Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Bachelor Thesis in Information Engineering

Computer Vision-based detection and tracking of fish in aquaculture environments

Bachelor Candidate

Giovanni Zebele

Student ID 1195783

Supervisor

Prof. Damiano Varagnolo

University of Padova

Graduation day

14th November 2022

Academic Year
2021/2022

To Alba and to my Mum and Dad, who more than anybody else, shaped me as a person.
Because on this day I want to celebrate and be proud of the person that I am.

Abstract

This thesis analyses some methods and algorithms for the detection and tracking of fish in submerged fish farming cages. This work is part of a more extensive project on the automation of fish farming that is being implemented in collaboration with Sintef Ocean AS and the Norwegian University of Science and Technology (NTNU). Overall, the work aims at assessing how Computer Vision (CV) techniques may aid evaluating how much and under which conditions the noise and appearance of robots within a farming cage provokes fear and aggressiveness in the fish population.

The main focus of this thesis is then assessing the performance of detecting fish using "You Only Look Once" (YOLO), a single-stage object detector based on a Convolutional Neural Network (CNN). Different models are obtained by training the detector on different portions of the data and their performance is compared and combined in a cross-validation approach to estimate how accurately the model will perform on real-life scenarios, i.e., on unseen data. Finally, the thesis presents a first approach to the tracking of fish using a StrongSORT tracker based on the YOLO models that were previously trained, and discusses the obtained results. All detector models were trained on custom datasets, with the data kindly being provided by the aforementioned research partners.

Sommario

Questa tesi analizza alcuni metodi e algoritmi per la rilevazione e il tracciamento di pesci in gabbie sottomarine adibite alla pescicoltura. Questo lavoro è parte di un progetto più esteso incentrato sull'automazione della pescicoltura e realizzato in collaborazione con Sintef Ocean AS e l'Università di Scienze e Tecnologia della Norvegia (NTNU). Nel complesso, il lavoro intende se le tecniche di Computer Vision possono aiutare nella valutazione di quanto e in quali condizioni il rumore e l'aspetto di robot introdotti nelle gabbie provocano paura e aggressività nella popolazione di pesci.

La tesi si concentra principalmente sul valutare la qualità del rilevamento dei pesci tramite l'ultilizzo di YOLO ("You Only Look Once" - "Guarda una sola volta"), un rilevatore monostadio basato su una Rete Neurale Convoluzionale (CNN). Tramite il training del rilevatore su diversi partizionamenti dei dati si sono ottenuti vari modelli predittivi, le cui perormance sono state confrontate e combinate con un approccio cross-validation per effettuare una stima dell'accuratezza del modello in uno scenario realistico, ovvero su dati completamene sconosciuti. Infine la tesi presenterà un primo approccio al tracciamento dei pesci usando un tracciatore StrongSORT basato sui modelli di YOLO precedentemente ottenuti, e verranno discussi i risultati. Tutti i modelli sono stati "addestrati" su dataset personalizzati, con le immagini che sono state gentilmente fornite dai suddetti partner di ricerca.

Contents

List of Figures **xi**

List of Tables **xi**

List of Code Snippets **xi**

List of Acronyms **xiii**

1 Introduction **1**

1.1 Motivation 1

1.2 Objectives 2

2 Object Detection Theory **3**

2.1 Definitions and Performance Metrics 3

2.1.1 Assessing the quality of the classification 3

2.2 YOLOv7 5

2.2.1 Operational insight 5

2.2.2 Limitations 5

3 Data **7**

3.1 Creating the dataset 7

3.2 Available data and referencing 8

3.3 Labelling 8

3.4 Training, validation, and test data sets 9

3.4.1 Training set 9

3.4.2 Validation set 10

3.4.3 Test set 10

3.5 Organising the dataset 10

3.6 Considerations 10

4 Methodology **13**

4.1 Cross-validation 13

4.2 Computing platform 14

4.3 Detection 14

4.3.1 Preliminary operations 14
4.3.2 Training YOLO 15
4.3.3 Training output 16
4.3.4 Testing 18
4.4 Inference 18
4.5 Tracking 19

5 Results & Discussion **21**
5.1 Detection results 21
5.1.1 Training results 21
5.1.2 Testing results 22
5.2 Inference & Tracking results 23

