Real-Time Underwater Fish Detection and Species Classification Using YOLOv7 Optimized for Google Coral

Jan M. Straub
M.S. Data and Computer Science
Mat.-Nr. 3405670
jan.straub@stud.uni-heidelberg.de

Abstract

This paper builds upon the work of Salman et al. [9], titled "Automatic fish detection in underwater videos by a deep neural network-based hybrid motion learning system". We took their initial idea and optimized it for mobile real-time use, specifically adapting the system for deployment on a Google Coral chip for local execution. You can find our GitHub repository here.

1. Introduction

Monitoring fish populations in coral reefs is crucial, as they are important indicators of overall ecosystem health. However, manually counting fish in underwater videos is highly unfeasible due to the time, effort, and expertise required. Automated solutions are essential to enable accurate and efficient tracking of fish populations in these complex environments. The task is difficult because of the highly variable conditions underwater, including changing lighting, the presence of numerous objects like coral and debris, and the distortion often found in underwater recordings.

In this paper, we focus on the first two steps of automated fish sampling. The first step is fish detection, which separates fish from other marine animals and background elements. The second step is fish species classification, where detected fish are assigned to predefined species categories. We exclude the third step, fish biomass measurement, as it lies beyond the scope of this work.

We build upon the work of Salman *et al.* [9], titled "Automatic fish detection in underwater videos by a deep neural network-based hybrid motion learning system".

2. Related Work

This section is divided into two parts: the first focuses on the types of datasets typically used for fish detection tasks, and the second explores the various approaches employed for detecting fish in these datasets.

2.1. Datasets

Datasets used in fish detection can be broadly classified into two categories: constrained and unconstrained. Constrained datasets are gathered in controlled environments where factors like lighting, movement, and background are carefully regulated. Typically, these datasets feature images captured in aquariums [8] or chambers specifically designed to guide fish movement [4]. This controlled setup reduces computational complexity, results in a smaller overall structure, and enables faster recognition speeds. In contrast, unconstrained datasets focus on real-world underwater videos, where conditions are highly variable and unpredictable. While these datasets involve higher computational complexity and slower recognition times, they provide a more realistic reflection of practical applications, leading to improved identification accuracy. A notable example of an unconstrained dataset is FishCLEF-2015 [1].

2.2. Fish detection

Fish detection approaches can be broadly categorized into one-step or two-step approaches. A typical one-step approach was employed by Hamzaoui *et al.* [3]. In their work they utilized transfer learning by using pre-trained model parameters to enhance YOLOv5's performance, thus significantly improving fish species recognition in aquaculture. Additionally, integrating residual structures, as proposed by Li *et al.* [6], optimizes network performance while reducing resource demands. A notable recent example in this category is FishFocusNet by Lu *et al.* [7]. They used alterable kernel convolution to extract richer features and reduce redundant calculations. An asymptotic feature pyramid network improved small fish detection, and they enhanced location accuracy by refining the regression loss.

A two-step approach to fish detection often starts with a pre-processing stage, such as applying a Gaussian Mixture Model (GMM) to isolate the foreground. In the work by Salman *et al.* [9], which our research builds upon, GMM and optical flow detection were used. These results were combined into a single image and fed into a ResNet-152 CNN. We discuss this approach in more detail later. Similarly, Jalal *et al.* [5] employed GMM and optical flow, alongside a ResNet-50 CNN, but further enhanced detection by running YOLO (You Only Look Once) in parallel, merging the outputs for more accurate results. Ben *et al.* [10] took a different approach by integrating a base network, a region proposal network, and a classifier network into a unified pipeline. This allowed them to combine appearance and motion information in a single image, improving the detection and classification of moving objects.

For a more comprehensive review of the state of fish recognition and classification methods Barbedo's [2] review provides a good starting point.

3. Approach

As noted in the Introduction, our work is based on the work presented by Salman *et al.*. This section is divided into two parts: first, we discuss the idea and implementation of their method, highlighting our improvements, and second, we detail the process of integrating the system into the Google Coral platform.

