machines

MDPI

Article

ULO: An Underwater Light-Weight Object Detector for Edge Computing †

Lin Wang [1], Xiufen Ye [1,*], Shunli Wang [1] and Peng Li [2]

[1] College of Intelligent Systems Science and Engineering, Harbin Engineering University, Harbin 150001, China; wanglin0222@hrbeu.edu.cn (L.W.); wangshunli@hrbeu.edu.cn (S.W.)
[2] Management School, Harbin Commerce University, Harbin 150080, China; lipeng@hrbcu.edu.cn
* Correspondence: yexiufen@hrbeu.edu.cn
† This paper is an extended version of our paper published in *YOLO Nano Underwater*: A fast and compact object detector for embedded device.

Abstract: Recent studies on underwater object detection have progressed with the development of deep-learning methods. Generally, the model performance increase is accompanied by an increase in computation. However, a significant fraction of remotely operated underwater vehicles (ROVs) and autonomous underwater vehicles (AUVs) operate in environments with limited power and computation resources, making large models inapplicable. In this paper, we propose a fast and compact object detector—namely, the Underwater Light-weight Object detector (ULO)—for several marine products, such as scallops, starfish, echinus, and holothurians. ULO achieves comparable results to YOLO-v3 with less than 7% of its computation. ULO is modified based on the YOLO Nano architecture, and some modern architectures are used to optimize it, such as the Ghost module and decoupled head design in detection. We propose an adaptive pre-processing module for the image degradation problem that is common in underwater images. The module is lightweight and simple to use, and ablation experiments verify its effectiveness. Moreover, ULO Tiny, a lite version of ULO, is proposed to achieve further computation reduction. Furthermore, we optimize the annotations of the URPC2019 dataset, and the modified annotations are more accurate in localization and classification. The refined annotations are available to the public for research use.

Keywords: object detection; edge computing; adaptive pre-processing; underwater

Citation: Wang, L.; Ye, X.; Wang, S.; Li, P. ULO: An Underwater Light-Weight Object Detector for Edge Computing. *Machines* **2022**, *10*, 629. https://doi.org/10.3390/machines10080629

Academic Editor: Antonios Gasteratos

Received: 5 July 2022
Accepted: 27 July 2022
Published: 29 July 2022

1. Introduction

The demand for aquatic products is growing. The poor operating environment of manual underwater fishing, the high risk to personnel safety, and the increased workforce cost have limited the industry's scale. As artificial intelligence has been applied to an increasing number of remotely operated underwater vehicles (ROVs) and autonomous underwater vehicles (AUVs), there has been a trend to use robots to catch marine products autonomously or semi-autonomously.

Many object detection and recognition algorithms have been directly transferred to marine research and industrial use, promoting the development of these fields. However, a large proportion of ROVs or AUVs work under power limitations (for instance, through battery power), and their endurance is closely related to the operating power of the onboard devices. Therefore, more efficient and lightweight algorithms become an immediate need for these devices with severely limited computational resources.

Since R-CNN [1] made convolutional neural networks popular in object detection, many state-of-the-art object detection architectures have been proposed. They can be roughly divided into two-stage object detection methods and single-stage methods. Two-stage methods, such as R-CNN, obtain the final results based on the region proposals selected as the first step. Though the subsequent Fast R-CNN [2] and Faster R-CNN [3] reduced the inference time, they still cannot meet the needs of the edge computing devices.

One-stage methods, such as SSD [4] and YOLO [5], were proposed for a better practical use as they search for the objects in the whole picture without the step for selecting proposals, which have better performance on the inference time. Recently, researchers have also used machine-learning methods to assist in searching efficient network architectures [6–9]. These auto-designed architectures can have a good balance between accuracy and inference time.

In this paper, we propose an underwater light-weight object detector—namely, ULO—for aquatic products, i.e., scallops, starfish, echinus, and holothurians, and this runs on edge computing devices, such as the NVIDIA Jetson Nano, with competitive performance. Moreover, we provide a lite version of ULO, ULO Tiny, which reduces the computation further.

Furthermore, an adaptive pre-processing module is proposed as a plug-in block, which benefits the detection performance. Furthermore, we optimized the annotations of the URPC2019 dataset and provided more accurate locations and labels of the targets. Code and refined labels will be available. (Code: https://github.com/wangsssky/YOLO_Underwater, Refined annotations: https://github.com/wangsssky/Refined-training-set-of-URPC2019/, accessed on 26 July 2022).

The preliminary work [10] was presented at the conference *IEEE OCEANS 2020*. In this journal extension, we updated our original work with concrete improvements in both structural designs and more comprehensive experiments.

- First, the novel network structures, ULO and ULO Tiny, are proposed, which further reduce the requirements of computation resources.
- Secondly, the adaptive pre-processing module is introduced as a light-weight and easy image enhancement method.
- Thirdly, more comprehensive experiments are conducted to evaluate the proposed networks fully.

The rest of the paper is organized as follows: Section 2 introduces the URPC2019 dataset for underwater object detection and our optimizations to the annotations. Section 3 provides the details of the proposed networks and modules, such as *YOLO Nano Underwater*, ULO, and ULO Tiny. In Section 4, we evaluate the proposed networks on the URPC2019 dataset and conduct an ablation study on the adaptive pre-processing module. Section 5 concludes this paper.

2. Dataset

A customized version of the dataset of the underwater robot picking contest 2019 (The competition website: http://www.urpc.org.cn/index.html, accesed on 29 July 2022) (URPC2019) is used in this work. The dataset consists of 4757 underwater images with corresponding annotations. Four types of objects are included, i.e., scallops, starfish, echinus, and holothurians, which are commonly farmed products in the ocean.

Underwater optical images are often blurred due to the attenuation of light in the underwater environment [11]. As a result, underwater targets are also more difficult to label and are prone to category errors and deviations in the boundaries of bounding boxes. Therefore, to evaluate the algorithm results more reasonably, the annotations of the dataset are optimized.

Compared to the original annotations, the refined bounding boxes are more accurate with fewer classification mistakes. Figure 1 shows some examples of the URPC2019 dataset and the comparisons between the refined and the original bounding boxes. To encourage the research of underwater object detection, we release the modified annotations to the public for academic use.