6 Conclusion and Future Work **27**
6.1 Conclusion 27
6.2 Future Work 28

References **29**

Acknowledgments **31**

List of Figures

3.1 Labelling example 7
3.2 A frame from the actual labelling process using the annotator 9
3.3 Tree representation of the dataset folder structure 11

4.1 Plot of mAP and mAP@[.5,.95] vs iteration number. Source: Weights & Biases 17
4.2 Plot of objectness loss in training and validation datasets vs iteration number. Source: Weights & Biases 17

5.1 Plot of mAP@[.5,.95] vs iteration number for all 9 training runs. The legend reports for each plot, the model number as defined in table 4.1 and the training and validation sets used. Source: Weights & Biases 21
5.2 Tracking/inference example n.1. Detection is performed by model n.2 on the upper half and by model n.8 on the lower half. 24
5.3 Tracking/inference example n.2. Detection is performed by model n.2 on the upper half and by model n.8 on the lower half. 25

List of Tables

3.1 Content of the label text files 9

4.1 Designed data set arrangement for cross-validation[1] 13

5.1 Test results including the mean and the standard deviation of the precision scores 22

List of Code Snippets

4.1 Python script to organize images and labels into the desired folders. This code creates in particular dataset configuration number one from table 4.1 14

List of Acronyms

NTNU	Norwegian University of Science and Technology
YOLO	"You Only Look Once"
CNN	Convolutional Neural Network
CV	Computer Vision
AI	Artificial Intelligence
ML	Machine Learning
OD	Object Detection
IoU	Intersection over Union. See 2.1.1
NMS	Non-Maximal Suppression
TP	True Positives
FP	False Positives
TN	True Negatives
FN	false Negatives
AP	Average Precision
mAP	mean Average Precision
R-CNN	Region-based Convolutional Neural Network
FPS	Frames Per Second
COCO	Commo Objects in Context
ROV	Remotely Operated Vehicle

1 Introduction

1.1 Motivation

Aquaculture is an essential contributor to the production of seafood for human consumption. In 2020, the global production of aquatic animals amounted to 178 million tonnes, of which more than 157 million tonnes (89%) were consumed by humans. Aquaculture contributed with an output of 88 million tonnes of fish, 49% of the total, with an overall first sale value estimated at USD 265 billion (FAO, 2022). These numbers show the absolute relevance of aquaculture in the balance of global food production, both economically and quantitatively. However, with the population and standards of living constantly growing in many countries, the demand for the global fishing industry will continue to increase.

As has already happened in almost every industry, adopting solutions based on the constant evolution of technology is an essential step to reach new standards of productivity and sustainability. As an example, current fish farms rely primarily on manual operations and close human interactions with the breeding process and the fish cage structures, but this may prove to be problematic in situations such as exposed areas where the environment can be inaccessible to humans. Therefore, it is necessary to explore new technologies that can help reduce the need for human interactions.

CV has a tremendous potential both in production and consumer environments, given its long history, extensive research and literature, and especially the prominence that Artificial Intelligence (AI) and Machine Learning (ML) have gained in the last decade. The most tangible example of how CV impacts our lives is self-driving cars, but it can be used for the most diverse purposes. Kasper-Eulaers et al. used YOLO to determine the number of parking spaces available in rest areas to facilitate the planning of rest periods for road transport companies, while Hou and Pan applied the same object detector for a more creative task: quantifying the aesthetic score in hotel images from a hotel booking website with the purpose of determining a correlation between the aesthetics of photos and the tendency of consumers to post reviews and overall ratings to the hotel.

But CV has already found its way in fish farming as well. New implementations of machine vision techniques have been commercialised to perform various tasks such as fish counting, size measurement and mass estimation, species and stock identification, and welfare monitoring(Zion, 2012). Public awareness and the potential economic impact of stressed animals have drawn particular attention to the latter. “When fish are stressed, they undergo various metabolic changes, all of which are expressed externally by variations in their behavior. Similarly, a change in fish feeding behavior, swimming behavior or skin color is a sign of unfavorable conditions, stress, distress or pathogenic conditions” (Zion, 2012).

