Fish Detection using Yolov8 on ROV images

Federico Staffolani[1], Enrico Maria Sardellini[2]
Department of Information Engineering,
Università Politecnica delle Marche
[1]s1114954@studenti.univpm.it, [2]s1120355@studenti.univpm.it

November 2024

Abstract

In this study, we address the challenge of detecting fish in underwater images captured by Remotely Operated Vehicles (ROVs) used by ENI in the Adriatic Sea. The goal is complex due to the low quality of the images, the variety of fish species, and the difficult environmental conditions such as water turbidity, depth, and the presence of marine objects such as algae and parts of oil platforms. We applied the YOLOv8 object detection architecture, using both small and large models, and further refined the model selection process by developing a custom weighted sum function to select the best training epoch, balancing different performance parameters such as mAP, precision, recall and validation losses. To improve the generalisability of the model, we used 5-fold cross-validation and tested different training configurations to investigate how data can affect model performance. Our final model, YOLOv8 Small with online data augmentation, gave the best results with an accuracy of 0.639, a recall of 0.599, an F1 score of 0.616 and a mAP50 of 0.574. This configuration minimised the carbon footprint to 3.90 kg and reduced the training time to 41 hours. These results demonstrate the potential of the model for fish detection, although limitations related to image quality remain, indicating the need for further dataset refinement.

Keywords: fish detection, YOLOv8, ROV images, underwater detection, computer vision

Authors' contribution: Federico Staffolani and Enrico Maria Sardellini contributed equally to this work.

1 Introduction

In recent years, the use of Remotely Operated Vehicles (ROVs) to observe and document marine life has become central to many environmental monitoring operations.

In practice, a Remotely Operated Vehicle is an unmanned underwater vehicle that is controlled from a remote location to ensure the safety of the pilot. They are typically used in oil wells and very deep water where the pressure would be fatal to the pilot on board.

The International Marine Contractors Association (IMCA) recognises five classes of ROVs. First class ROVs, the type that provided the imagery for our study, are small, fully electric, camera-equipped ROVs that can reach a maximum depth of 300m and have a lifting capacity of up to 10kg. As mentioned above, this class of ROV is often used for visual and cathodic inspection of platforms due to their small size and ease of manoeuvrability within structures. Recently, they have also been used to inspect pipelines and even fish farms.

In our case study, the problem statement begins with ENI, an Italian multinational energy exploration and production company, which uses ROVs on a daily basis to survey fish abundance around offshore platforms in the Adriatic Sea. This data is essential for assessing the ecological impact of artificial structures on local marine ecosystems, and provides insights for possible removal or modification to protect the environment.

Manual analysis of images collected by ROVs is labour-intensive and error-prone, given the large volume of data collected daily and the inherent variability of the underwater environment. Fish captured in these images can vary significantly in shape, colour and size depending on the species, while environmental conditions such as depth, turbidity, lighting and the presence of extraneous elements (such as algae or platform fragments) can greatly affect image quality. In addition, small or distant fish can blend into the background, further complicating automated visual identification.

To address these challenges, deep learning techniques, especially object detection models, have shown great promise in analysing images in complex environments such as underwater scenes. In fact, models based on convolutional neural networks (CNNs) can learn directly from data, identifying patterns and visual features even under high variability. Compared to multi-stage recognition methods, single-stage approaches such as the YOLO (You Only Look Once) architecture offer an ideal balance between accuracy and speed, which is crucial in scenarios where fast response times are required.

With the recent introduction of YOLOv8, object detection has been further improved in terms of accuracy and adaptability to environmental variability, thanks to an architecture that combines backbone, neck and head components for multi-scale detection and feature fusion. This makes this neural network particularly suitable for our task, where the system needs to accurately distinguish between different species of fish under varying lighting and visibility conditions.

The aim of this study is to develop a YOLOv8-based pipeline for automated fish detection in ROV-collected imagery, minimising the impact of environmental variability and species-specific characteristics. Our implementation introduces also a weighted sum function to automatically select the best trained model, optimising not only the mean average precision (mAP), but also other critical metrics such as precision, recall and validation losses.

Through 5-fold cross-validation and the use of data augmentation techniques, this work further explores how data variability affects model performance, with the aim of building a system capable of effectively generalising to new environmental contexts.

The integration of deep learning techniques in this application area can represent a significant step forward in the automation of marine ecological analysis, with potential benefits for the conservation and management of offshore infrastructure.

2 State of art

In general, object detection in underwater environments has progressed significantly due to two main methods: Two-stage and single-stage methods. Two-stage methods, such as R-CNN and its successors (Fast R-CNN and Faster R-CNN), start with the generation of region proposals, followed by feature extraction and classification. Although these methods can achieve high accuracy, their computational intensity and low processing speed limit their suitability for real-time applications, especially in dynamic environments where Remotely Operated Vehicles (ROVs) are deployed.

In contrast, one-step methods such as YOLO simplify the detection process by predicting bounding boxes and class probabilities in a single step, greatly improving processing speed. This is particularly useful in scenarios where immediate feedback is required. However, one-stage methods can have problems with class imbalance and misclassification, especially when some species are underrepresented in the training datasets.

