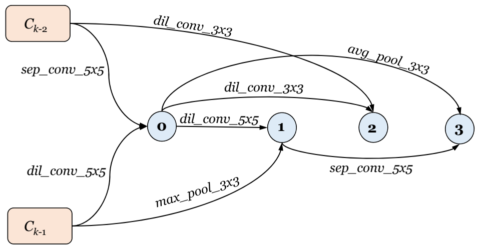
(**c**) SArch_2: Normal Cell



(**d**) SArch_2: Reduction Cell



(**e**) SArch_3: Normal Cell



(**f**) SArch_3: Reduction Cell

Figure A2. Cell structures of SArch_1, SArch_2 and SArch_3.

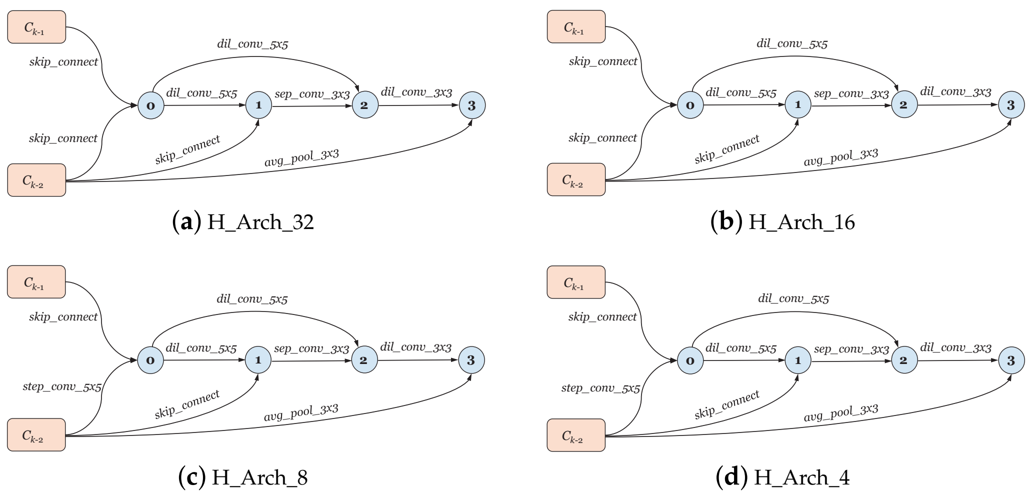


Figure A3. Cell structures obtained over HRSID with $ops\_ratio = 0$.

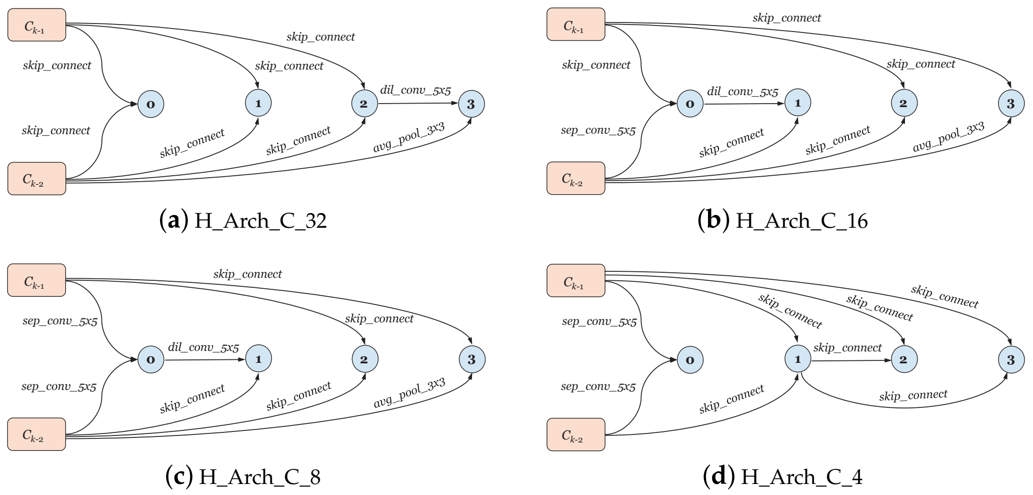


Figure A4. Cell structures obtained over HRSID with $ops\_ratio = 0.012$.