1.2 Objectives

NTNU and Sintef, partners of this research, have already carried out work that goes in the direction of improving the welfare of farmed fish. More specifically, previous projects focused on the use of CV to track feeding pellets in underwater cages and assess the quality of feeding and estimate feed waste (Thorset and Stray, 2021). The scope of the present study is to approach the problem of using CV to automatically estimate the impact of remotely controlled activities carried out by autonomous vehicles within fish cages. This has been achieved with the use of YOLO, a state-of-the-art Object Detection (OD) framework, and a YOLO-based StrongSORT tracker. Due to the limited time available to study the problem at hand, this thesis provides a partial solution, mainly taking into consideration the best approach to the detection part, and suggesting a solution for the tracking. A more functional employment of tracked data and full use of the 3D data that was provided by Sintef, in order to evaluate and track 3-dimensional movement of the fish are listed as future work.

2

Object Detection Theory

OD consists in distinguishing different objects in an image or a video and provides bounding boxes that identify where every detected object is. As most AI applications, OD is heavily based on data analysis, more specifically pattern recognition and matching. ML methods leverage data to improve performance on a set of tasks. The main goal of any learner is to use their own experience to generalise skills and knowledge to a new context, and for a learning machine, this translates into gaining the ability to perform accurately on new, unseen tasks or data after having experienced a learning data set. It is clear that ML provides a feasible solution to the task of detecting objects in any image or video, after training a model on data that have been first reviewed by humans.

YOLOv7, a model from the YOLO network family, will be used to solve the detection problem in this thesis. This chapter will go through the main performance metrics and terminology associated with ML (and more specifically OD) to allow for an easier understanding of the results discussed in the next chapters. Furthermore, it will give a brief explanation of the operation of the chosen detector.

2.1 Definitions and Performance Metrics

OD must be distinguished from image classification, which assigns specific concepts to an image or video, giving a certain understanding of what is shown, and segmentation, which instead labels each pixel in an image with certain concepts to define the object exactly, pixel by pixel.

2.1.1 Assessing the quality of the classification

Listed below are some metrics that recur when evaluating the output of a detector (Zakoldaev and Alisa A, 2021):

- True Positives (TP): the number of correctly classified objects of the relevant class (i.e.

the class that we are interested in). E.g., in fish cage footage, an actual fish that is classified as a fish.

- True Negatives (TN): the number of correctly classified objects of an irrelevant class, e.g., a detection that is not a fish, but is also not classified as such.
- False Positives (FP): the number of objects of an irrelevant class defined as objects of a relevant class, e.g., a part of the background or dirt particles that are detected as fish.
- false Negatives (FN): the number of objects of the relevant class, defined as objects of the irrelevant class, e.g., a fish that is not classified as such.
- Ground truth: information that is known to be real or true. In object detection and CV in general ground truths are the actual occurrences of a specific class in an image, that is, the union of TP and FN.
- Intersection over Union (IoU): the ratio between the area where a predicted bounding box and the corresponding ground truth overlap and the area of the union of the bounding box and the ground truth.

$$\text{IoU} = \frac{\text{Bbox} \cap \text{GroudTruth}}{\text{Bbox} \cup \text{GroudTruth}}$$

Based on these values, the following metrics are calculated to assess the effectiveness and correctness of the detection:

- Precision: proportion of predicted positives that are actually correct, i.e. true positives over total detections.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- Recall: proportion of actual positives that are correctly predicted, i.e. true positives over the total number of ground truths.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Precision and recall make it possible to draw the precision-recall curve, which is found by evaluating precision and recall throughout a set of predictions for a given class and then sorting it in descending order by prediction confidence. Hui provided a guided example of the calculation of the precision-recall curve. The area under the precision-recall curve represents the class-wise Average Precision (AP). One of the most popular metrics is mean Average Precision (mAP), which is obtained as the mean of Average Precision over all classes. A further variation of this indicator is mAP@[.5,.95]. This metric differs from the first because it takes into account IoU thresholds ranging from 0.5 to 0.95, with a stride of 0.05, while mAP only considers detections with an IoU threshold of 0.5.

2.2 YOLOv7

2.2.1 Operational insight

Detectors usually work in one or two stages, with YOLOv7 being the first type. In a single-stage detector, the image is evaluated in steps with different resolutions as a result of convolution and pooling layers to extract information about objects of differently sized objects. The backbone of the network is usually a pre-trained classifier such as VGG16, Resnet or other high-performance CNNs. Unlike two-stage detectors, which break down the problem statement into detecting possible object regions and classifying the predicted regions into object classes, YOLO proposes the use of an end-to-end neural network that creates bounding boxes and evaluates class probabilities all at once.

