PyTorch faster R-CNN for Marine Life Object Detection

Daniel Jordan
danielharley.jordan@studenti.unipd.it

Jake Jackson
jake.jackson@studenti.unipd.it

Abstract

Marine life object detection is an exciting young field with enormous potential to help in conservation efforts. In this study, a PyTorch faster R-CNN is experimented with on the benchmark brackish dataset [8]. Some interesting preprocessing methods are tested that have potential that is not yet realised in this study. Canny edge detection is tested both in isolation and overlaid on top of the original image. Red channel isolation is also experimented with in hopes to minimise the blue effect that water induces through light attenuation. The best trained faster R-CNN in this study was able to infer with a mean average precision of 0.7391. This was achieved with a tuned Adam optimizer on images that were simply resized. The model can make inferences with a speed of 8.08 frames per second.

1. Introduction

Underwater object detection is a relatively new field of computer vision. Large organized datasets are not currently available and few papers have been published on the topic. This study uses the underwater brackish dataset [8] with just over 14 500 images and 6 categories. This pales in comparison to the Microsoft COCO above water dataset of 330 000 images and 80 categories [6]. Effective implementation remains an important step for marine applications such as robot navigation and sea life monitoring. The ocean is a vast unexplored part of nature that is vital to our existence. It has absorbed roughly 30% of human CO_2 emissions from the industrial revolution to the mid 1990s [3]. This carbon capture relies on the countless species of plants and animals that exist in a delicate balance of resources. By deploying computer vision technology species populations can be estimated. This can inform government bodies of decisions on overfishing policies and installations of marine structures such as oil rigs or tidal power. This study investigates the potential of the PyTorch Faster R-CNN model to identify marine animals in images. The model will output a bounding box for each animal detected and the predicted label. The performance will be monitored in terms of the frames per second inferred (fps) and the mean average precision (mAP).

2. Related Work

One of the main challenges with underwater computer vision is the attenuation of light as it traverses the water. Each light frequency losses intensity at a different rate. This is why objects underwater often appear bluer when at greater distances. This can confuse the algorithm. The same object will appear in different colours depending on the distance from the camera. A paper published in 2019 showcased an algorithm for correcting the colour information based on a known distance to the target [1].

There have been two papers published that also use the brackish dataset. One from the Shanghai Ocean University in 2021 [12]. The model used was YOLO v4. They report an impressive mAP of 0.9265 and a fps of 44.22. A more novel approach was deployed by Vijiyakumar Krishnan in 2022. A hybrid of two separate models is used, RetinaNet and EfficientNet. They named it HDCNN-UODT. The hybrid model achieved a mAP of 0.9615 and an incredible fps of 310.25 [4].

3. Dataset

This Brackish dataset contains 14,674 images. Only 12,444 images contain animals. To save on computation resources only the images that contain animals were used. Each image has a corresponding XML file with bounding box and label information. There are 6 labels and their frequency of occurrence is shown in figure 4. There is a great imbalance that creates a model bias toward the labels with more data. The images were captured by Aalborg University. The camera was mounted at 9 meters depth on the Limfjords channel in northern Denmark. This naturally results in noisy low-contrast images. Deep Neural Networks (DNNs) can surpass human observations on this type of image [2]. Since many recordings in biological settings, match this criterion. This investigation aims to help illustrate the

Figure 1. Example image of the Brackish dataset [8], showing available bounding box information (targets) and predictions.

potential advantages of using innovative techniques like the faster R-CNN in these frequent settings.

Figure 2. Frequency of each label that appears in the Brackish dataset [8].

This is a benchmark dataset and as such, it is pre-split into training, validation, and test sets with an 8:1:1 ratio. Each image has an original size of 960x540 pixels, resized to 400x400. The images are scaled down to allow sizeable batches to fit onto the GPU's RAM during training. They are made square as we believed this was a requirement of the model for any fully connected layers. Later we discovered that as this was once the case there is an adjustment that can allow any image dimensions to be fed into a CNN with fully connected final layers. This is explained in an article by Grigory Serebryakov [11].