Recent advances in fish detection have used these capabilities to address the complexity of object detection in underwater environments. For example, an early study by Li et al. (2015) [1] used Fast R-CNN to achieve a mean accuracy (mAP) of 81.4%, which was improved to 82.7% using the Faster R-CNN model (Li et al., 2016) [2]. However, these works have often relied on datasets such as Fish4Knowledge, which, while useful, have limitations such as low resolution and unrepresentative clipping, and fail to capture the full diversity of real-world conditions. Indeed, many other existing datasets, including Rockfish and QUT, suffer from insufficient diversity and representation of underwater environments.

For the classification part alone, the study by Li et al. (2022) [3] proposed a new method called Tripmix-Net, a CNN-based fusion model that combines information from residual networks at multiple scales. Experiments on a dataset of 15 categories of fish demonstrated the effectiveness of the model, which achieved an accuracy of 95.31%, significantly outperforming traditional methods. This approach represents a new concept for fine-grained classification of fish images in complex contexts.

Al Muksit et al. (2022) [4] introduced new datasets, such as DeepFish and OzFish, to help overcome these challenges. These datasets contain high-definition images of marine habitats. They used these datasets to develop and test two new YOLO-Fish models based on Yolov3, which showed significant improvements in detection.

A recent study by Connolly et al. (2022) [5] adapted automated detection and classification frameworks for video recorded with ROVs in shallow marine waters, addressing issues related to the use of mobile cameras. They tested the performance of three CNN models (Detectron Faster R-CNN, Detectron2 Faster R-CNN and YOLOv5) for the automated detection and counting of fish of two species, Girella tricuspidata and Acanthopagrus australis, achieving F1 scores between 85.9% and 91.4%. The study found that despite the additional challenges posed by mobile cameras, such as blurring and object size variation, the YOLOv5 and Detectron models managed to maintain high performance. This research represents an important step towards efficient image processing and generalisation of detection models in dynamic contexts.

Instead, in our work we differentiated our approach by training on the recently released YOLOv8, an advanced single-stage architecture optimised for speed and accuracy. Unlike Faster R-CNN, which may not meet the real-time requirements of ROV-based applications, YOLOv8 enables fast and efficient underwater image processing, supporting robust fish detection in varying environmental conditions. Using YOLOv8's anchor-free design and integration of multi-scale features, our method aims to optimise model performance while ensuring compatibility with real-time operations. This distinguishes it from previous studies that relied on earlier versions of YOLO or slower, two-stage models.

3 Materials and methods

3.1 Dataset and Preprocessing

The initial material provided by ENI comprised nine folders, one for each ROV from which the acquisitions were made, with a total of approximately 4,000 images. The complete list of initial datasets in alphabetical order consists of: Agostino B, Amelia A, Barbara A, Barbara H, Cervia A, Cervia C, Emma, Fratello Cluster, PCWB.

An initial analysis showed that the information content was heterogeneous for various reasons. Firstly, there were several images that did not contain any fish, and consequently no annotations. Furthermore, there were differences in the semantics of the labels attached to the fish. Some of the labels were the biological category name of the fish, rather than a more generic label. Finally, there were numerous point annotations located in the centre of the fish, which were problematic for the object detection task as they did not provide a bounding box.

We also noticed that the Emma and Barbara A datasets, in addition to being the largest, had very high quality images; however, in some images we found numerous shoals of fish that were either not labelled correctly or with almost no labelling at all, which could cause several problems during training.

In terms of preprocessing, our initial focus was on the structure of the datasets, including aspects such as image size, number of labels and overall size.

The first step was to remove the images without annotations, leaving us with a total dataset of about 2600 images.

Then, using an automatic script, all labels were replaced with 'fish' to reduce the problem to the distinction between the generic class and the background, without any semantic ambiguities between the different species. It was then necessary to convert all point annotations with centred bounding boxes, with the chosen size of 20x20, so it must be taken into account that this approximation may have introduced some inaccuracies in the correct coverage of objects with boxes.

Furthermore, following a manual examination of the dataset, it became evident that a number of bounding boxes, added by those who had labelled the images, were of an inappropriate size for the fish and thus required resizing via the Roboflow interface.

As a final preprocessing step, we manually eliminated any images that contained dense shoals of fish without annotation, as it would take time to make them suitable for training through manual annotation, so that we could save them for maximum use in testing. The structure of the various processed datasets is summarised in the Table 1.

Dataset	Num of images	Num of labels	Labels/images
Agostino B	37	390	10,5
Amelia A	20	61	3,1
Barbara A	1253	37.760	30,1
Barbara H	91	5.120	56,3
Cervia A	274	2.279	8,3
Cervia C	69	1.387	20,1
Emma	768	21.744	28,3
Fratello Cluster	30	752	25,1
PCWB	87	622	7,1

Table 1: Number of images, labels and ratio between them

In order to obtain the most generalisable results for the model, we chose a 5-fold cross-validation.

Initially, the datasets with the smaller number of images was merged with other datasets with similar characteristics, including brightness given by depth, water color. This was done to ensure consistency across subdivisions. Table 2 illustrates the aggregation of datasets based on the previous characteristics and sizes.

The datasets aggregated include: Agostino B and Cervia C, Fratello Cluster and Barbara A, PCWB and Amelia A. We left the other 3 separate resulting in a total of 6 datasets.