For the following experiments, the images are randomly divided into the training, validation, and testing sets. The ratio of the number of training/validation and testing images are roughly set at 5:1 by convention. Specifically, there are 3600, 400, and 757 images in training, validation, and testing sets, respectively.

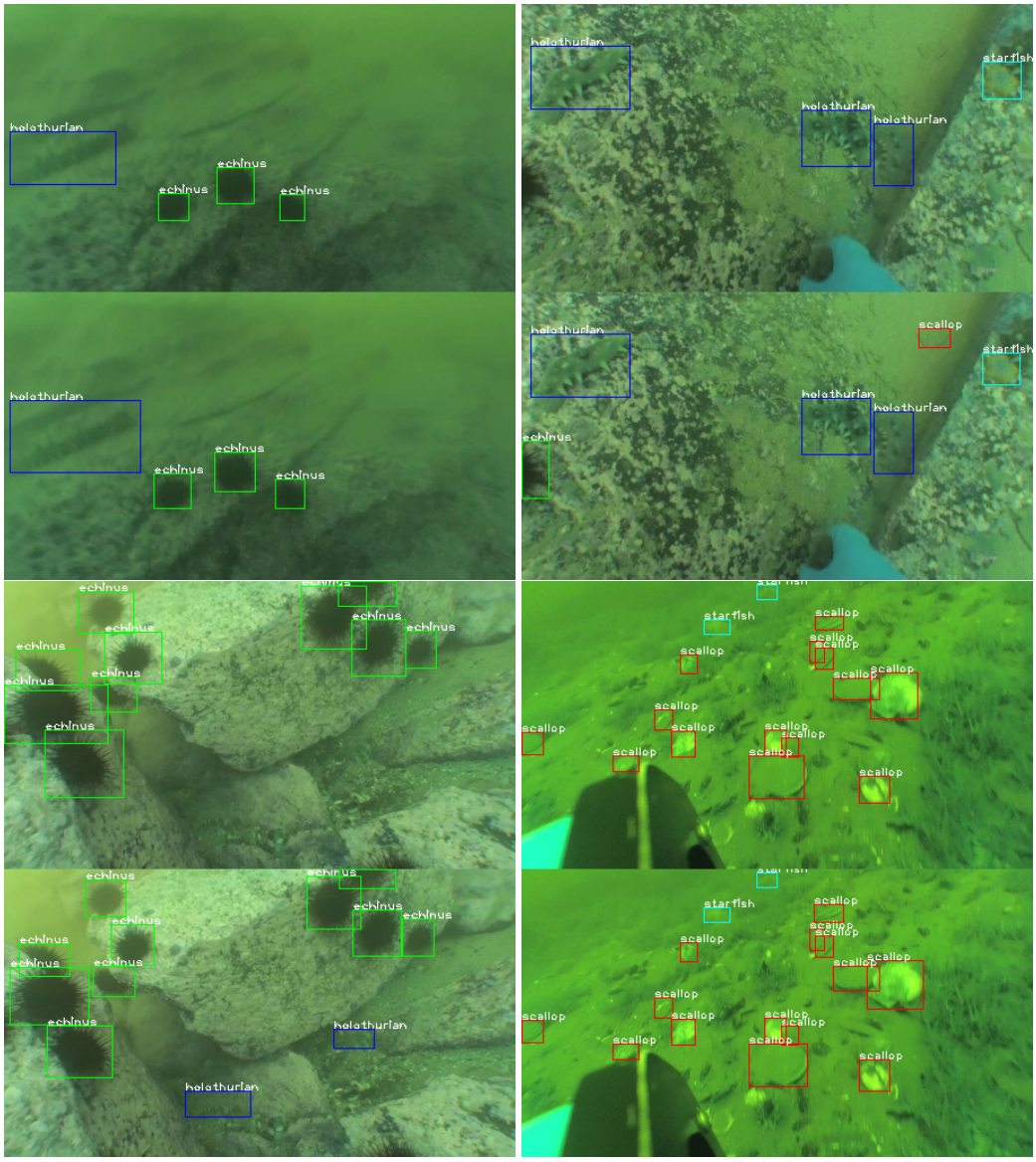


Figure 1. Examplesof the underwater images in the URPC2019 dataset. We show the original annotations (above) and the optimized annotations (below) corresponding to each image. The refined bounding boxes contain fewer annotations errors of missing, mislabeling, and inaccurate positioning caused by blurred images. Better viewed in zoom and color.

3. Methods

YOLO- [5,10,12–16] like object detectors are widely used both in academic scenarios and industries for its excellent balance of performance and efficiency, which has been proven to be a classic and pivotal approach of one-stage detection networks in practical applications. In this paper, we propose a compact YOLO-like network structure for computation resources limited devices, which applies to autonomous marine vehicles. Moreover, we propose an adaptive image enhancement module for the case of severe degradation of underwater image quality. In the section, the originally proposed network structure, *YOLO Nano Underwater*, is first introduced, and then the novel improvements made in this extension are presented.

3.1. YOLO Nano Underwater

YOLO Nano [6] is designed to be a highly compact neural network based on the YOLO-v3 network by the collaboration of machine-driven and human-specified design.

The YOLO Nono network achieves a dramatic reduction in the number of parameters and the amount of computation compared to YOLO-v2 and YOLO-v3 [6].

YOLO Nano Underwater is proposed based on the carefully designed YOLO Nano network with two improvements. First, we remove several layers to further reduce the number of parameters and the floating-point operations (FLOPs). Moreover, setting the number of channels to an integer multiple of 8 helps to fully utilize the processing units of the hardwares [17]. Therefore, we align the filter number of the convolutional layers to 8×.

The basic macro structures of *YOLO Nano Underwater* are kept the same as YOLO Nano, and the network is stacked by the modules of the residual Projection Expansion Projection (PEP), Expansion Projection (EP), and Fully Connected Attention (FCA). The EP module is a depth-wise separable comvolution [18,19] with expanded channels. The PEP module adds a convolutional layer with kernel size 1×1 before the EP module, which projects the input into a lower dimension and reduces the computation [6]. The FCA module introduces the attention mechanism into the architecture, which helps the network pay greater attention to informative features. Figure 2 shows the architecture of the proposed *YOLO Nano Underwater*.