The YOLO algorithm works by dividing the image into N grids, all with the same size of SxS. Each grid is responsible for detecting and locating a single object, the one it contains. Correspondingly, these grids predict B boundary boxes, each associated with a 4-tuple (x,y,w,h) and a box confidence score. The confidence score reflects how likely the box contains an object (objectness) and how accurate the boundary box is. The values w and h define the size of the box, while x and y are offsets to the corresponding cell. Each cell has a number C of conditional class probabilities (the probability that the detected object belongs to a particular class). It follows that predictions have a shape of (S,S,Bx5+C) (Hui, 2018b). This process reduces computational complexity by delegating both detection and recognition to the cells, but contextually brings about a lot of duplicate predictions since multiple cells generate different bounding boxes for the same object. YOLO deals with this issue by suppressing all bounding boxes that have confidence scores below a set threshold . This technique is called Non-Maximal Suppression (NMS). At first, the algorithm looks at the probability score associated with each decision and chooses the largest. Then it suppresses the bounding boxes that have the largest IoU with the current high probability bounding box. (Bandyopadhyay, 2022)

2.2.2 Limitations

Even though "YOLOv7 surpasses all known object detectors in both speed and accuracy in the range from 5 FPS to 160 FPS and has the highest accuracy 56.8% AP among all known real-time object detectors" (Wang, Bochkovskiy, and Liao, 2022), it comes with several limitations. The one-object-per-cell rule limits how close detected objects can be. YOLO struggles to detect and segregate small objects in images that appear in groups, as each grid is constrained to detect only a single object. Objects that naturally come in groups, such as a line of ants or a school of fish in this study case, where fish are very close to each other, are difficult for YOLO to detect and localise.

YOLO is also characterised by lower accuracy compared to much slower OD algorithms such as Fast Region-based Convolutional Neural Network (R-CNN), but the higher performance of the latter would not justify such an increase in time and computational power required to complete the task. All things considered, YOLOv7 has been deemed more suitable to achieve the objective of the project.

3 Data

3.1 Creating the dataset

For training detectors that rely on deep learning such as YOLOv7, large amounts of diverse data are required. The data, provided by Sintef, were obtained by salmon farms and consist of images taken from a salmon pen. The network is trained in a supervised manner, which requires the training set to have labels to obtain feedback on the correctness of the predictions (Jin, 2022). Thus, data have to be reviewed by humans in a process called labelling that consists of manually defining the ground truths in a set of images by drawing bounding boxes that identify the fish.