Dataset	Water color	Depth	Average fish size
Agostino B	Dark green	5-10m	0,14%
Amelia A	Aqua green	6-8m	0,18%
Barbara A	Light blue/Gray/Black	2-72m	0,13%
Barbara H	Light blue	5-40m	0,11%
Cervia A	Light blue/Aqua green	3-25m	0,10%
Cervia C	Green	8-20m	0,10%
Emma	Light blue/Dark blue	0-70m	0,14%
Fratello Cluster	Light blue	2-10m	0,06%
PCWB	Light green	4-6m	0,11%

Table 2: Environmental characteristics and average size of fish as a percentage of the total image in datasets

Next, we used a script that takes six datasets as input and divides each of them into five portions to perform cross-validation. In this process, each input dataset is divided into five portions, where each portion represents a fraction of the dataset used in the cross-validation process. In each loop (or fold), one of the portions is selected as the test set, while the remaining four portions are combined to form the training set.

This partitioning process can be expressed mathematically as follows: given a data set D with n observations, the data set is partitioned into five parts D_1, D_2, D_3, D_4, D_5 so that each part contains $\frac{n}{5}$ observations. During each cross-validation cycle, one of these parts (e.g. D_i) is used as the test set, while the combination of the other four parts D_j, where $j \neq i$, constitutes the training set.

The process is repeated for each of the five folds k, where $k = 1, 2, 3, 4, 5$, ensuring that each part of the data set is used once as a test set and four times as a training set. Within each fold, the split is such that 80% of the data is allocated to training and 20% to testing. This split allows for a comprehensive and balanced evaluation of the model across all parts of the dataset, mitigating bias due to data distribution.

3.2 YOLOv8 Architecture

The YOLO neural network divides an input image into a grid of cells, where each cell is responsible for predicting the probability of an object's presence, the bounding box coordinates, and the object's class. This single-pass approach avoids the need for region proposals, enabling real-time detection.

Unlike anchor-based models, YOLO is anchor-free. This means it directly predicts the object center rather than calculating the offset from predefined anchor boxes, reducing the number of box predictions needed. This approach speeds up Non-Maximum Suppression (NMS)—a post-processing step that eliminates redundant bounding boxes, ultimately streamlining the inference process. It's architecture consists of 53 convolutional layers and includes cross-stage partial connections to enhance information flow across layers.

The structure is made up of three main components: Backbone, Neck and Head.

The Backbone captures essential features from the input image and provides a hierarchical representation, identifying low-level patterns in initial layers and more complex patterns in deeper layers. It also supports multi-scale representation to capture features at different levels of abstraction.

The Neck bridges the Backbone and Head and performs feature fusion to integrate contextual information. It creates feature pyramids by aggregating the feature maps produced by the Backbone, improving the accuracy of recognition for objects at different scales.

The Head is the final component responsible for producing the output, including bounding boxes and object confidence scores for detection.

In addition to these three essential building blocks, several other components contribute significantly to the effective functioning of a YOLO model.

For example, the Bottleneck block consists of a convolution block paired with a shortcut connection, also known as a skip or residual connection. This shortcut connection allows for a smoother gradient flow during training, which helps to alleviate the vanishing gradient problem, thereby increasing the model's learning efficiency.

Another important block is the C2f block, which starts with a convolutional layer that splits the feature map. One part of this split feature map is sent to the Bottleneck block, while the other part is sent directly to the Concat block. The depth multiple parameter determines the number of bottleneck blocks used within the C2f structure.

Next is the SPPF (Spatial Pyramid Pooling Fast) block, which contains a convolutional layer followed by three MaxPool2D layers. These pooled feature maps are then concatenated and fed through another convolutional layer, allowing YOLO to handle images of different resolutions. By pooling information at multiple scales, the SPPF block improves the network's adaptability to different image sizes.

Finally, the MaxPool2D layers play a critical role by reducing the spatial dimensions of the input, effectively reducing computational cost and isolating dominant features. The degree of downsampling achieved by these layers is determined by pooling parameters such as kernel size and stride, optimising the ability of the mesh to capture essential detail while remaining computationally efficient.

The YOLOv8 model family includes several variants, each of which balances speed, accuracy and model size to meet different requirements. These variants are primarily distinguished by specific parameters: the depth multiple (d), which determines the number of bottleneck blocks within the C2f block; the width multiple (w), which adjusts the number of channels in the convolutional layers; and the max channel (mc), which sets an upper limit on the number of channels within the network. By tuning these parameters, each variant can be optimised to prioritise either a more compact and faster model or a larger model with higher accuracy, depending on the application requirements.

3.3 Hyperparameters and Augmentation settings

In our experiment, we trained all models using k-fold cross-validation as previously mentioned. Each training session took place on a remote server using docker containers on one or two GPUs provided to us. Table 3 summarises the different configurations used in the various trainings.

Initially, we trained both small and large versions of YOLOv8 for 100 epochs to evaluate the behaviour of the network on this particular task. In these initial trainings, the batch size was set to 8 to optimise the GPU load, and the number of workers was set to 0 due to parallelism issues, while the image size was set to 640x640 to preserve the original aspect ratio.

Two additional training runs were then run using the small version for 300 epochs. The first retained identical hyperparameters to the previous training, while the second used a batch size of 16 due to increased GPU availability and applied online augmentation to the training set.