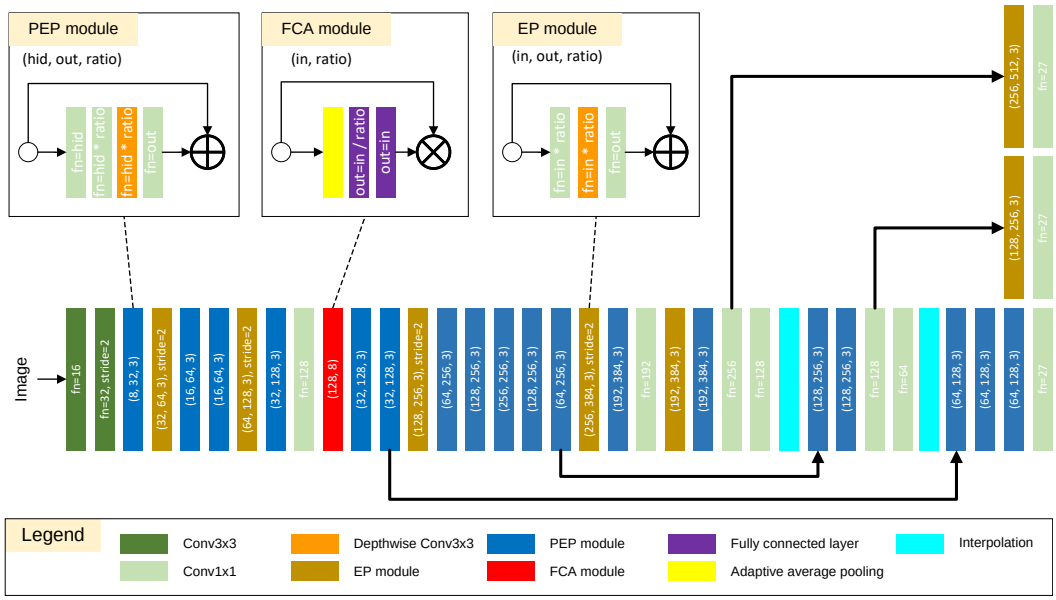


Figure 2. Theschematic diagram of the *YOLO Nano Underwater* architecture. EP stands for the Expansion Projection macro architecture, and PEP stands for the residual Projection Expansion Projection structure. FCA is a lightweight Fully Connected Attention module. The specific parameters of each layer are shown in the blocks. Batch normalizations and activations are omitted for simplicity. * stands for multiply. Better viewed in zoom and color.

Table 1 shows the computation cost evaluated by FLOPs and network parameters comparison with some state-of-the-art YOLO architectures. The FLOP and number of parameters of our proposed *YOLO Nano Underwater* are only 88.51% and 70.99% of YOLO Nano.

3.2. ULO and ULO Tiny

In the last two years, many new efficient architectures and network designs have been proposed. We take these as the new gradients to modify our *YOLO Nano underwater* network. Moreover, to address the problem of underwater optical image degradation, we design an adaptive pre-processing module that can automatically learn pre-processing parameters based on the task scenario. Two models for underwater light-weight object detectors are proposed at different scales—namely, ULO and ULO Tiny, respectively.

Table 1. The FLOPs and the number of parameters of different models. The input size of the network is set to $1 \times 3 \times 512 \times 512$.

Model	FLOPs (G)	Parameters
YOLO-v3	49.47	61,539,889
YOLO-v3 Tiny	4.13	8,676,806
YOLO-v4	45.10	63,953,841
YOLO-v4 Tiny	5.60	6,074,582
YOLO Nano	6.57	6,396,048
YOLO Nano Underwater	4.93	4,540,792
ULO	3.42	3,943,632
ULO (w/o APM)	3.42	3,930,320
ULO Tiny	2.38	3,416,016
ULO Tiny (w/o APM)	2.38	3,402,704

3.2.1. Ghost Modules

GhostNet [20,21] is an efficient network architecture, which was proposed based on the observation that the feature maps' redundancy is important for the network's success. Therefore, the authors used cheap operations to generate pseudo feature maps, significantly reducing the computation. GhostNet has been successfully used in many resource-limited applications, such as remote-sensing and bed-side medical scenarios [22–24].

The Ghost module is the basic block of the GhostNet. It separates the conventional convolution layer into two steps: First, the input passes through the normal convolution layer with fewer filters. Then, more feature maps are created based on the output of the first step by cheap linear operations. The Ghost bottleneck is built by stacking two Ghost modules, such as the residual block [25]. Moreover, as the attention mechanism is proven to be an effective module in practice [26], the squeeze-and-excitation block [27] (the FCA module in YOLO Nano) also becomes an optional component in the Ghost bottleneck module.

In addition, the cheap operations used in the convolution layer, we also apply G-GhostNet mechanism [21] to the *YOLO Nano Underwater* network that the cheap operations are used to produce pseudo feature maps across layers to spare the computation resources further. Figure 3 shows the structure of the Ghost module and Ghost bottleneck.

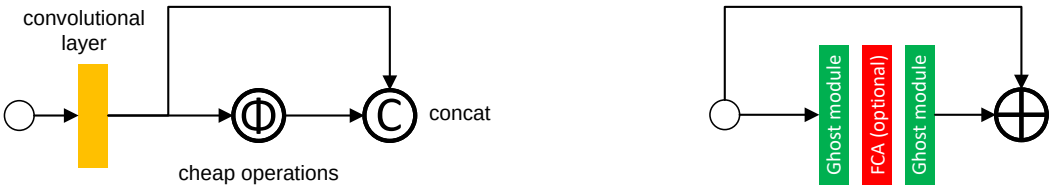


Figure 3. The schematic diagram of the Ghost module (**a**) and Ghost bottleneck (**b**). The Ghost module partially uses cheap operations to generate pseudo feature maps, which is more computation efficient than traditional convolution. The Ghost bottleneck is a residual unit [25] built up by two ghost modules and an optional squeeze-and-excitation block. Batch normalizations and activations are omitted for simplicity.