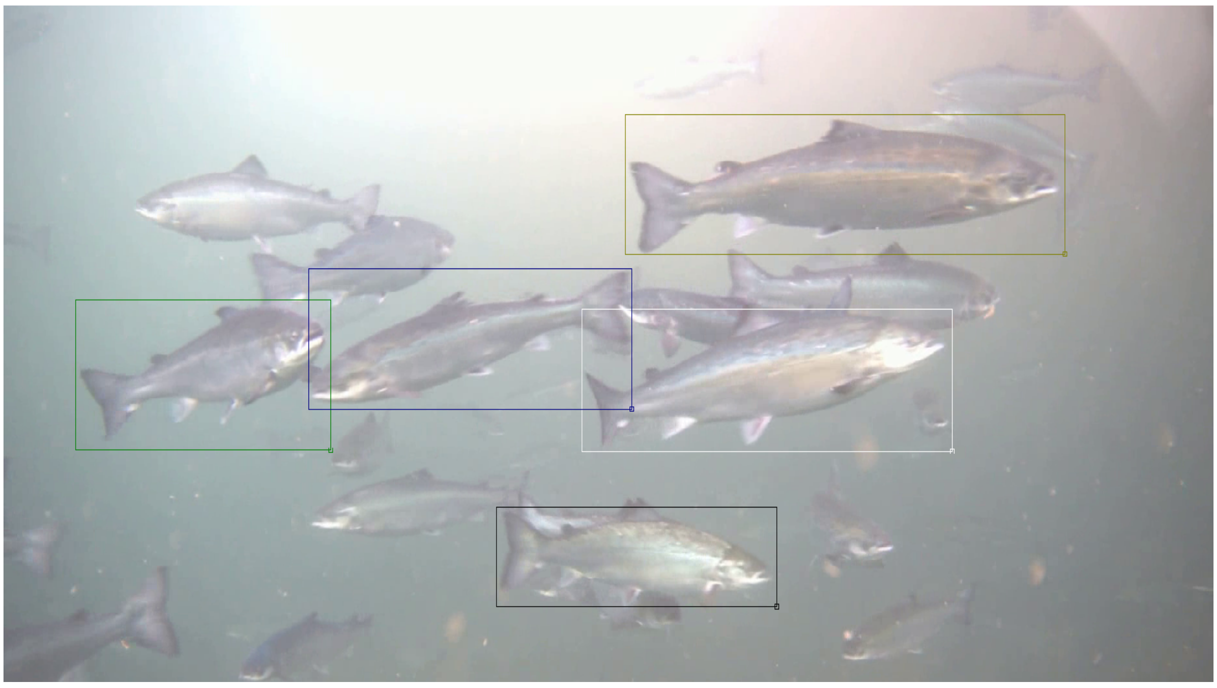

Figure 3.1: Labelling example

3.2 Available data and referencing

The custom data set used for training, validation and testing was composed of three videos, which will be later referenced as follows:

Video A: `114309 Sc M9 8Meter High-High No2.m4v`

Video B: `115409 Sc M9 8Meter Low-0 No2.m4v`

Video C: `150609 Sc M9 8Meter High-0 No6.m4v`

Videos A and B present similar characteristics, showing fish more or less at the same distance from the camera during the entire running time, while video C features more variability, with the school of fish being very close to the camera at times, but also with several frames with no fish to label. This is an important aspect, because images with no labels help the network recognise the background, thus improving generalisation of the model. Additionally, video C also presented a generally higher density of fish.

3.3 Labelling

In our case, the images were extracted from the videos provided by Sintef and labelled using a Python script provided by Stian Jakobsen, my peer at NTNU. What the script does is it creates a list of the videos in the specified video folder and display to the user how many frames of each video have already been labelled. The user can then choose which video to label and whether or not to start labelling from the last labelled frame. Subsequently, the script opens a new window with the two images associated with the frame to be labelled (3.2). There were two because the data was from Stian's project, which aims at obtaining detection and tracking on 3-dimensional videos. The annotator extracts only 2% of the total number of frames, which for the 25 Frames Per Second (FPS) videos in our dataset, translates to one frame every 2 seconds. Labelling every single frame would have been pointless, given that differences between frames that are temporally very close are almost non-existent.

It is quite noticeable in figure 3.1 that the bounding boxes drawn around the entire body of the fish can be dispersive, since they are made up for a good part of the background or parts of other fish. Accordingly, Stian and I agreed not to label the entire fish, but only the head or tail, to provide the detector with smaller but more precise ground truths. Eventually, the tail was preferred because it was deemed to have a more distinctive shape that could work better with the detector (see 3.2). Every object detector requires that the labelled data be provided in a specific way. For YOLO, the labelling format is pretty simple: for every image in the set, a corresponding `txt` file of the same name must be created. The text file contains in each line the information about the object class and the bounding box coordinates, as outlined in table 3.1.