Given the considerable variability and challenges of the underwater dataset, we introduced online augmentation during the second training session to improve model robustness and generalisation. The dataset includes images with different lighting, visibility and environmental conditions, as well as fish of different sizes, shapes and orientations. Applying data augmentation helps to simulate these variations in the training data, allowing the model to learn more diverse feature representations. This is important in our context, as it allows the model to better generalise to unseen images, where fish may appear under different lighting, or in unusual poses or scales.

Online augmentation was implemented using the Ultralytics library, which applies augmentation randomly and on-the-fly for each batch. This approach allows for a consistent flow of varied training images, improving model generalisation without the need for manual adjustments to individual images.

The following specific augmentation settings were employed:

- **hsv_s=0.4**: adjusts saturation up to 40%, altering fish colors to simulate different lighting and water conditions.
- **hsv_v=0.4**: varies brightness by up to 40%, useful for mimicking diverse light levels underwater.
- **scale=0.5**: scales images by up to 50%, representing fish at varying distances and sizes.
- **fliplr=0.5**: enables horizontal flipping at 50%, increasing image variety and enhancing model adaptability to different orientations.

This augmentation strategy allowed us to create a more resilient model capable of effectively identifying fish across varied environmental conditions, a crucial requirement for real-world underwater detection applications.

Model	**Epochs**	**Batch Size**	**Image Size**	**Online Augmentation**
YOLOv8 Small	100	8	640x640	No
YOLOv8 Large	100	8	640x640	No
YOLOv8 Small	300	8	640x640	No
YOLOv8 Small	300	16	640x640	Yes

Table 3: Summary of Hyperparameters

3.4 Model Selection and Weighted Sum Function

After training, once the results were obtained, we reasoned on how to choose for each fold the epochs on which to make inference.

Ultralytics selects the best model for inference using the `best.pt` file, which is saved at the epoch with the highest fitness score. The fitness function used by default is a weighted combination of several metrics, primarily focusing on mAP and setting the precision and recall coefficient to zero. Specifically, the fitness is calculated as:

$$\text{fitness}_i = 0.0 \cdot P_i + 0.0 \cdot R_i + 0.1 \cdot \text{mAP}_{50,i} + 0.9 \cdot \text{mAP}_{50:95,i}$$

This method emphasizes the mean Average Precision (mAP) across different IoU thresholds, especially prioritizing mAP at 0.5:0.95.

In contrast, we propose an alternative method that incorporates both mAP at different IoU thresholds and the F1 score, while penalizing higher losses. Our custom function is defined as:

$$\max_{1 \le i \le \text{last}} f_i = \text{mAP}_{50,i} \cdot 0.25 + P_i \cdot 0.375 + R_i \cdot 0.375 - L_{\text{box},i} \cdot 0.5 - L_{\text{cls},i} \cdot 0.5$$

$$i^* = \arg \max_{1 \le i \le \text{last}} f_i$$

In this formulation:

$$\begin{aligned}
P_i &\text{ is the precision at epoch } i,\\
R_i &\text{ is the recall at epoch } i,\\
L_{\text{box},i} &\text{ is the validation box loss at epoch } i,\\
L_{\text{cls},i} &\text{ is the validation class loss at epoch } i,\\
\text{mAP}_{50,i} &\text{ is the mean Average Precision at IoU = 50\% at epoch } i,\\
i^* &\text{ is the epoch that maximizes } f_i\text{, chosen as the best for inference.}
\end{aligned}$$

This approach provides a more balanced evaluation for a detection task, taking into account not only mAP, but also accuracy, recall and both classification and box losses, leading to a potentially more robust selection of the optimal model with less risk of over-fitting.

To check whether our formula was indeed a valid alternative to the default Ultralytics formula, we carried out a measurement of the results of the validation test inferences for each fold, comparing the epoch chosen by the library as the best and the one chosen by our function. This evaluation was conducted using the YOLOv8 Small training results with 300 epochs.

Measurements were performed using a script that only considered bounding boxes that exceeded a threshold of 0.6. "Boxes detected" refers to the number of bounding boxes obtained by inference, while "Average confidence" refers to the average of the bounding box values, or the confidence of the model in identifying fish. The values are obtained from the difference between the values of the two models considered, and if there is a positive value it means that the epoch chosen by our function performs better than the default epoch, vice versa it will be negative.

A summary of these results can be found in the Table 4, and judging from the higher number of positive values, we can conclude that our custom function identified a better epoch than the default one.

Metrics	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Total**
Box recognised	+278	+79	-165	-277	+254	+169 (SUM)
Average confidence	+0.01	+0.01	+0.01	+0.02	0	+0.01 (AVG)

Table 4: Measurement of inference differences with threshold of 0.6

4 Results

As described above, our first test was conducted using the YOLOv8 Large and Small models, both trained for 100 epochs. The aim was to test whether the Large version would produce results with a higher accuracy than the Small version, while requiring a longer training time.

As shown in Table 5, the results of the Large model are slightly better in all the parameters considered (precision, recall, F1-score and mAP50). However, from the

Figures 1 and 2, it is evident that the box loss and class loss curves have not yet reached stationarity, indicating a residual room for improvement in training.