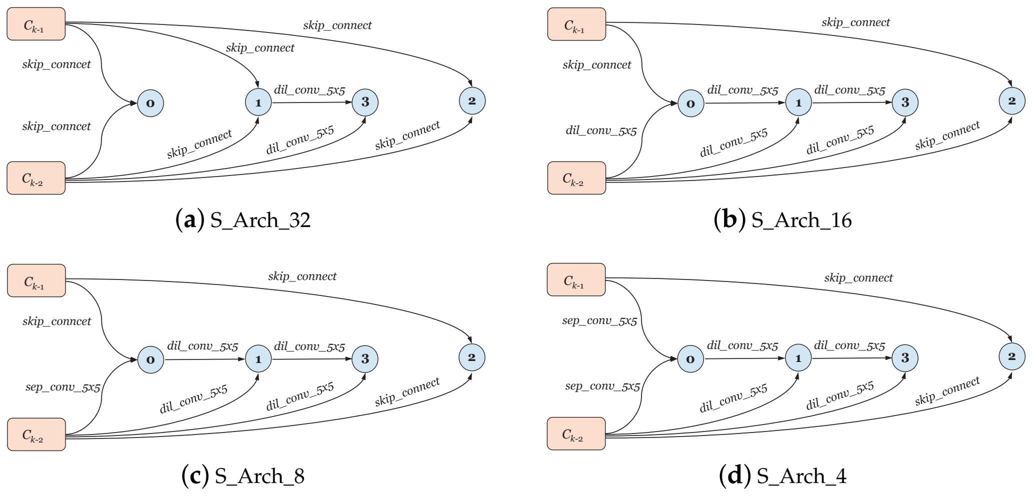
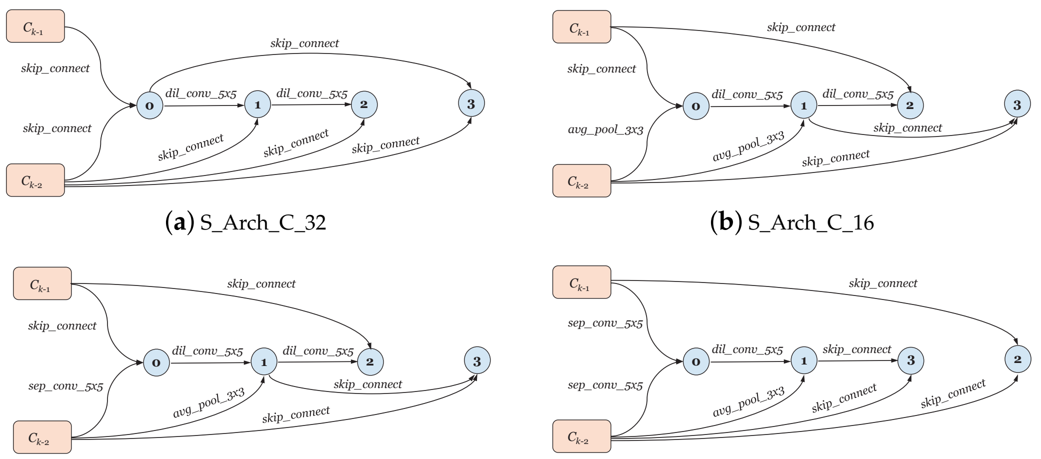


Figure A5. Cell structures obtained over SSDD with $ops\_ratio = 0$.



(**a**) S_Arch_C_32 (**b**) S_Arch_C_16 (**c**) S_Arch_C_8 (**d**) S_Arch_C_4

Figure A6. Cell structures obtained over SSDD with $ops\_ratio = 0.003$.