Various prepossessing techniques are trialed in an attempt to improve the mAP. Edges are well known in the computer vision community to hold a significant portion of the information required for classification. Thus a canny edge detection transform is trialed. The clearest threshold is chosen by eye over a few images. With more resources, the threshold would be included within a hyperparameter tuning. To maximize valuable information retention, another trial was completed where the edges are overlaid on top of the original image. Underwater, light attenuation is dependent on frequency. This is why in deep underwater images objects further away appear bluer. For this reason, it is speculated that the red part of the image contains more

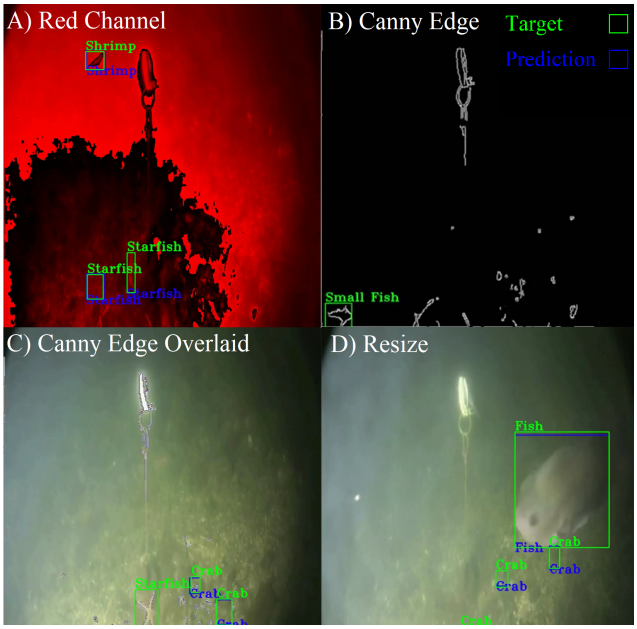

Figure 3. Image showing trailed prepossessing techniques.

accurate information about the object being observed. The idea is that it should not be as altered by light attenuation compared with blue. As a trial, the model is fed only the red channel of each image.

4. Method

For this investigation, the chosen model is the faster R-CNN. It is a popular object detection algorithm that to the best of our knowledge has not yet been applied to underwater benchmark datasets. During the inference of an image with an R-CNN the image goes through multiple steps that link together for the final result. First, it is passed through a region proposal network which aims to output possible bounding boxes and corresponding objectness scores. This score is the relative degree of certainty that there is an object within the box. To generate the region proposals many convolutional feature filters are applied to the image and a small network is slid over the feature maps. A feature that has been learned will be brighter in the feature map and be more likely to contain an object. The region proposal network contains within also a bounding box regressor. This aims to refine the bounding box coordinates to enclose the object tightly. It also contains a foreground/background classifier which decides the objectness score. The image within these boxes is then placed into a CNN designed for object classification to predict the label of the object inside the box.

In previous versions of the algorithm, each part needed to be trained separately and the results were joined together. In the latest version (Faster R-CNN) they are all part of a single architecture that can be back-propagated during train-

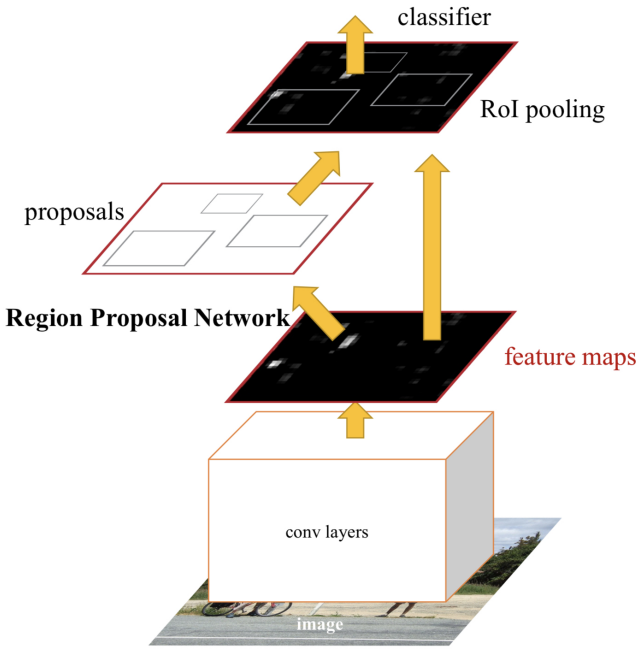

Figure 4. Diagram taken from the original faster R-CNN paper [10], summarising the region proposal network.