Method	Precision	Recall	F1	mAP50	Co2	Time
Large 100	0.616	0.592	0.601	0.558	2.92 kg	22 h
Small 100	0.608	0.586	0.595	0.555	2.00 kg	18 h
Small 300	0.630	0.598	0.611	0.571	4.85 kg	57 h
Small 300 with Aug	0.639	0.599	0.616	0.574	3,90 kg	41 h

Table 5: Average, for each method, of best model maximums values for each fold

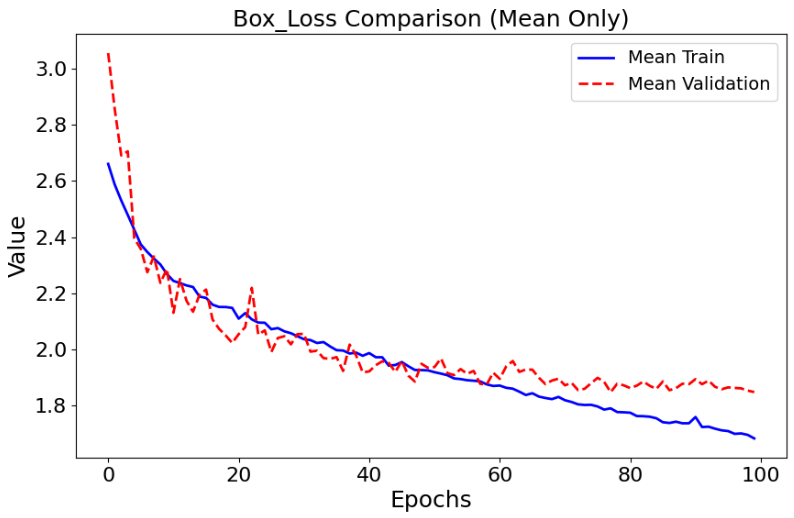

Figure 1: Box loss Large 100

By this time, however, we only had one GPU available, which forced us to choose which model to continue our study on. We opted to continue training with the Small model for 300 epochs, since although the Large model had shown slightly better results, it would not have been justified to spend more time and CO_2 emissions, quantified through measurements using the CodeCarbon library, for a marginal advantage in terms of accuracy.

As shown in Figure 3, the training curve has reached a certain stationarity, while the validation curve shows a beginning of overfitting.

Analysing Figure 4, we can derive two further pieces of information: firstly, fold 5 is the one with the lowest precision values, thus penalising the model's overall average. Furthermore, looking at the curve of fold 5, we can see that the training stopped before the 300 epochs. This behaviour, also observed in other folds, is due to the optimiser Adam, who stopped the model and proceeded to the next fold when he did not detect any significant improvement in the last 100 epochs.

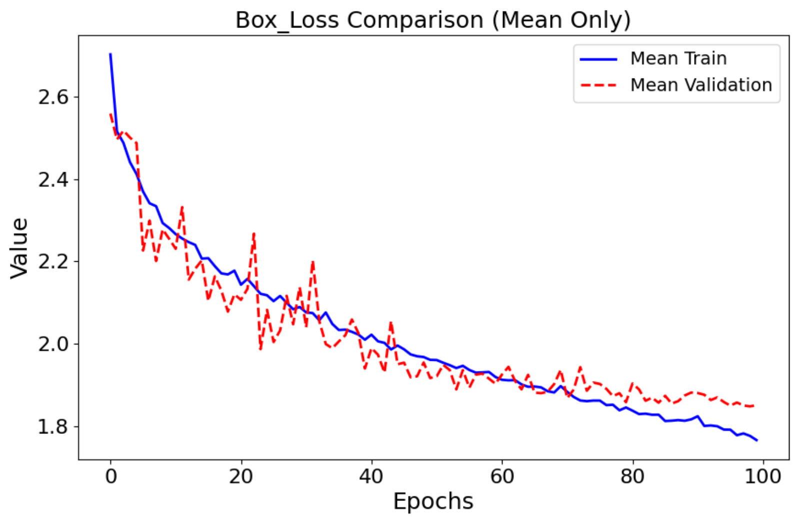

Figure 2: Box loss Small 100

Looking back at Table 5, we can see that the results showed improvements in all parameters, although they are not as optimal as expected. This is due to the quality of the dataset, which has blurry images and very small fish in the background, elements that are difficult to detect and on which the model struggles to guarantee a safe classification as we can see in Figure 5. In order to determine the inferences, we developed an additional script that allows, once the model and the folder containing the images have been selected, to generate the images with bounding boxes by applying a predetermined threshold. This script was used with our models, trained on an additional fold of images that we employed as a test set and the threshold was set at 0.5.

The last test was carried out starting with the Small version with 300 epochs and adding online augmentation with the parameters given above, to allow the model to detect less visible fish more effectively, thus improving the metrics. As this would have increased the number of images to be trained, resulting in a greater use of time and resources, we chose a batch size of 16 to optimise speed.

Although the problem with the values in fold 5 was not solved, as can be seen in Figure 6 and in Figure 7, the results improved compared to the version without augmentation, while training time and CO_2 emissions decreased drastically due to a higher batch size. Comparing the two inferences, both strengths and weaknesses emerge. In both cases, it is observed that fish, especially those in the foreground, are detected with greater certainty. Online augmentation seems to have improved accuracy, allowing the model to recognise more fish with greater certainty than the previous version. On the other hand, the models seem to have difficulty recognising many of the fish in the blurred background, although this varies from image to image. As shown in Figure 8, the model can recognise a shoal of very small fish with good confidence.