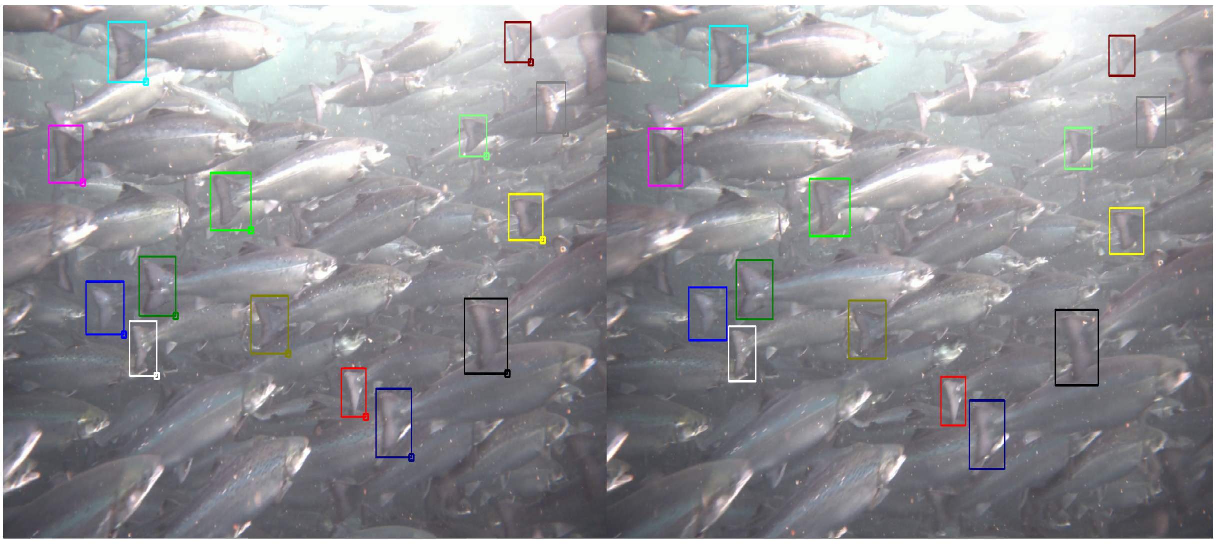

Figure 3.2: A frame from the actual labelling process using the annotator

class	x-coordinate	y-coordinate	width	height
0	0.09322916666666667	0.3175	0.028125	0.0575

Table 3.1: Content of the label text files

The first number indicates which class the object belongs to; in our case, there is only one class of objects (fish). The rest of the numbers are the coordinates of the centre of the bounding box and its dimensions. Note that each line in the file contains only the numerical values separated by a space.

3.4 Training, validation, and test data sets

ML algorithms learn to make predictions by building a mathematical model from input data, which are usually divided into multiple data sets. In particular, three data sets are commonly used in different stages of the creation of the model: training, validation, and test sets.

3.4.1 Training set

Training in CV is the process of manipulating the weights of a (deep) neural network to map an input to an output. The goal of training is to produce a trained (fitted) model that generalises well to new, unknown data. Training is performed by presenting the algorithm the labelled data, to find patterns and empirical relationships that it is going to use to make predictions on new data. This approach has a natural tendency to overfit the training data, meaning that they can identify and exploit apparent relationships in the training data that do not hold in general. The training process is iterated after getting feedback of the correctness of predictions made on validation data. Iterations are referred to as epochs.

Models can be trained from scratch, meaning that they can be taught to recognise objects without having experienced any data. In our case, as is usually done, training starts from a pre-trained model. YOLOv7 provides 6 different models that are pre-trained on the Commo Objects in Context (COCO) dataset, a large-scale OD dataset, which labels 80 classes of different common real-life objects on more than 200000 images. This is an obvious choice, especially having a limited dataset.

3.4.2 Validation set

Since our goal is to find the network that has the best performance on new data, the simplest approach to comparing different networks is to evaluate the performance on data that are independent of those used for training. After each training iteration, YOLO assesses the performance of the last model by testing it on an independent validation set, allowing it to choose the best model that was obtained so far and at the same time adjusting the hyperparameters (see 4.3.2). Basically, the candidate models are successive epochs of the same network, and the best epoch yields the final model.

3.4.3 Test set

Since validation can itself lead to some overfitting of the validation set, the performance of the selected network should be confirmed by measuring its performance on a third set of data that the model has never been fed. If a model fit to the training dataset also performs well with the test dataset, minimal overfitting has taken place.

3.5 Organising the dataset

To reduce the risk of problems such as overfitting and to maximise the performance of the model, validation and test datasets should not be used to train the model. Thus, before every train run the dataset had to be organised properly. See figure 3.3 on the next page for a schematisation of the folder structure used to organise the data. Note that the folders `images` and `labels` both contain the subfolder `left`, because, as already explained in 3.3, the annotator labels stereo images, thus saving both right and left images and labels.

3.6 Considerations

The labelling process was extremely tedious, mechanical, and time consuming. It was particularly so in our case, since data to be labelled were stereoscopic, non-rectified videos, which means that there were two images per frame, and after setting an annotation on the left image, the corresponding bounding box on the right image required positional adjusting to precisely fit the tail.

Another challenge found in the process was deciding which fish to label and which to ignore. As can be seen in figure 3.2 there are several fish whose tails were not labelled, even though they are well discernible. This was an arbitrary choice made with the purpose of optimising the detection of fish that swim in the foreground, and minimising spurious detections.