ing. This was key to gaining speed in both training and inference. The model is provided with a backbone CNN with an architecture that is known to perform well. This is used for both the region proposal and object classification with as many learned feature filters being shared between the two. Similar algorithms are YOLO and retina net. They have many similarities in that they all deploy some sort of region proposal and are composed of a single architecture with no need to train multiple models separately. They all also use a backbone CNN, ResNet50 is a popular choice and used in this study. The differences are in the details. For example, YOLO divides the image into a grid, each grid cell is assigned the label that it has the highest probability of containing. Each grid cell can only have one label and so all bounding boxes with this cell as its center can only be classified as that label. This is a limitation. If two different objects are in one cell there will be an error. Although it speeds up computation. The RetinaNet deploys what they call a focal loss which aims to counteract the label imbalance between foreground and background. Most proposed regions will not contain an object which creates an extreme label imbalance giving the algorithm a bias toward detecting the background over objects. Papers deploying versions of YOLO and RetinaNet were mentioned in the related work section and their performances will be compared in this study.

Training every weight of the ResNet50 with the small brackish dataset would not yield good results. It would also not be time feasible on a single low-class GPU. The

Optimizer	Preprocessing	mAP	fps
Adam	Resize	0.6685	8.10
Tuned Adam	Resize	0.7391	8.07
SGD	Resize	0.7288	8.06
Adam	Red Channel	0.4112	8.12
Adam	Canny Edge	0.0000	8.38
Adam	Canny Edge Overlaid	0.4032	8.11

Table 1. Results for all of the final trained models. Resize to 400x400 pixels was applied for all models.

Model	Preprocessing	mAP	fps
YOLO-V4 [12]	Unknown	0.9265	44.22
HDCNN-UODT [4]	”colour correction”	0.9615	310.25

Table 2. Two models available from the literature that were also trained on the Brackish Dataset.

ResNet50 was previously trained on the large MS COCO dataset [6]. Most of the weights resulting from this training are used in this implementation. A few of the weights are retrained on the brackish dataset. The MS COCO dataset contains only above-water images. There are no underwater datasets sizeable enough to be a replacement. Many underwater computer vision experiments rely on weights trained on above water images. This limits their performance potential.

5. Experiments

There are 6 final models under investigation with various optimizers and preprocessing techniques applied. These are summarized in the results table 2. The main quantities to asses performance are mean average precision (mAP) and frames per second (fps). The fps is simply computed as the number of images being inferred, divided by the time taken to carry out the inference. There are various standards set for computing the mAP. This study uses the MS COCO mAP which is apart of the torchmetrics library. In short the average precision (AP) is the area under the precision recall curve. Various standards use different approximations of this area to save on computation time. The area is also different depending on the IoU threshold chosen. Different standards deploy different sets of IoU thresholds. The mAP for a single class is the mean of the AP for each IoU threshold. The mAP for all the classes is the mean of the mAP for each class. The full mAP calculation is clearly explained in an article by Kukil [5]. The IoU threshold is the intersection over union of the true bounding box and a predicted box of matching class. Intersection is the area of overlap, union is the total area of the two boxes minus the intersection. If this is over the threshold then the inferred box is said to be a true positive match. The mAP is also dependent on the score threshold. The score is the degree of certainty an in-

ferred box contains an object. It was found that a 0 score threshold always led to a higher mAP. Yet from inspecting the inferred images the performance at 0 threshold is very low. Many boxes are placed over regions where there is truly no object, this would lead to a low mAP. Yet the mAP is higher because additionally many boxes are placed over a single object, often with a correct label. The metric mAP does not punish identifying an object many times. For this reason to identify the ideal score threshold we defined a custom metric. The score threshold is proportional to the number of inferred boxes. The modulated difference between the true number of boxes and inferred number of boxes is calculated. This was calculated over the validation images for 50 score thresholds between 0 and 1, see figure 5. The minimum difference reveals the best score threshold. In this way the best score threshold was found for each of the final models.