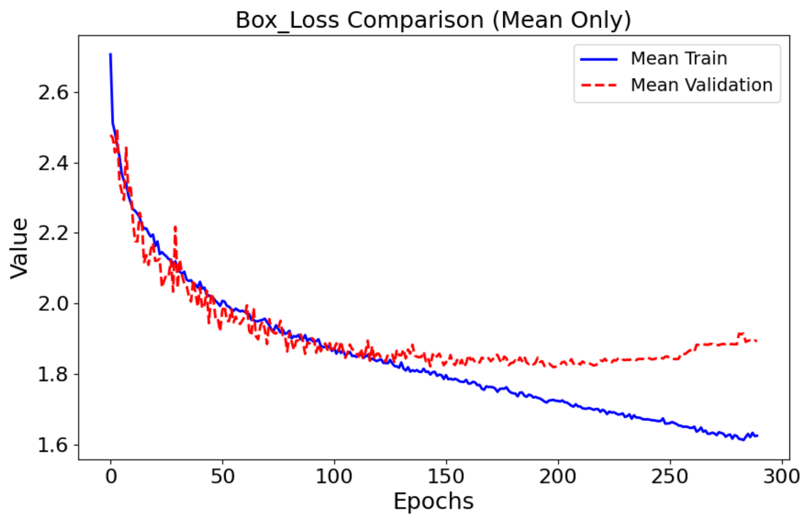

Figure 3: Box loss Small 300

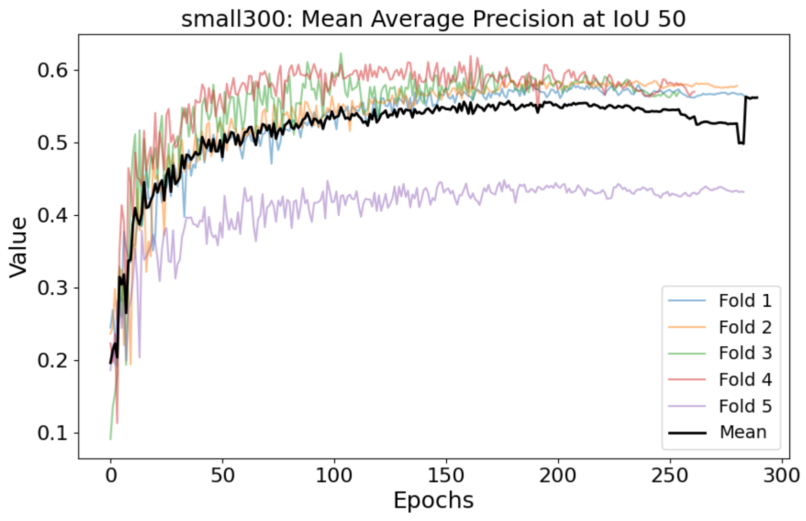

Figure 4: mAP50 Small 300

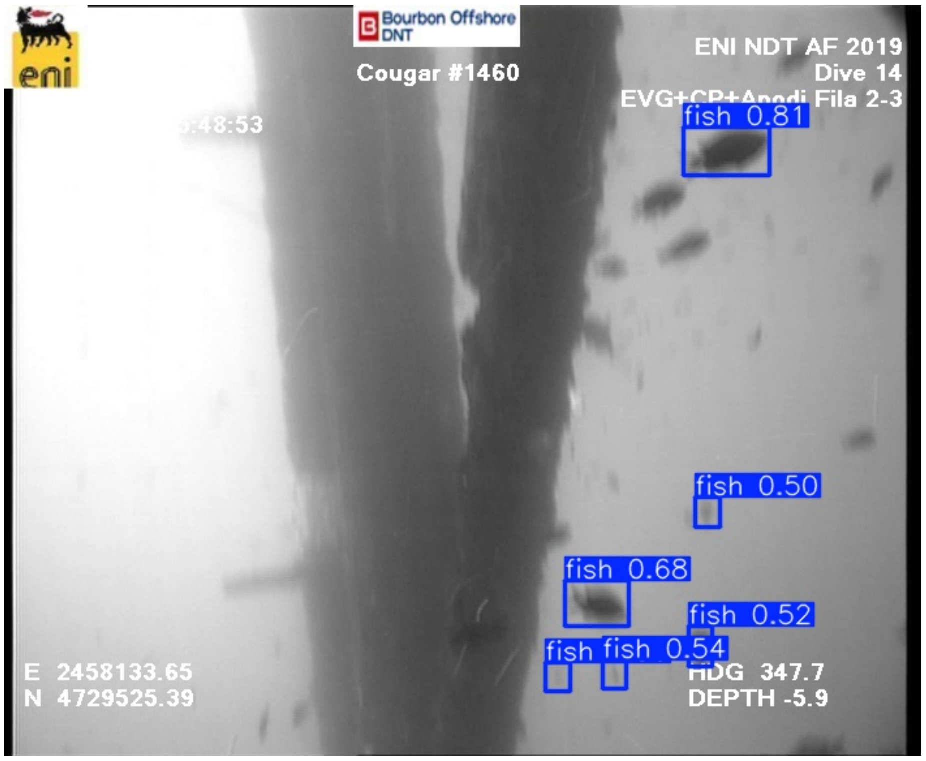

Figure 5: Blurred image (frame 7), Small 300 model (fold 1, epoch 225)