3.1. Paper Implementation

The initial step involved cleaning and updating the original code, while rewriting all necessary components in Python. As outlined in the original paper, we began by applying a Gaussian Mixture Model (GMM) to detect foreground objects in each frame. This approach helps isolate moving objects, such as fish, from the background. However, underwater environments, especially coral reefs, present significant challenges due to constantly changing backgrounds influenced by factors like lighting shifts, water quality, and the movement of coral and plants. These variations can reduce the effectiveness of traditional object detection models, making robust pre-processing useful.

One particular issue is that GMM may incorrectly classify fish as part of the background, especially in scenes where fish are static or appear in the background. This results in misdetections. To address this, the original paper introduced the Farneback optical flow algorithm detection technique, which focuses purely on capturing motion within the video. By detecting changes between consecutive frames, optical flow helps identify moving fish more accurately, compensating for the limitations of GMM in highly dynamic environments such as coral reefs.

Since neither the GMM nor the optical flow images retain the visual details of the original frame, a gray-scale version of the original frame is incorporated to preserve this crucial information. The gray-scale frame, the GMM output, and the optical flow map are combined into a single RGB image. In this arrangement, the gray-scale image occupies the red channel, the GMM output is assigned to the green channel, and the optical flow data is placed in the blue channel.

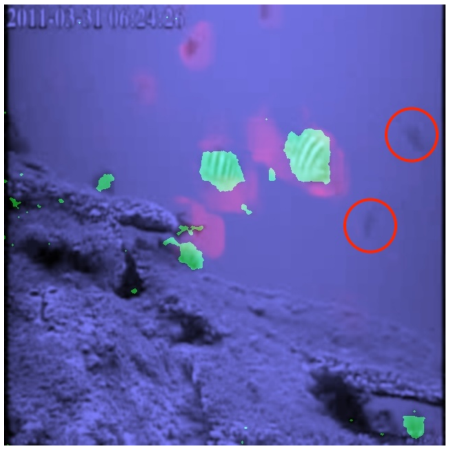


Figure 1: This is an example of a combined image YOLO receives, with the gray-scale frame in red, GMM in green, and optical flow in blue. The red circles highlight fish missed by both GMM and optical flow, underscoring the challenges of underwater fish detection. Additionally, instances where either GMM or optical flow detects something are clearly visible, demonstrating the importance of combining both methods for more accurate results.

We found that the GMM detection was too sensitive, leading to excessive noise in the results. To address this, we applied a blur to the images before processing them through the pipeline, which resulted in cleaner outputs. However, the GMM noise issue was not completely resolved. Additionally, we adjusted the optical flow parameters, reducing the downscaling from 50% to 5% of the original size to enable more detailed tracking. To enhance monitoring further, we increased the levels to capture motion with greater precision. An example of the improved combined image can be seen in Figure 1.

This approach enables YOLO to leverage pre-processing steps, such as motion detection and background separation, to effectively distinguish between different fish species. As a result, the model is better equipped to navigate the complexities of underwater environments, significantly enhancing fish detection accuracy. By utilizing multi-channel input, YOLO can not only identify the presence of fish but also differentiate between species with diverse shapes and patterns.

In the original paper, Salman *et al.* used a ResNet-152 CNN for training on the combined images. We opted for YOLOv7 [11] as our object detection network, as it is better suited for real-time applications. Given our limited hardware and the goal of deploying the model on Google Coral, we chose YOLOv7 tiny. This version has fewer layers and parameters, offering higher processing speed and

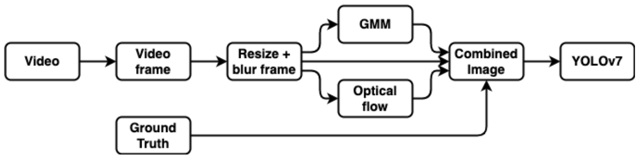


Figure 2: The frame pipeline for the YOLO training.

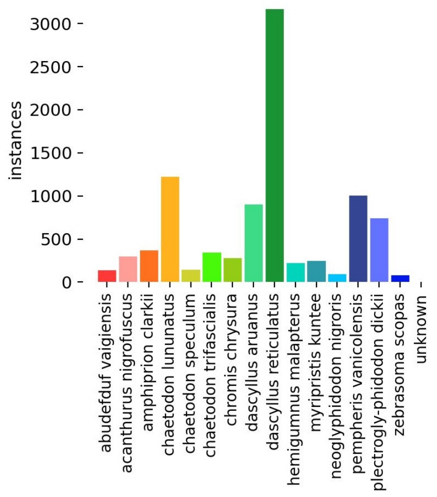


Figure 3: The distribution of instances for each fish species reveals a significant imbalance in the number of occurrences per species. This disparity leads to weaker detection and classification performance for species with fewer instances, impacting the overall model accuracy.