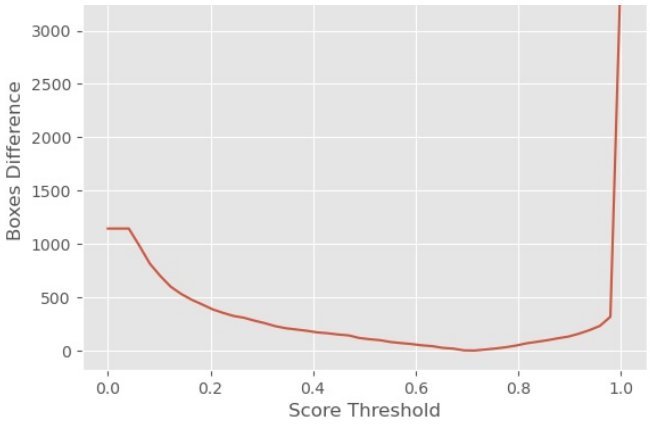

Figure 5. Illustrating the effect of score threshold on the models ability to find the correct number of boxes. 'Boxes Difference' is the difference between labeled boxes and inferred boxes in the validation dataset. The SGD optimizer and resize only preprocessing is used.

The small resources available limits the hyperparameter tuning. One epoch takes approximately 50 min. Convergence begins to occur after approximately 10 epochs. This is with the default ResNet50 architecture and default Adam optimizer. The number of tuned hyperparameters is kept at a minimum. The ResNet50 architecture is kept at default. A hyperparameter search is conducted via Optuna over Adam's various parameters. Optuna by default uses a Bayesian approach to choosing the parameters it trials. More specifically it uses the tree structured parzen algorithm [7]. To overcome the small resources available a novel approach is tested. The tune is composed of 3 steps. In the first step Optuna is set to vary the learning rate with the goal of lowering the number of epochs required to achieve an acceptable validation loss. This is set as 0.2 which corresponds to a mAP of 0.5. This was chosen as in this study default Adam achieves an mAP of 0.65 after 10 epochs, and the faster R-CNN achieved 0.48 on the MS COCO dataset [6]. An acceptable loss is achieved in one epoch with a tuned learning rate. The next tune is then limited to 1 epoch per trial. In this second tune, all of Adams parameters are varied to minimise the validation loss. This allows Optuna to explore the parameter space. It can then generate the relative parameter importance, as shown in figure 6. The last step is a third tune where only the two "most important" parameters are selected, learning rate and beta1. The others are set to the best previously found values. This is run for another 20 trials. Each trial ended early if it underperformed. This was defined as having a training loss greater than 5 after 10% of the epoch was complete.

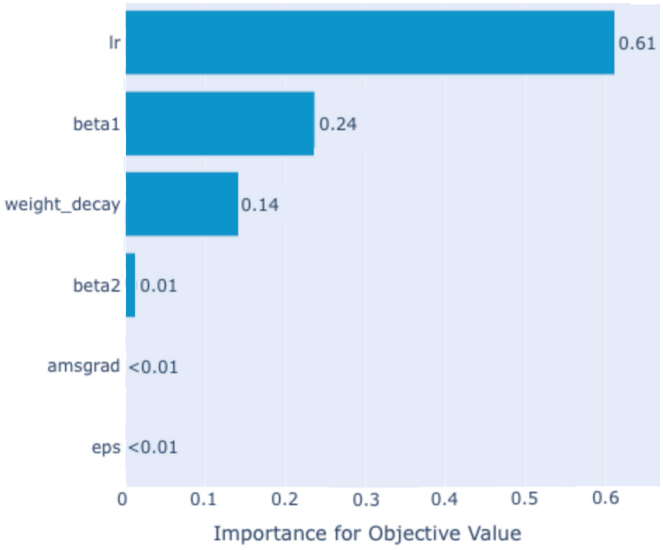

Figure 6. Chart generated by Optuna after 20 trials over all Adam parameters. Showing the relative hyperparameter importances. In the order shown they are, learning rate, beta1, weight decay, beta2, AMSGrad, epsilon. The objective value Optuna used is the validation loss.

The three optimizers tested are Adam default, Adam with tuned hyperparameters and SGD with momentum. The SGD parameters are set to a learning rate of 0.001, momentum 0.9, and a weight decay of 0.0005. This optimizer was tested as it was used in another PyTorch faster R-CNN implementation [9].

Figure 7 and table 2 show that the tuned Adam optimizer achieved the best results in terms of mAP. The hyperparameter tuning allowed for a small improvement on the Adam optimizer. Even with only one epoch being used for each trial of the tuning. SGD is very close and would likely perform better if it was similarly tuned.