3.2.2. Decoupled Head

The coupled head for classification and localization may harm the performance [16,28]. Furthermore, experiments show that the network with decoupled head converges faster than the network with coupled head [16]. Therefore, we use a decoupled head in our ULO and ULO Tiny network design. The schematic diagram of the decoupled head is shown in Figure 4.

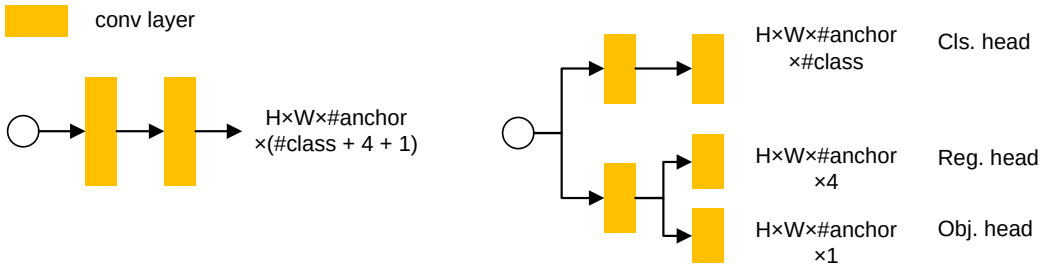


Figure 4. The schematic diagram for the decoupled head in detection. The decoupled head deals with the classification and localization by separated network structures. (**a**) Coupled head. (**b**) Decoupled head.

3.2.3. Adaptive Pre-Processing Module

The underwater images suffer from problems, such as non-uniform lighting, low contrast, and diminished color [29]. Moreover, the degradation of underwater images fails in computer systems used for visual inspection of images [30]. Some methods have been proposed to handle this with deep neural networks [31,32] and achieved good results. However, heavy computation may not be applicable for edge computing. Therefore, we propose the adaptive pre-processing module (APM), a simple but effective pre-processing module. APM automatically adapts to the task. Moreover, it is light-weight and easy to use.

Digital gain, white balance, color correction, and gamma correction are widely used manipulations in image enhancement [33]. They are defined as follows:

$$\mathcal{G}_{Digital\ Gain}(x, \phi) = \phi_{dg} \cdot x, \tag{1}$$

$$\mathcal{G}_{White\ Balance}\left(\begin{bmatrix} x_r \\ x_g \\ x_b \end{bmatrix}, \phi\right) = \begin{bmatrix} \phi_r \cdot x_r \\ x_g \\ \phi_b \cdot x_b \end{bmatrix}, \tag{2}$$

$$\mathcal{G}_{Color\ Correction}\left(\begin{bmatrix} x_r \\ x_g \\ x_b \end{bmatrix}, \phi\right) = \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix} \begin{bmatrix} x_r \\ x_g \\ x_b \end{bmatrix} + \begin{bmatrix} \phi_{b1} \\ \phi_{b2} \\ \phi_{b3} \end{bmatrix}, \tag{3}$$

$$\mathcal{G}_{Gamma\ Correction}(x, \phi) = x^{\phi_\gamma}, \tag{4}$$

where x is the input RGB color image, ϕ is the hyperparameters for adjustments. For simplicity, we merge such manipulations in one step and set the individual gamma correction parameters for each channel, as the light attenuation rates are different for different colors. Therefore, the overall image enhancement is defined as:

$$\mathcal{G}\left(\begin{bmatrix} x_r \\ x_g \\ x_b \end{bmatrix}, \phi\right) = \left(\begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix} \begin{bmatrix} x_r \\ x_g \\ x_b \end{bmatrix} + \begin{bmatrix} \phi_{b1} \\ \phi_{b2} \\ \phi_{b3} \end{bmatrix}\right)^{\left[\phi_{\gamma_r}, \phi_{\gamma_g}, \phi_{\gamma_b}\right]^T}. \tag{5}$$

There are 15 hyperparameters in total for a three-channel input image. Methodologically, this method applies to input images with an arbitrary number of channels. For example, there will be three parameters for a single-channel grayscale image and 24 for a four-channel CMYK image. We design a small module for learning the hyperparameters automatically. The adaptively learned parameters may better fit the specific tasks than manually designed pre-processing methods.

The APM module consists of an adaptive average pooling layer and two fully-connected layers. The pooling layer dramatically reduces the dimension of the input image and extracts the primary information about the color in the image. The fully-connected layers enable the fusion of information between different color channels and output the 15 parameters for the image enhancement processing. Finally, the sigmoid activation en-

sures that the output values are in the interval $(-1, 1)$, which facilitates stability during training. Figure 5 shows the structure of the APM.

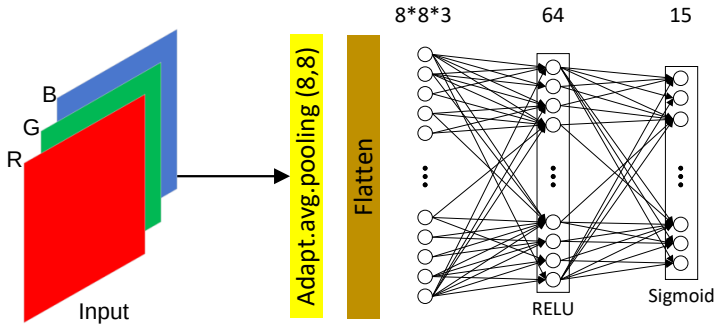


Figure 5. The schematic diagram of the adaptive pre-processing module. The APM module inputs the RGB image and outputs the hyperparameters adapted to the specific task for the image enhancement processing.

3.2.4. Architectures of ULO and ULO Tiny

We keep the framework of the newly proposed ULO the same as *YOLO Nano Underwater* and replace the EP and PEP modules with Ghost modules or Ghost bottleneck modules. Moreover, the decoupled head and APM are used. The structure of ULO and ULO Tiny are shown in Figure 6.