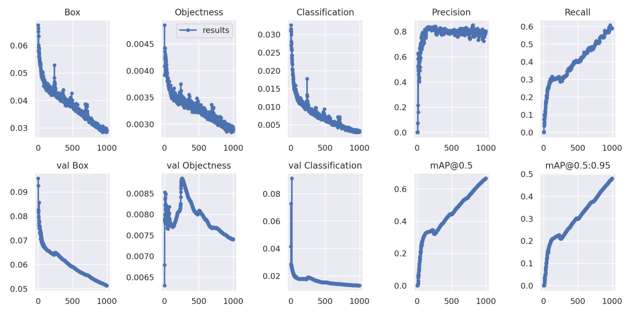
lower hardware requirements at the cost of accuracy.

To improve efficiency, we also streamlined the process by eliminating the need to save images after each pipeline step, opting to save only the ground truth and final combined images. Figure 2 outlines the image pipeline we implemented to process each video frame.

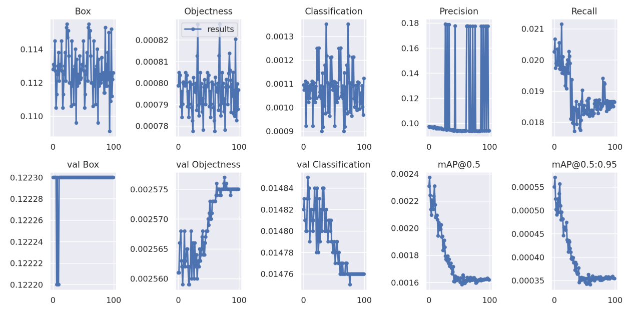
For the initial training, we used the FishCLEF-2015 [1] dataset, which contains 20 training videos and 73 validation videos featuring 15 different fish species. The distribution of fish species instances can be seen in Figure 3. The hardware used for training included an AMD Ryzen 7 CPU and an Nvidia RTX 4080 Super GPU.

3.2. Google Coral Implementation

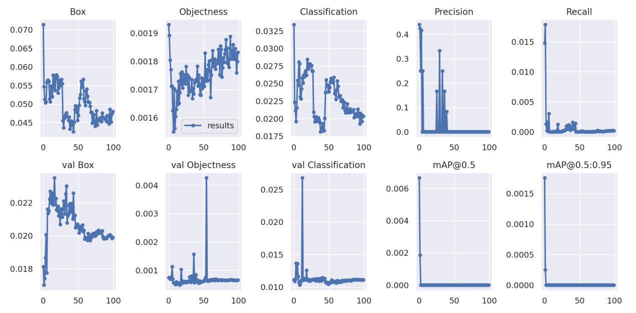
Thanks to one of our team members, we had access to a Google Coral chip and planned to use it for a mobile real-time detection application. Unfortunately, due to internal issues, we were unable to implement our solution. Since the Coral chip operates with TensorFlow Lite (TFLite), we would have needed to convert our YOLO weights to TensorFlow format first and then to TFLite. Additionally, to optimize performance on the Edge TPU, we would have had to compile the model specifically for that platform.



(a) Training results after 1000 epochs, demonstrating the model's learning progress.



(b) Training results obtained following hyperparameter tuning, showing the decline in performance.



(c) Training results with a reduced input image size of 240 pixels, illustrating the impact on model.

Figure 4: Training results of the baseline YOLOv7 model, including experiments on hyperparameter tuning and varying input image sizes.

4. Experiments

We conducted several experiments to enhance both the performance and robustness of our approach. Initially, we trained the default YOLOv7 model on our combined images to assess the training metrics. Following this, we trained the YOLOv7 tiny model from scratch. We fine-tuned the model to achieve better results, reduced the input image size to boost processing speed, and explored the integration of edge detection into the images to improve feature extraction and overall detection accuracy.