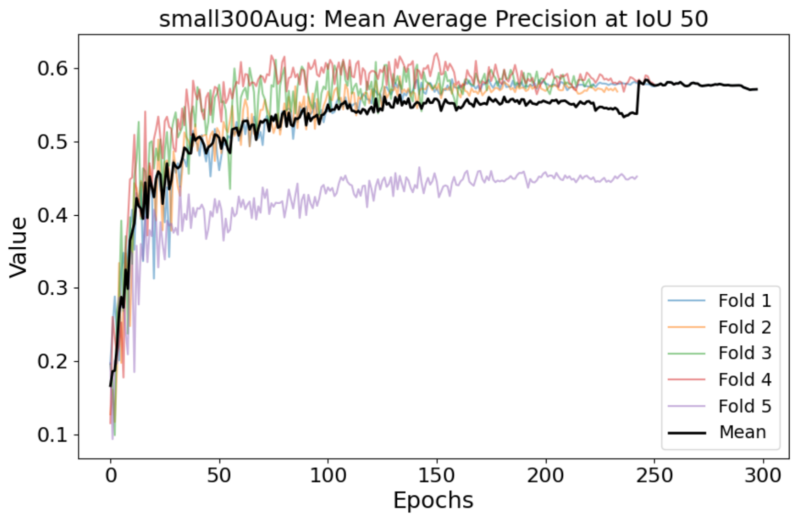

Figure 6: mAP50 Small 300 with augmentation

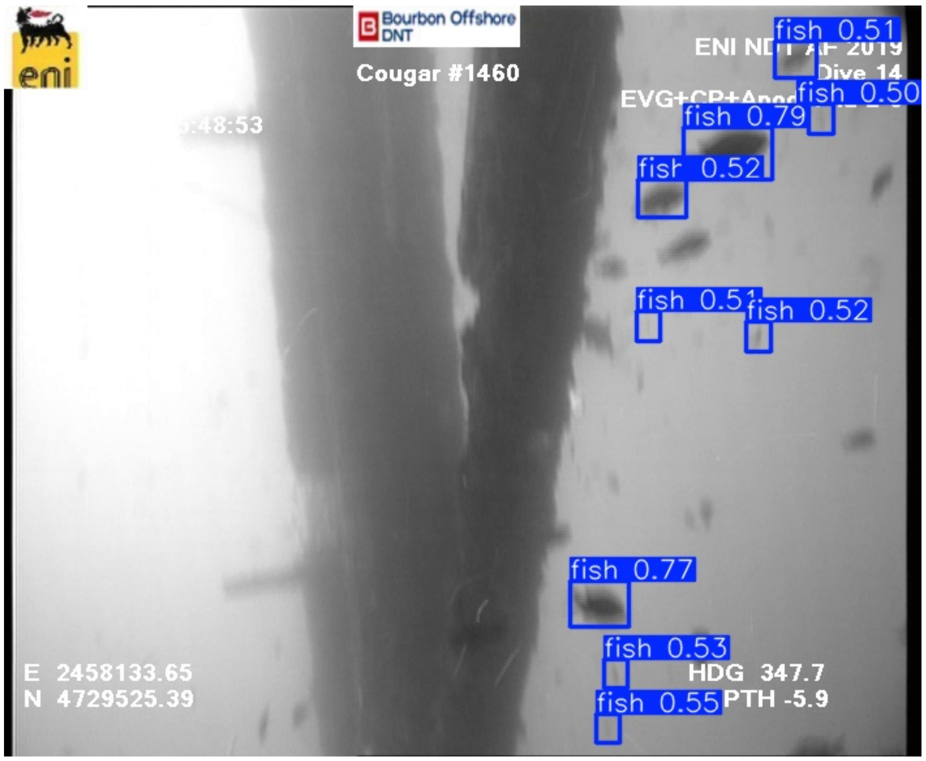

Figure 7: Blurred image (frame 7), Small 300 Augmented model (fold 1, epoch 198)