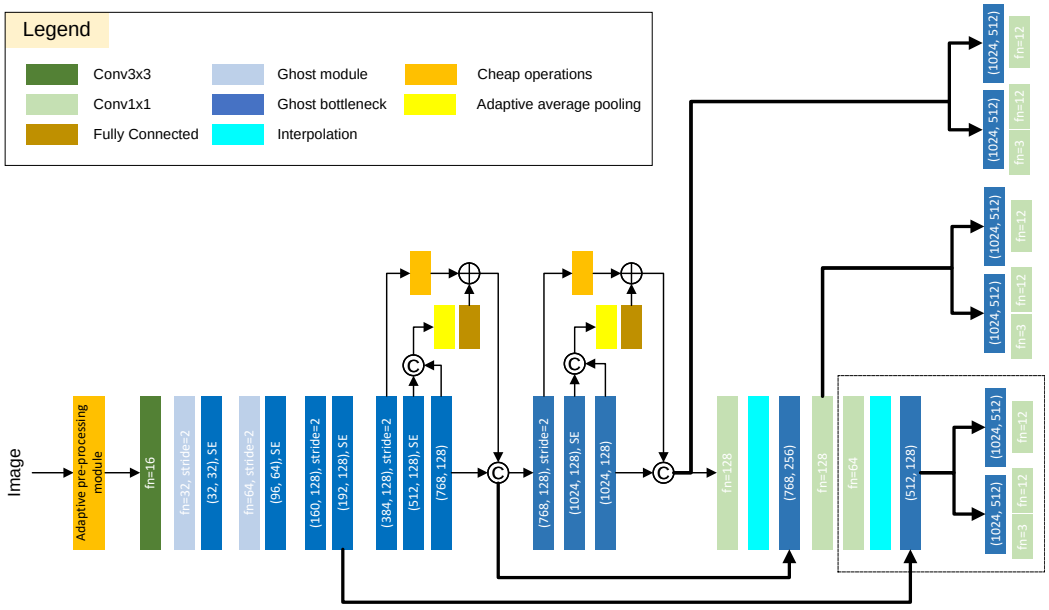


Figure 6. The schematic diagram of the ULO and ULO Tiny architectures. In the newly proposed networks, Ghost modules, decoupled head, and adaptive pre-processing module are used. The difference in structure between ULO Tiny and ULO is that ULO Tiny does not contain the structure in the dotted box to reduce the network parameters further. Batch normalizations and activations are omitted for simplicity. Better viewed in zoom and color.

The computation cost of ULO and ULO Tiny are shown in Table 1. The FLOPs and the number of parameters in ULO are 3.42 G and 3.94 M, only 59.37% and 86.85% of that in *YOLO Nano Underwater*. Moreover, ULO Tiny takes 2.38 G FLOPs and 3.40 M parameters, further reducing the need for computation resources.

4. Experiments and Results

In this section, we first conduct experiments to evaluate our models' performance in the underwater detection scenario, and then we investigate the role of the proposed adaptive pre-processing module by ablation study.

4.1. Implementations Details

Experiments were conducted on the URPC2019 dataset to compare our model with the state-of-the-art YOLO models, i.e., YOLO-v4, YOLO-v4 Tiny, YOLO-v3, YOLO-v3 Tiny, and YOLO Nano. For a fair comparison, we implemented the compared networks using PyTorch and kept the same training parameters, random seed, and image pre-processing methods.

Augmentations were deployed during the training phase, such as cropping, randomly flipping, affine transformation, and color jitter. The input image size is 512×512. The ADAM is the optimizer with weight decay 5×10^{-4}. The cosine annealing schedule [15,34] was applied, and the initial learning rate is 1×10^{-3}. All the models were trained with 300 epochs and batch-size 64 from scratch. The confidence and NMS thresholds were set to 0.25 and 0.45, respectively.

4.2. Main Results

The model is trained on the training set and evaluated on the validation set for model selection. The final results were tested using the testing set. Table 2 lists the results evaluated by average precision on both the testing and validation sets. Figure 7 illustrates the FLOPs-mAP curve and the Parameters-mAP curve, respectively.

Table 2. Performance of the object detectors evaluated by average precision. The top three results are marked in **red**, **green**, and **blue**, respectively.

Model	Average Precision @Testing-Set					@Validation-Set
	Mean	Holothurian	Echinus	Scallop	Starfish	Mean
YOLO-v3	**64.91%**	**47.45%**	**83.36%**	**57.96%**	**70.89%**	**65.4396%**
YOLO-v3 Tiny	36.15%	8.11%	66.27%	29.20%	41.03%	38.7254%
YOLO-v4	52.27%	**26.63%**	71.72%	**47.57%**	**63.15%**	53.2539%
YOLO-v4 Tiny	49.36%	23.94%	70.05%	41.77%	61.67%	50.1678%
YOLO Nano	44.00%	13.45%	73.88%	44.44%	44.21%	44.2392%
YOLO Nano Underwater	46.37%	13.04%	75.43%	45.89%	51.11%	47.5504%
ULO	**63.53%**	**40.35%**	**84.25%**	**58.73%**	**70.79%**	**65.1283%**
ULO Tiny	**53.48%**	25.49%	**79.21%**	46.99%	62.22%	**56.9383%**

Overall, YOLO-v3 achieved the best performance on the URPC2019 dataset, following 1 ULO and ULO Tiny. However, the ULO and ULO Tiny models were much smaller than YOLO-v3. Moreover, the difference between ULO and YOLO-v3 in mAP was only 1.38%. Therefore, the ULO and ULO Tiny can be more applicable to the devices in which computation resources are limited.

Comparing architectures with similar network designs, the larger models often achieved better performance. For example, YOLO-v3 outperformed YOLO-v3 Tiny, and YOLO-v4 outperformed YOLO-v4 Tiny. This is also true for ULO and ULO Tiny, as the larger models with more detection heads can provide more anchors. On the other hand, small models also significantly reduce the computational complexity, lower hardware requirements, and broaden usage scenarios.

However, the model performance of YOLO-v4 in URPC2019 is not better than YOLO-v3, somewhat beyond our expectation, likely because more complex models tend to require richer data for training, and the URPC2019 dataset is still not as rich in content and sample scale as the object detection datasets for natural scenes, such as VOC [35] and COCO [36].

As can be seen from the performance-computational resource curves presented in Figure 7, our proposed ULO and ULO Tiny models have significantly smaller resource requirements and competitive performance simultaneously.