4.1. YOLOv7 baseline

The initial training run of 100 epochs produced unsatisfactory results. However, after extending the training period, there was a significant improvement, as reflected in Figure 4a. The training metrics demonstrated steady progress, indicating that the model effectively learned from the data. While the validation metrics improved, they fluctuated at times. Although there were no signs of overfitting, the model still struggled with new data during detection tests, indicating weaker generalization than anticipated, even after addressing the dataset imbalance. Additionally, the model's performance was insufficient for real-time applications, managing only around three frames per second (fps), which is far below the requirements for smooth video processing.

4.2. Hyperparameter tuning

In the first YOLOv7-tiny training run, we kept the hyperparameters at their default values to establish a baseline for future optimization. During tuning, we made several adjustments. We slightly increased the momentum, as the dataset contained significant noise, and this adjustment helped stabilize the training process. Additionally, we raised the IoU threshold to improve detection accuracy. Using the built-in k-means clustering method, we calculated optimal anchor boxes tailored to our dataset. Given the class imbalance, we increased the focal loss gamma to handle this issue better. We also applied stronger augmentations like rotation, translation, scaling, and shear to enhance the model's robustness. Finally, we leveraged YOLO's built-in evolutionary algorithm to refine the hyperparameters further automatically.

4.3. Reduce image size

The accuracy and computational complexity of YOLO models are tied to the size of the input images. By default, YOLO uses 640x640 pixel images, which balances accuracy and processing time. While YOLOv7 supports various inference resolutions, we experimented with reducing the image size to 240x240 pixels to speed up the pipeline utilizing transfer learning. This adjustment boosted the combined image creation speed to 16 frames per second (fps). However, despite the improved processing speed, the lower resolution hurt detection and classification performance, particularly for smaller fish.

4.4. Edge detection

In our final experiment, we integrated edge detection into the pre-processing pipeline, hoping to help differentiate objects in the often-ambiguous underwater environment. The idea was that sharper object boundaries could improve detection, but we were uncertain if this would conflict with the feature extraction mechanisms already built into YOLO. Additionally, we needed to weigh the increased computational cost of edge detection against any potential gains in accuracy.

5. Findings

We documented the findings of our experiments in Table 1. As discussed in 4.1, training for 100 epochs with the default parameters and configuration of the YOLOv7 model yielded moderate results, with relatively low objectness and classification losses. Extending the training significantly improved performance, suggesting that longer training durations are essential for achieving good object detection.

A larger training dataset could have mitigated the overfitting issue. Additionally, hyperparameter tuning did not enhance the model's performance; in fact, it negatively impacted it, as shown in Figure 4b. However, this could be due to the insufficient number of epochs, as it is somewhat expected that performance might initially worsen when tuning model parameters before stabilizing with extended training. The reduction in image size also caused a noticeable drop in the model's performance, as shown in Figure 4c, highlighting the importance of higher resolution for YOLO models.

The YOLOv7 tiny models performed poorly across the board. Interestingly, the model trained on unaltered images performed slightly better, suggesting that the use of GMM and optical flow in the pre-processing pipeline might have interfered with YOLO's ability to detect and classify objects effectively. This conclusion is reinforced by the even poorer performance when edge detection was added to the pipeline.

Surprisingly, our short YOLOv10 training run showed promising potential, though the training time was too brief to draw definitive conclusions.

6. Conclusion

Unfortunately, the results of our training and experiments fell short of our expectations. Apart from the initial YOLOv7 baseline model, which already struggled with detection tasks, all subsequent models performed poorly. It is challenging to pinpoint exactly where things went wrong, but time constraints likely prevented us from fully training and optimizing the models. Although some experiments showed potential, even with extended training, the model metrics improved too slowly to produce a viable solution in the time we had.