Figure 8 shows that some classes were easier to learn than others. This is partially influenced by the label imbalance, see figure 4. The crab had the most occurrences and was the second easiest to recognize. The nature of the animal also affects the model's ability to classify. Starfish have

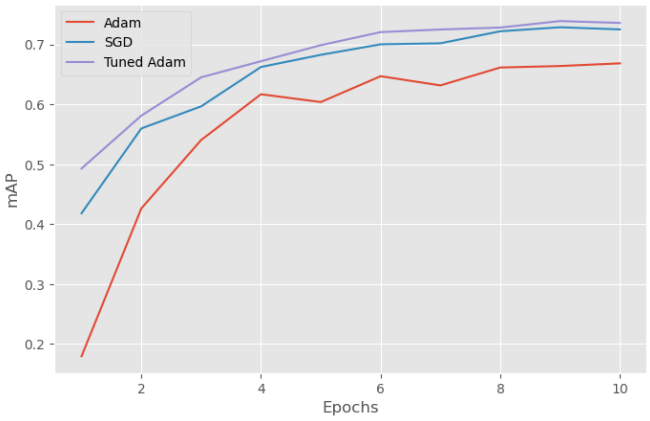

Figure 7. The mAP is calculated for each optimizer at each of 10 epochs. Only resize to 400x400 prepossessing is implemented.

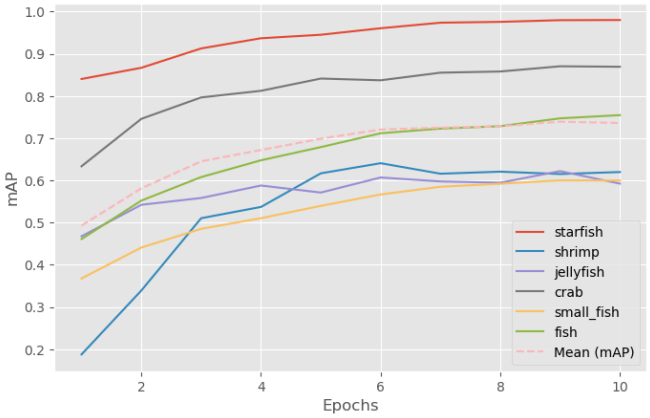

Figure 8. The mAP is separated into the various classes for the most accurate final model; Tuned Adam optimizer with resize transform.

a very distinct unchanging shape and are mostly stationary making them the easiest to classify even though there is more data on crabs. The small fish are fast and succumb to motion blur. They also often occlude one another making them more difficult to classify regardless of the large amount of data. The model has significantly less data on jellyfish and shrimp which both change shape and move fast. It is impressive the model can identify them with similar precision as small fish.

Figure 9 compares the performances of models with preprocessing applied. Each uses the default Adam optimizer. Some improvement was seen for the first epoch with 'Red Channel'. After the third, there is a significant performance drop. With limited resources available no further epochs are run. The 'Canny Edge' preprocessing is especially disappointing with an mAP of 0. It is likely that tuning the canny thresholds will improve performance.

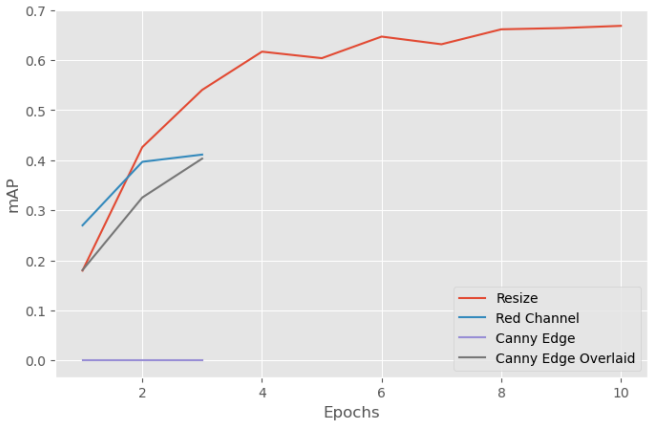

Figure 9. Pre-processing model performance compared to only resizing. The default Adam optimizer is used. Only 3 epochs were allowed as there were no performance gains.