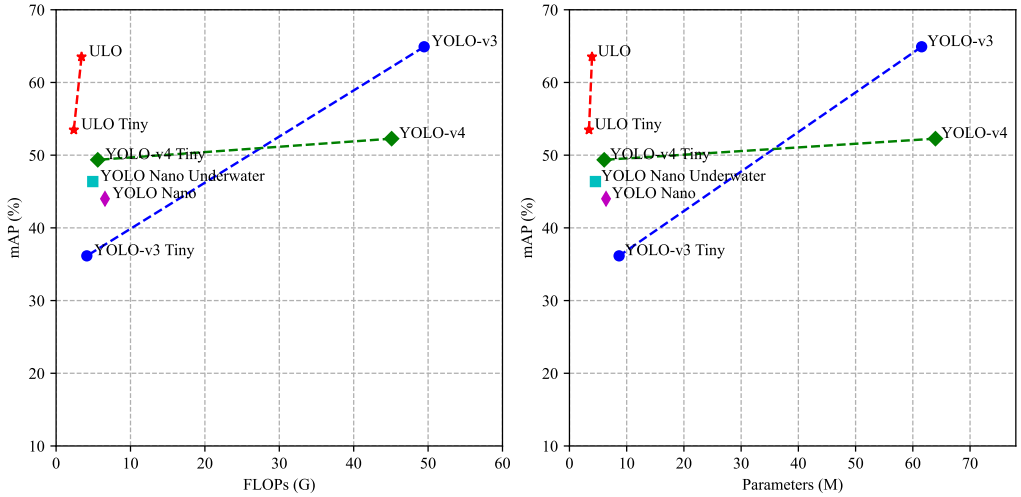
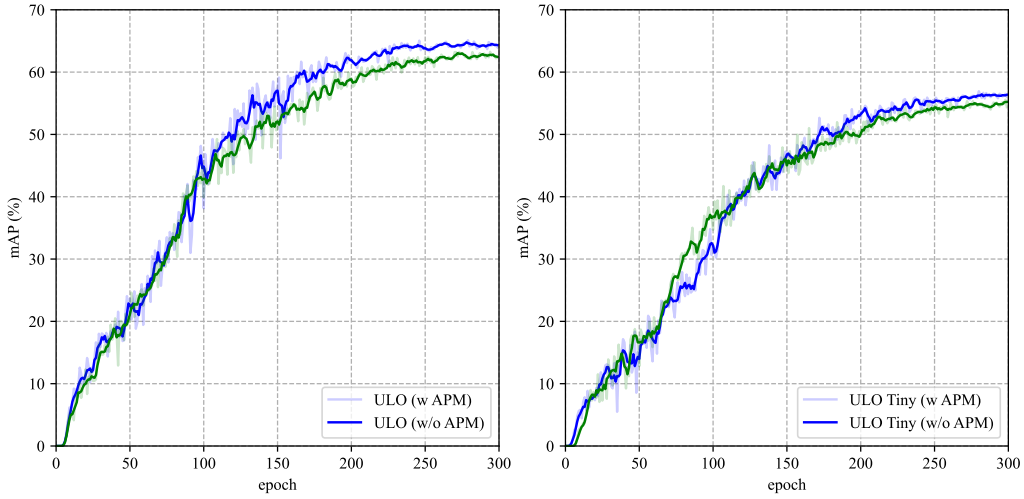


Figure 7. The FLOPs-mAP curve (**left**) and the Parameters-mAP curve (**right**).

4.3. Ablation Study on APM

In the ablation experiment, we investigated the effects of APM on performance enhancement. Experiments were deployed to observe whether there is a difference in performance between adding APM when training the ULO and ULO Tiny. The training settings were kept the same, except for whether the APM was used or not. Figure 8 shows the mAPs evaluated on the validation set of each epoch during training. Table 3 illustrates the performance tested on the testing set of URPC2019.



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

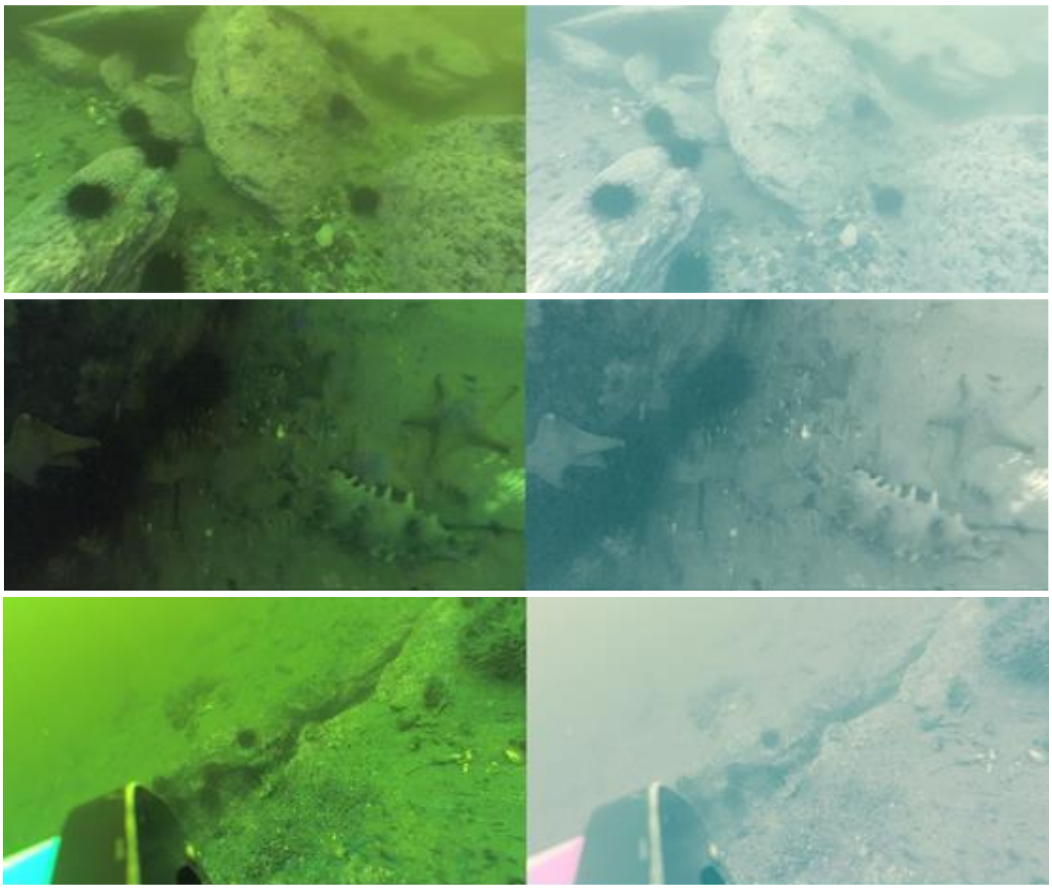
Figure 8. The mean average precision values tested on the validation set during training. The models with APM have a more significant performance improvement in the middle and final stages of training than those without. (**a**) ULO. (**b**) ULO Tiny.