References

1. Feraru, V.A.; Andersen, R.E.; Boukas, E. Towards an Autonomous UAV-based System to Assist Search and Rescue Operations in Man Overboard Incidents. In Proceedings of the 2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Abu Dhabi, United Arab Emirates, 4–6 November 2020; pp. 57–64.
2. 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 (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
3. Girshick, R. Fast R-CNN. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015; pp. 1440–1448.
4. 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:1704.04861v1.
5. Chen, S.; Zhan, R.; Wang, W.; Zhang, J. Learning Slimming SAR Ship Object Detector Through Network Pruning and Knowledge Distillation. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2021**, *14*, 1267–1282. [CrossRef]
6. Jiang, J.; Fu, X.; Qin, R.; Wang, X.; Ma, Z. High-Speed Lightweight Ship Detection Algorithm Based on YOLO-V4 for Three-Channels RGB SAR Image. *Remote Sens.* **2021**, *13*, 1909. [CrossRef]
7. Liu, H.; Simonyan, K.; Yang, Y. DARTS: Differentiable Architecture Search. In Proceedings of the 7th International Conference on Learning Representations (ICLR), New Orleans, LA, USA, 6–9 May 2019; pp. 1–13.
8. Xu, Y.; Xie, L.; Zhang, X.; Chen, X.; Qi, G.J.; Tian, Q.; Xiong, H. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In Proceedings of the 8th International Conference on Learning Representations (ICLR), Addis Ababa, Ethiopia, 26–30 April 2020; pp. 1–13.
9. Wei, S.; Zeng, X.; Qu, Q.; Wang, M.; Su, H.; Shi, J. HRSID: A High-Resolution SAR Images Dataset for Ship Detection and Instance Segmentation. *IEEE Access* **2020**, *8*, 120234–120254. [CrossRef]
10. Li, J.; Qu, C.; Shao, J. Ship detection in SAR images based on an improved faster R-CNN. In Proceedings of the SAR in Big Data Era: Models, Methods and Applications (BIGSARDATA), Beijing, China, 13–14 November 2017; pp. 1–6.
11. Li, J.; Qu, C.; Peng, S.; Deng, B. Ship detection in SAR images based on convolutional neural network. *Syst. Eng. Electron.* **2018**, *40*, 1953–1959.
12. Li, J.; Qu, C.; Peng, S.; Jiang, Y. Ship Detection in SAR images Based on Generative Adversarial Network and Online Hard Examples Mining. *J. Electron. Inf. Technol.* **2019**, *41*, 143–149.
13. Guo, H.; Yang, X.; Wang, N.; Gao, X. A CenterNet++ model for ship detection in SAR images. *Pattern Recognit.* **2021**, *112*, 107787. [CrossRef]
14. Zhang, Z.; Zhang, L.; Wang, Y.; Feng, P.; He, R. ShipRSImageNet: A Large-Scale Fine-Grained Dataset for Ship Detection in High-Resolution Optical Remote Sensing Images. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2021**, *14*, 8458–8472. [CrossRef]
15. Geng, X.; Zhao, L.; Shi, L.; Yang, J.; Li, P.; Sun, W. Small-Sized Ship Detection Nearshore Based on Lightweight Active Learning Model with a Small Number of Labeled Data for SAR Imagery. *Remote Sens.* **2021**, *13*, 3400. [CrossRef]
16. Sun, Z.; Dai, M.; Leng, X.; Lei, Y.; Xiong, B.; Ji, K.; Kuang, G. An Anchor-Free Detection Method for Ship Targets in High-Resolution SAR Images. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2021**, *14*, 7799–7816. [CrossRef]
17. Li, Y.; Zhang, S.; Wang, W. A Lightweight Faster R-CNN for Ship Detection in SAR Images. *IEEE Geosci. Remote Sens. Lett.* **2022**, *19*, 4006105. [CrossRef]
18. Zhang, T.; Zhang, X. High-Speed Ship Detection in SAR Images Based on a Grid Convolutional Neural Network. *Remote Sens.* **2019**, *11*, 1206. [CrossRef]