6. Conclusion

Underwater object detection is an interesting use case for deep neural networks, in particular the faster R-CNN. It has exciting implications for marine wildlife conservation efforts. The Brackish dataset is a small benchmark dataset for the comparison of models designed for underwater object detection. It has bounding box information for 12,444 images and 6 labels. Prepossessing techniques are tested for performance improvement 3. Our tests show no significant improvement. It is likely the canny edge detection could improve performance if its thresholds are tuned. A novel approach to hyperparameter tuning is implemented. Each trial is limited to a single epoch. This was discovered in a learning rate tuning to be the minimum required to reach an acceptable validation loss of 0.2. Which corresponds to a mAP of approximately 0.5. The tuned optimizer achieved a mAP of 0.7391, this is a small improvement to the default Adam, 0.6685 mAP. SGD with momentum was very close with a value of 0.7288 mAP. None of these values come close to 0.9265 which was achieved with YOLO-v4 by Minghua Zhang [12]. It is also shy of the value achieved by Vijiyakumar Krishnan using a hybridization of RetinaNet and EfficientNet (HDCNN-UODT), 0.9615 mAP [4]. The frames per second are also better on the other models, YOLO-V4 44.22 fps and HDCNN-UODT 301.25 fps. The faster R-CNN achieved an average fps of 8.08 in this study. The faster R-CNN has not proven itself as a strong contender in this study. Although there are significant improvements to be made with the hyperparameter tuning. Given more resources, the trials would be allowed to converge before being terminated. The preprocessing techniques deployed also have potential. Instead of relying on the red channel a more detailed study could be made on which light frequencies are attenuated the most and the colours could be

adjusted to discard these frequencies using a Fourier transform. The example image for canny edge detection in figure 3 clearly shows the outline of a small fish. If the thresholds are adjusted a similar result could be obtained on a significant proportion of the images to increase performance. This study has not broken records but its methods show promise for further experimentation.

7. Contributions

This is a two person project and each member contributed to a large degree on every aspect of the project. The work was completed in close proximity with many tasks being shared coherently in real time.

References

[1] Derya Akkaynak and Tali Treibitz. Sea-thru: A method for removing water from underwater images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[2] Robert Geirhos, David H. J. Janssen, Heiko H. Schütt, Jonas Rauber, Matthias Bethge, and Felix A. Wichmann. Comparing deep neural networks against humans: object recognition when the signal gets weaker, 2017.

[3] Nicolas Gruber, Dominic Clement, Brendan R. Carter, Richard A. Feely, Steven van Heuven, Mario Hoppema, Masao Ishii, Robert M. Key, Alex Kozyr, Siv K. Lauvset, Claire Lo Monaco, Jeremy T. Mathis, Akihiko Murata, Are Olsen, Fiz F. Perez, Christopher L. Sabine, Toste Tanhua, and Rik Wanninkhof. The oceanic sink for anthropogenic co¡sub¿2¡/sub¿ from 1994 to 2007. *Science*, 363(6432):1193–1199, 2019.

[4] Vijiyakumar Krishnan, Govindasamy Vaiyapuri, and Akila Govindasamy. Hybridization of deep convolutional neural network for underwater object detection and tracking model. *Microprocessors and Microsystems*, 94:104628, 2022.

[5] Kukil. Mean average precision (map) in object detection. *Learn OpenCV*, 2022.

[6] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[7] Yoshihiko Ozaki, Yuki Tanigaki, Shuhei Watanabe, Masahiro Nomura, and Masaki Onishi. Multiobjective tree-structured parzen estimator. *The Journal of artificial intelligence research*, 73:1209–1250, 2022.

[8] Malte Pedersen, Joakim Bruslund Haurum, Rikke Gade, Thomas B. Moeslund, and Niels Madsen. Detection of marine animals in a new underwater dataset with varying visibility. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[9] Sovit Ranjan Rath. Pipeline to train pytorch faster rcnn object detection model. *Debugger Cafe, Machine Learning and Deep Learning*, 2021.

[10] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[11] Grigory Serebryakov and Satya Mallick. Fully convolutional image classification on arbitrary sized image. *Learn OpenCV*, 2020.

[12] Minghua Zhang, Shubo Xu, Wei Song, Qi He, and Quanmiao Wei. Lightweight underwater object detection based on yolo v4 and multi-scale attentional feature fusion. *Remote Sensing*, 13(22), 2021.