Table 3. Ablation study of the adaptive pre-processing module. Experiments were conducted on ULO and ULO Tiny networks with or without APM, and the results were evaluated by the mAP. The results demonstrate that APM is beneficial in improving performance in both ULO and ULO Tiny. The best results are in bold.

Model	Average Precision @Testing-Set					@Validation-Set
	Mean	Holothurian	Echinus	Scallop	Starfish	Mean
ULO (w/o APM)	59.78%	32.34%	83.45%	58.20%	65.15%	63.2606%
ULO (w APM)	**63.53%**	**40.35%**	**84.25%**	**58.73%**	**70.79%**	**65.1283%**
ULO Tiny (w/o APM)	53.18%	24.08%	78.81%	**47.57%**	**62.26%**	55.5682%
ULO Tiny (w APM)	**53.48%**	**25.49%**	**79.21%**	46.99%	62.22%	**56.9383%**

As can be seen in Figure 8, the performance improvement of the models with APM was slightly faster at the beginning of the training, and in the middle and final stages of training, the performance improvement of the model with APM is more evident than that of the model without APM. Table 3 also numerically demonstrates the improvement of APM for model performance, indicating that APM is beneficial to the model performance.

We show some examples of the images processed by the APM module in Figure 9. It can be seen that the processed image corrects the color bias of the underwater image to some extent compared to the original image, and this is all achieved in the task-based learning by the model itself.



(**a**) Input images (**b**) Pre-processed images by APM

Figure 9. Compared to the original images (**a**), the pre-processed images output by APM (**b**) correct the underwater color bias to an extent.

Experiments on Edge Device

We deployed our models to Nvidia Jetson Nano, a 10-watt, $120 edge computing device. The details of the experiments' results are shown in Table 4. It achieves an inference speed of 5.11 and 6.73 FPS with the PyTorch framework for ULO and ULO Tiny, respectively. In the semi-precision mode, the performance of the model remains roughly unchanged, with some insignificant improvements in inference time, which may be related to the hardware architecture.

We also observe a 30% improvement in inference time with an input size of 416 but a 10-point decrease in performance, which may be because we focus more on the impact of the model design and do not use multi-scaled inputs in training. In practical applications, adopting more training techniques, such as diverse input scales, mixup/mosaic augmentation, and using pre-training of the models on large datasets are expected to improve the performance of the models further. Moreover, there is still much room to improve the inference time, such as accelerating by TensorRT [37] or TVM [38].

Table 4. The average processing time, FPS, and mean precision of each model evaluated on Nvidia Jetson Nano.

Settings	ULO			ULO Tiny		
	avg. Time (s)	FPS	mAP (%)	avg. Time (s)	FPS	mAP (%)
PyTorch fp32 @512	0.1955	5.11	63.45	0.1485	6.73	53.47
PyTorch fp16 @512	0.1931	5.17	62.69	0.1470	6.80	52.23
PyTorch fp16 @416	0.1370	7.30	53.06	0.1029	9.71	41.84
ONNX [39] fp32 @512	0.3640	2.75	63.45	0.2274	4.40	53.47

5. Conclusions

Underwater intelligent systems suffer from limited power consumption and computational resources. In response, this paper proposed a series of light-weight models, including ULO, ULO Tiny, and *YOLO Nano Underwater*, which can work on edge computing devices, such as Nvidia Jetson Nano, with satisfying performance.

Some of the latest network designs, such as Ghost modules, decoupled head, and the attention mechanism, were used. With such ingredients, the proposed well-designed models are compact and efficient. These models are significantly reduced in terms of FLOPs, and the model parameters compared to YOLO-v3/YOLO-v4. The performance was preserved, while the computational resource consumption decreased. ULO achieved 97.9% of YOLO-v3's performance using less than 7% of the FLOPs, which allows these models to better adapt to low-power, computational resource-constrained scenarios. Moreover, we deployed an adaptive pre-processing module for automatic image enhancement and proved it can improve the task performance in underwater object detection.

At the same time, underwater images have been relatively less studied compared with natural scenes, and fewer datasets are available. Therefore, high-quality underwater detection datasets are essential for the development of related research. For this reason, we optimized some mis-annotated labels in the URPC2019 dataset, hoping to provide a more reliable reference for evaluating the underwater models. The modified annotations have been released to the public for research use.

Author Contributions: Conceptualization, software, and methodology, L.W.; data curation, L.W. and S.W.; writing—original draft preparation, L.W.; writing—review and editing, X.Y., S.W. and P.L.; visualization, P.L.; supervision, project administration, and funding acquisition, X.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (Grant No. 41876100), the National key research and development program of China (Grant No. 2018YFC0310102 and 2017YFC0306001), and the State Key Program of National Natural Science Foundation of China (Grant No. 61633004).

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.