Overall, our final dataset consisted of 950 labelled images. For good detection results, it is generally advised that the dataset should contain at least 2000 images, but, as shown in the next chapters, the amount of data available was enough to yield good detection and tracking results.

```
stereo_dataset
├─ images
│  └─ left
│     ├─ test
│     │  ├─ left_test_1.png
│     │  └─ ...
│     ├─ train
│     │  ├─ left_train_1.png
│     │  └─ ...
│     └─ val
│        ├─ left_val_1.png
│        └─ ...
└─ labels
   └─ left
      ├─ test
      │  ├─ left_test_1.txt
      │  └─ ...
      ├─ train
      │  ├─ left_train_1.txt
      │  └─ ...
      └─ val
         ├─ left_val_1.txt
         └─ ...
```

Figure 3.3: Tree representation of the dataset folder structure

4 Methodology

4.1 Cross-validation

A cross-validation approach was taken in this study to estimate the performance of the detector on real-life settings (i.e. on completely new data) and to select the best model for inference on new videos. Cross-validation is a resampling method that uses different portions of the dataset to test and train a model on different iterations. In our implementation, the reshuffling of the dataset was not done between iterations, instead, YOLO was trained a total of nine times, rearranging the dataset before every training run. Dataset reorganisation was done by creating the train, val and test sets with different configurations of images and labels form the three videos (or a mix of them). This allowed to get models trained on data that show more variability, as well as situations with more specific images, thus representing more real-life situations, where data availability and variability is not always granted. Table 4.1 displays which videos were used for the three sets in each training run.

Training run	**Training set**	**Validation Set**	**Test set**
1	Video A/B	Video A/B	Video C
2	Video B/C	Video B/C	Video A
3	Video A/C	Video A/C	Video B
4	Video A	Video B	Video C
5	Video B	Video C	Video A
6	Video C	Video A	Video B
7	Video B	Video A	Video C
8	Video C	Video B	Video A
9	Video A	Video C	Video B

Table 4.1: Designed data set arrangement for cross-validation[1]

[1]The notation 'Video X/Y' means that frames from videos X and Y were put together and randomly selected to put into the specified folder

4.2 Computing platform

For any neural network, the training phase of the deep learning model is the most resource-intensive task. Adjusting the weights of the network requires going through large amounts of data, which in practice translates to a huge number of arithmetic operations. Powerful GPUs are required to complete the training in a reasonable amount of time, in fact they are optimised for parallel calculations. Without availability of the kind of computational power required for the task, the obvious solution was recurring to cloud computing platforms, which provide access to powerful GPUs, both for free (in limited quality and quantity) or for a fee. At first training attempts were done on Google COlab, Google's cloud computing environment, but the final choice was a service called Gradient by the company Paperspace. After cloning YOLO's GitHub repository and creating the folder structure as described by figure 3.3, the environment could be set up for training.

4.3 Detection

4.3.1 Preliminary operations

Before starting a new training run, all the files (both images and labels) were moved from the `test`, `train` and `val` folders back to the parent folder `left`, and then split again in the desired configuration. Both stages were performed by a Python script that was specifically designed for the task. By running different portions of the code in a Python Notebook, it was possible to execute the operations separately. Additional code (not reported) was created in order to check the number of files in the different folders.

```python
import glob
import shutil
from sklearn.model_selection import train_test_split

### Function that creates a list of filenames with a name pattern by searching in the labels
    folder (only the filename, not the path and the extension)

def find_filename_pattern(filename_no_extension):
    filename_lab = 'stereo_dataset/labels/left/' + filename_no_extension + '?.txt'
    lab_list = glob.glob(filename_lab)
    for i in range(4):
        filename_lab = filename_lab.split('?', 1)[0] + '??' + filename_lab.split('?', 1)[1]
        lab_list.extend(glob.glob(filename_lab))
    lab_list = [lab.replace(lab_path, '').replace('.txt', '') for lab in lab_list]
    return lab_list


###Creating filename lists, each associated to a video, then organizing them into train, val
    and test file lists and finally moving them to the correct folders

img_path = 'stereo_dataset/images/left/'
lab_path = 'stereo_dataset/labels/left/'
filenames = ['left_114309 Sc M9 8Meter High-High No2_', 'left_115409 Sc M9 8Meter Low-0 No2_',
     'left_150609 Sc M9 8Meter High-0 No6_']
video_A = find_filename_pattern(filenames[0])
```

```python
video_B = find_filename_pattern(filenames[1])
video_C = find_filename_pattern(filenames[2])

#Organising files into train, val, test lists - to be changed between training runs
#This code creates the configuration: Videos A and B are mixed and frames randomly separated
    between the trai and val sets. Video C used as test set
trainval = video_A + video_B
train_files, val_files = train_test_split(trainval, test_size=0.2)
test_files = video_C

#Moving train images and labels into appropriate folders
for f in train_files:
    shutil.move(img_path + f + '.png', img_path + 'train/')
    shutil.move(lab_path + f + '.txt', lab_path + 'train/')

#Moving validation images and labels into appropriate folders
for f in val_files:
    shutil.move(img_path + f + '.png', img_path + 'val/')
    shutil.move(lab_path + f + '.txt', lab_path + 'val/')

#Moving test images and labels into appropriate folders
for f in test_files:
    shutil.move(img_path + f + '.png', img_path + 'test/')
    shutil.move(lab_path + f + '.txt', lab_path + 'test/')
```