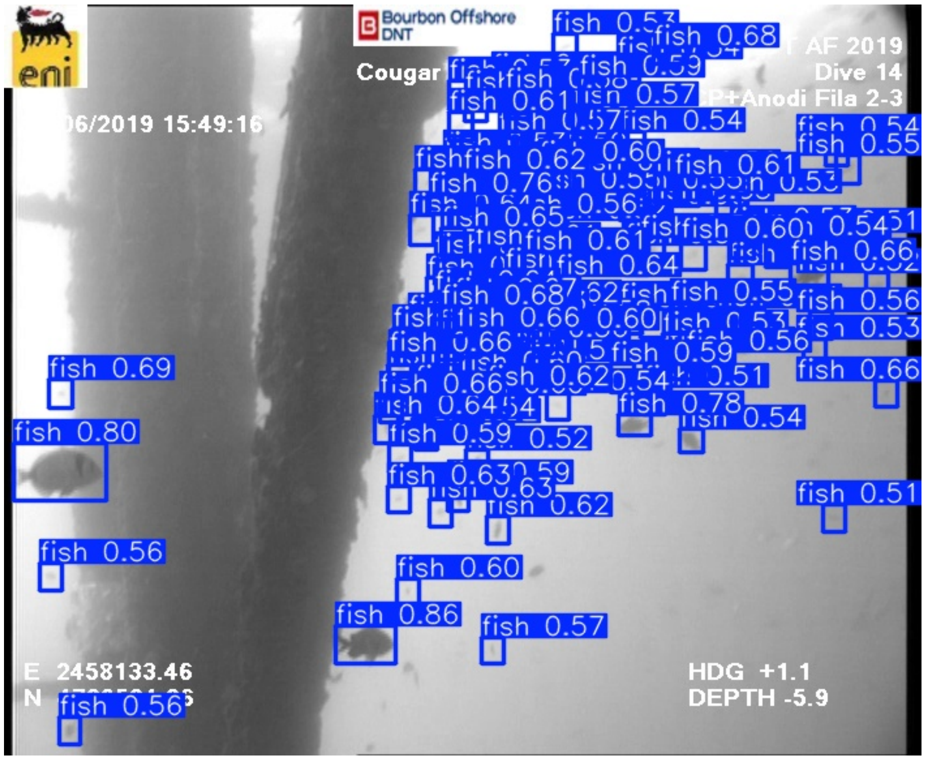

Figure 8: Shoal of fish (frame 14), Small 300 model (fold 5, epoch 196)

5 Conclusions and future work

This study evaluated the application of YOLOv8 models for fish detection in ROV imagery, addressing the challenges posed by species variability, environmental factors, and problems posed by small fish in the background.

Prior to k-fold cross validation, extensive preprocessing was undertaken to optimise the dataset. However, it is recognised that the initial approximation using automatic labelling for punctiform annotations and manual relabelling of some erroneous bounding boxes may have introduced errors that affected the overall accuracy of the model. This highlights the need for future efforts to further improve image quality, as a refined dataset would likely improve model performance by reducing noise from erroneous labels. One way of doing this could be to use Roboflow's intelligent automatic labelling system, which accurately selects the full contour of objects of interest.

In these experiments, both the YOLOv8 Large and Small models were tested, with the Large model showing slightly better performance in terms of precision, recall and mAP50. However, due to the longer training time and higher CO_2 emissions associated with the Large model, the focus was placed on the YOLOv8 Small model, which was trained over 300 epochs with data augmentation to balance accuracy and efficiency.

Results showed that extended training and data augmentation led to improvements in all performance metrics, although limitations due to dataset quality remained. Blurred images and small fish in the background posed particular challenges to detection accuracy, while the model's performance on fold 5 revealed potential inconsistencies in the data that warrant further analysis.

Future work should also consider improvements in data enhancement techniques. Given the challenges posed by the imagery and the difficulty of detecting small or partially obscured fish, more targeted augmentation could help make the model more robust to these conditions. Also, experimentation with other newer architectures, such as YOLOv11 models, could also provide valuable insights. These models may offer improvements in detection accuracy and efficiency, potentially surpassing the capabilities of YOLOv8 and addressing the specific challenges of underwater imaging.

In summary, this study highlights the potential of YOLOv8 models for fish detection in ROV imagery and suggests that targeted data preprocessing, robust augmentation and architectural advances could further improve performance. The development of a reliable detection system tailored to ENI's environmental assessments of marine ecosystems remains an achievable goal, providing critical support for data-driven decision making in offshore infrastructure management.

References

[1] Xiu Li, Min Shang, Hongwei Qin, and Liansheng Chen. Fast accurate fish detection and recognition of underwater images with fast r-cnn. In *OCEANS 2015 - MTS/IEEE*, pages 1–5, Washington, 2015. IEEE.

[2] Xiu Li, Min Shang, Jing Hao, and Zhixiong Yang. Accelerating fish detection and recognition by sharing cnns with objectness learning. In *OCEANS 2016 - Shanghai*, pages 1–5. IEEE, 2016.

[3] Lipeng Li, Feipeng Shi, and Chunxu Wang. Fish image recognition method based on multi-layer feature fusion convolutional network. *Ecological Informatics*, 72:101873, 2022.

[4] Abdullah Al Muksit, Fakhrul Hasan, Md. Fahad Hasan Bhuiyan Emon, Md Rakibul Haque, Arif Reza Anwary, and Swakkhar Shatabda. Yolo-fish: A robust fish detection model to detect fish in realistic underwater environment. *Ecological Informatics*, 72:101847, 2022.

[5] Rod M. Connolly, Kristin I. Jinks, Cesar Herrera, and Sebastian Lopez-Marcano. Fish surveys on the move: Adapting automated fish detection and classification frameworks for videos on a remotely operated vehicle in shallow marine waters. *Frontiers in Marine Science*, 9, 2022.

Websites consulted:

- Roboflow Docs – `https://docs.roboflow.com/`
- Ultralytics Docs – `https://docs.ultralytics.com/`
- CodeCarbon – `https://codecarbon.io/`
- Detailed Explanation of YOLOv8 Architecture – `https://medium.com/@juanpedro.bc22/detailed-explanation-of-yolov8-architecture-part-1-6da9296b954e`
- Understanding YOLOv8 Architecture – `https://www.labellerr.com/blog/understanding-yolov8-architecture-applications-features/`
- YOLOv8 Architecture Explained – `https://abintimilsina.medium.com/yolov8-architecture-explained-a5e90a560ce5`
- Wikipedia ROV – `https://en.wikipedia.org/wiki/Remotely_operated_underwater_vehicle`