References

1. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA, 23–28 June 2014; pp. 580–587.
2. Girshick, R. Fast R-CNN. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 8–10 June 2015; pp. 1440–1448.
3. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. *Adv. Neural Inf. Process. Syst. (NeurIPS)* **2015**, *28*. [CrossRef] [PubMed]
4. 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 European Conference on Computer Vision (ECCV), Amsterdam, The Netherlands, 8–16 October 2016; pp. 21–37.
5. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
6. Wong, A.; Famuori, M.; Shafiee, M.J.; Li, F.; Chwyl, B.; Chung, J. YOLO Nano: A highly compact you only look once convolutional neural network for object detection. In Proceedings of the Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NeurIPS), Vancouver, BC, Canada, 13 December 2019; pp. 22–25.
7. Tan, M.; Le, Q. EfficientNet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International Conference on Machine Learning (ICML), PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 6105–6114.
8. Tan, M.; Le, Q. Efficientnetv2: Smaller models and faster training. In Proceedings of the International Conference on Machine Learning (ICML), PMLR, Virtual, 18–24 July 2021; pp. 10096–10106.
9. Tan, M.; Pang, R.; Le, Q.V. EfficientDet: Scalable and efficient object detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 14–19 June 2020; pp. 10781–10790.
10. Wang, L.; Ye, X.; Xing, H.; Wang, Z.; Li, P. *YOLO Nano Underwater*: A fast and compact object detector for embedded device. In Proceedings of the Global Oceans 2020: Singapore–US Gulf Coast, Biloxi, MS, USA, 5–30 October 2020; pp. 1–4.
11. Akkaynak, D.; Treibitz, T. Sea-thru: A method for removing water from underwater images. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019; pp. 1682–1691.
12. Redmon, J.; Farhadi, A. YOLO9000: Better, faster, stronger. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 7263–7271.
13. Redmon, J.; Farhadi, A. YOLOv3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
14. Huang, R.; Pedoeem, J.; Chen, C. YOLO-LITE: A real-time object detection algorithm optimized for non-GPU computers. In Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 10–13 December 2018; pp. 2503–2510.
15. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. Yolov4: Optimal speed and accuracy of object detection. *arXiv* **2020**, arXiv:2004.10934.
16. Ge, Z.; Liu, S.; Wang, F.; Li, Z.; Sun, J. YOLOX: Exceeding yolo series in 2021. *arXiv* **2021**, arXiv:2107.08430.
17. Yu, J.; Huang, T.S. Universally slimmable networks and improved training techniques. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea, 27 October–2 November 2019; pp. 1803–1811.
18. 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.
19. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520.
20. 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 (CVPR), Seattle, WA, USA, 13–19 June 2020; pp. 1580–1589.
21. Han, K.; Wang, Y.; Xu, C.; Guo, J.; Xu, C.; Wu, E.; Tian, Q. GhostNets on Heterogeneous Devices via Cheap Operations. *Int. J. Comput. Vis.* **2022**, *130*, 1050–1069. [CrossRef]
22. Paoletti, M.E.; Haut, J.M.; Pereira, N.S.; Plaza, J.; Plaza, A. Ghostnet for hyperspectral image classification. *IEEE Trans. Geosci. Remote. Sens.* **2021**, *59*, 10378–10393. [CrossRef]
23. Cai, Z.; Xie, Q. Attention based GC-GhostNet for forest pests detection. In Proceedings of the 2021 IEEE fourth International Conference on Electronics and Communication Engineering (ICECE), Xi'an, China, 17–19 December 2021; pp. 113–117.
24. Li, P.; Wang, L.; Luo, Y. Ghost-UNet: An Efficient Method for Wound Surface Segmentation. *Basic Clin. Pharmacol. Toxicol.* **2020**, *127*, 288.
25. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.

26. Guo, M.H.; Xu, T.X.; Liu, J.J.; Liu, Z.N.; Jiang, P.T.; Mu, T.J.; Zhang, S.H.; Martin, R.R.; Cheng, M.M.; Hu, S.M. Attention mechanisms in computer vision: A survey. *Comput. Vis. Media* **2022**, *8*, 331–368. [CrossRef]
27. Hu, J.; Shen, L.; Sun, G. Squeeze-and-excitation networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *8*, 7132–7141.
28. Wu, Y.; Chen, Y.; Yuan, L.; Liu, Z.; Wang, L.; Li, H.; Fu, Y. Rethinking classification and localization for object detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020; pp. 10186–10195.
29. Bazeille, S.; Quidu, I.; Jaulin, L.; Malkasse, J.P. Automatic underwater image pre-processing. In Proceedings of the CMM'06, Brest, France, 16–19 October 2006.
30. Raveendran, S.; Patil, M.D.; Birajdar, G.K. Underwater image enhancement: A comprehensive review, recent trends, challenges and applications. *Artif. Intell. Rev.* **2021**, *54*, 5413–5467. [CrossRef]
31. Yu, X.; Qu, Y.; Hong, M. Underwater-GAN: Underwater image restoration via conditional generative adversarial network. In Proceedings of the International Conference on Pattern Recognition (ICPR), Beijing, China, 20–24 August 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 66–75.
32. Yang, M.; Hu, K.; Du, Y.; Wei, Z.; Sheng, Z.; Hu, J. Underwater image enhancement based on conditional generative adversarial network. *Signal Process. Image Commun.* **2020**, *81*, 115723. [CrossRef]
33. Kim, H.; Lee, K.M. Controllable Image Enhancement. *arXiv* **2022**, arXiv:2206.08488.
34. Loshchilov, I.; Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv* **2016**, arXiv:1608.03983.
35. Everingham, M.; Eslami, S.; Van Gool, L.; Williams, C.K.; Winn, J.; Zisserman, A. The pascal visual object classes challenge: A retrospective. *Int. J. Comput. Vis.* **2015**, *111*, 98–136. [CrossRef]
36. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft COCO: Common objects in context. In Proceedings of the European Conference on Computer Vision (ECCV), Zurich, Switzerland, 6–12 September 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 740–755.
37. Vanholder, H. Efficient inference with tensorrt. In Proceedings of the GPU Technology Conference, San Jose McEnery Convention Center, Silicon Valley, CA, USA, 4–7 April 2016; Volume 1, p. 2.
38. Chen, T.; Moreau, T.; Jiang, Z.; Zheng, L.; Yan, E.; Shen, H.; Cowan, M.; Wang, L.; Hu, Y.; Ceze, L.; et al. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), Carlsbad, CA, USA, 8–10 October 2018; pp. 578–594.
39. Bai, J.; Lu, F.; Zhang, K. Onnx: Open Neural Network Exchange. Available online: https://github.com/onnx/onnx (accessed on 26 July 2022).