We also believe that the size of the dataset played a significant role. As illustrated in Figure 3, the number of instances for different fish species varies considerably. Even with a more balanced distribution, the challenges of underwater object detection remain substantial, as fish pat-

YOLO configurations	Epochs	Objectness Loss	Classification Loss	Precision	Recall	mAP@0.5
v7 default	100	0.008	0.019	0.697	0.290	0.288
v7 default	1000	0.003	0.003	0.801	0.589	0.666
v7 tuned	100	0.002	0.014	0.094	0.018	0.001
v7 240px	100	0.000	0.011	0.000	0.000	0.000
v7 tiny orginal	300	0.009	0.077	0.256	0.010	0.001
v7 tiny	350	0.004	0.044	0.089	0.007	0.000
v7 tiny default	300	0.009	0.076	0.337	0.008	0.000
v7 tiny default	350	0.013	0.060	0.337	0.064	0.003
v7 tiny tuned	440	0.000	0.009	0.000	0.003	0.000
v7 tiny 240px	550	0.004	0.036	0.339	0.007	0.002
v7 tiny edge	550	0.004	0.036	0.000	0.000	0.000
v10 default	32	-	-	0.359	0.025	0.008

Table 1: The training results across the different YOLO models and configurations we tested reveal some notable findings. Most significant is the dramatic drop in performance when edge detection was introduced, as we had anticipated. In this context, *default* refers to the model's performance without any hyperparameter tuning. *Tuned* indicates that we adjusted the hyperparameters as described earlier. *240px* signifies the reduction in image resolution from 640 to 240 pixels. *Original* refers to models trained on images without any pre-processing. Finally, *edge* corresponds to models incorporating edge detection in the pre-processing pipeline.

terns have evolved to make them less visible to predators. In hindsight, we should have opted for a broader range of datasets instead of focusing on a single one and attempting to optimize it to improve performance.

Our tests indicated that using GMM for background separation and optical flow for motion capture may have hindered performance more than it helped. However, a more thorough investigation would be needed to draw a definitive conclusion. We believe that incorporating a pre-processing step to identify fish-shaped objects might have enhanced YOLO's performance by reducing image complexity, although it could also have introduced new challenges.

Future work should focus on utilizing the latest version of YOLO, as our brief training sessions showed promising results. Additionally, reducing the size of the neural network and incorporating training with various datasets would undoubtedly enhance the overall robustness of the method.

References

[1] Joly A., Goeau H., Glotin H., Spampinato C., Bonnet P., Vellinga W.-P., Planquè R., Rauber A., Palazzo S., Fisher R., et al. Lifeclef 2015: multimedia life species identification challenges. *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 462–483, 2015. 1, 3

[2] Jayme Garcia Arnal Barbedo. A review on the use of computer vision and artificial intelligence for fish recognition, monitoring, and management. *Fishes*, 2022. 2

[3] Mahdi Hamzaoui, Mohamed Ould-Elhassen Aoueileyine, Lamia Romdhani, and Ridha Bouallègue. An improved deep learning model for underwater species recognition in aquaculture. *Fishes*, 2023. 1

[4] Euan Sinclair Harvey, Shortis, Stuart Robson, Mathew Stadler, and Michael Cappo. A system for stereo-video measurement of sub-tidal organisms: Implications for assessments of reef fish stocks. 1995. 1

[5] Ahsan Jalal, Ahmad Salman, Ajmal Mian, Mark Shortis, and Faisal Shafait. Fish detection and species classification in underwater environments using deep learning with temporal information. *Ecological Informatics*, 57:101088, 2020. 2

[6] Yan Li, Xinying Bai, and Chunlei Xia. An improved yolov5 based on triplet attention and prediction head optimization for marine organism detection on underwater mobile platforms. *Journal of Marine Science and Engineering*, 2022. 1

[7] Zhaoxuan Lu, Xiaolong Zhu, Haitao Guo, Xingang Xie, Xiangzi Chen, and Xiangqian Quan. Fishfocusnet: An improved method based on yolov8 for underwater tropical fish identification. *IET Image Processing*, 2024. 1

[8] Roboflow. Aquarium dataset, 2020. 1

[9] Ahmad Salman, Shoaib Ahmad Siddiqui, Faisal Shafait, Ajmal Mian, Mark R Shortis, Khawar Khurshid, Adrian Ulges, and Ulrich Schwanecke. Automatic fish detection in underwater videos by a deep neural network-based hybrid motion learning system. *ICES Journal of Marine Science*, 77(4):1295–1307, 02 2019. 1

[10] Abdelouahid Ben Tamou, Abdesslam Benzinou, and Kamal Nasreddine. Multi-stream fish detection in unconstrained underwater videos by the fusion of two convolutional neural network detectors. *Applied Intelligence*, 51:5809 – 5821, 2021. 2

[11] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 2