19. Liu, S.; Kong, W.; Chen, X.; Xu, M.; Yasir, M.; Zhao, L.; Li, J. Multi-Scale Ship Detection Algorithm Based on a Lightweight Neural Network for Spaceborne SAR Images. *Remote Sens.* **2022**, *14*, 1149. [CrossRef]
20. Feng, Y.; Chen, J.; Huang, Z.; Wan, H.; Xia, R.; Wu, B.; Sun, L.; Xing, M. A Lightweight Position-Enhanced Anchor-Free Algorithm for SAR Ship Detection. *Remote Sens.* **2022**, *14*, 1908. [CrossRef]
21. Alkhaleefah, M.; Ma, S.; Tan, T.; Chang, L.; Wang, K.; Ko, C.; Ku, C.; Hsu, C.; Chang, Y. Accelerated-YOLOv3 for Ship Detection from SAR Images. In Proceedings of the IEEE International Geoscience and Remote Sensing Symposium, IGARSS, Brussels, Belgium, 11–16 July 2021; pp. 3030–3032.
22. Xu, X.; Zhang, X.; Zhang, T. Lite-YOLOv5: A Lightweight Deep Learning Detector for On-Board Ship Detection in Large-Scene Sentinel-1 SAR Images. *Remote Sens.* **2022**, *14*, 1018. [CrossRef]
23. Sun, Z.; Leng, X.; Lei, Y.; Xiong, B.; Ji, K.; Kuang, G. BiFA-YOLO: A Novel YOLO-Based Method for Arbitrary-Oriented Ship Detection in High-Resolution SAR Images. *Remote Sens.* **2021**, *13*, 4209. [CrossRef]
24. Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. *arXiv* **2018**, arXiv:1804.02767.
25. Bochkovskiy, A.; Wang, C.; Liao, H.M. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv* **2020**, arXiv:2004.10934.
26. Ma, X.; Ji, K.; Xiong, B.; Zhang, L.; Feng, S.; Kuang, G. Light-YOLOv4: An Edge-Device Oriented Target Detection Method for Remote Sensing Images. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2021**, *14*, 10808–10820. [CrossRef]
27. Ge, Z.; Liu, S.; Wang, F.; Li, Z.; Sun, J. YOLOX: Exceeding YOLO Series in 2021. *arXiv* **2021**, arXiv:2107.08430.
28. Zoph, B.; Le, Q.V. Neural architecture search with reinforcement learning. In Proceedings of the 5th International Conference on Learning Representations (ICLR), Toulon, France, 24–26 April 2017; pp. 1–16.
29. 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 (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; pp. 8697–8710.
30. Liang, H.; Zhang, S.; Sun, J.; He, X.; Huang, W.; Zhuang, K.; Li, Z. DARTS+: Improved Differentiable Architecture Search with Early Stopping. *arXiv* **2020**, arXiv:1909.06035.
31. Chu, X.; Wang, X.; Zhang, B.; Lu, S.; Wei, X.; Yan, J. DARTS-: Robustly Stepping out of Performance Collapse Without Indicators. In Proceedings of the 9th International Conference on Learning Representations (ICLR), Virtual, 3–7 May 2021; pp. 1–22.
32. Wang, Y.; Shi, H.; Chen, L. Ship Detection Algorithm for SAR Images Based on Lightweight Convolutional Network. *J. Indian Soc. Remote Sens.* **2022**, *50*, 867–876. [CrossRef]
33. Wu, B.; Dai, X.; Zhang, P.; Wang, Y.; Sun, F.; Wu, Y.; Tian, Y.; Vajda, P.; Jia, Y.; Keutzer, K. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR Long Beach, CA, USA, 16–20 June 2019; pp. 10734–10742.
34. Kong, X.; Han, W.; Liao, L.; Li, B. An analysis of correctness for API recommendation: Are the unmatched results useless? *Sci. China Inf. Sci.* **2020**. *63*, 190103. [CrossRef]
35. Redmon, J.; Farhadi, A. YOLO9000: Better, Faster, Stronger. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, Honolulu, HI, USA, 21–26 July 2017; pp. 6517–6525.
36. YOLOv2 Tiny and YOLOv4 Tiny. Available online: https://github.com/AlexeyAB/darknet (accessed on 2 February 2022).
37. YOLOv5. Available online: https://github.com/ultralytics/yolov5 (accessed on 2 February 2022).