Code 4.1: Python script to organize images and labels into the desired folders. This code creates in particular dataset configuration number one from table 4.1

After that, it was necessary to define the `data.yaml` file in the dataset folder. It is a simple text file that contains the path to the image folders for the different stages, train, val and test, a well as information about the number and the name of classes of objects to be detected (in our case, just the 'fish' class).

In order to perform training, a terminal connected to the machine provided by Gradient was created. After changing the working directory to the YOLO main folder, it was possible to install all the Python Packages required by YOLO. This was done by submitting the line

```
pip install -r reqirements.txt
```

Lastly, the package for the Weights & Biases (Wandb) service was installed. Wandb logs the entire set of information associated with every training run. This includes performance metrics and bounding box predictions (both on training and validation data) at the end of every epoch, output to the terminal and records of the use of system resources.

4.3.2 Training YOLO

At this point, the training of the detector could be started. This was done by running the `train.py` script inside the main YOLO folder, with the following command:

```
python train.py --workers 8 --device 0 --batch 4 --img 1280 --data stereo_dataset/data.yaml --
    cfg cfg/training/yolov7.yaml --weights '' --name yolov7 --hyp data/hyp.scratch.custom.yaml
     --epoch 200
```

Below is a brief explanation of the options used in the example:

`--workers` - Maximum number of dataload workers. These are the subprocesses that run in parallel during training.

`--device` - Specifies the device(s) to use for training. 0 indicates a single GPU.

`--batch` - Determines the number of samples processed before an update of the model is created.

`--img` - The resolution for the images to be resized to. The bigger the size of the images that the network works with, the better the model should be, since bigger images allow more details to be provided to the algorithm.

`--data` - Path to the yaml file mentioned in 4.3.1

`--cfg` - Path to the yaml configuration file. This file defines the architecture of the network. The only edit needed to this file was to determine the number of classes.

`--weights` - The weights file which represents the starting point of the training. The two apostrophes indicate the default model which is `yolov7.pt`. See 3.4.1.

`--name` - Specifies the name of the folder where the training results will be saved.

`--hyp` - Path to the hyperparameters file. It contains a list of parameters that control the whole learning process of the model. In this project the default hyperparameters defined in `hyp.scratch.custom.yaml` were used.

`--epoch` - Determines the number of iterations that the training has to be performed over.

The duration of the training runs varied from 2 to 4½ hours, depending on the available machine and the size of the train set.

4.3.3 Training output

The training output we are interested in consists in the `best.pt` file. It contains the weights of the neural network model that performed best on validation data. The best performance is determined by a fitness function that is defined as a weighted sum of precision, recall, mAP and mAP@[.5,.95]. The default weights were used, which are [0.0, 0.0, 0.1, 0.9], thus the best epoch is determined almost entirely by the mAP@[.5,.95] score. The number of epochs was limited to 200 because, as we can see in figure 4.1, both mAP and mAP@[.5,.95] stop increasing around the 150th iteration, and training the model for longer would not have produced a better model. Indeed, in this training run the best weights were saved at epoch 185.

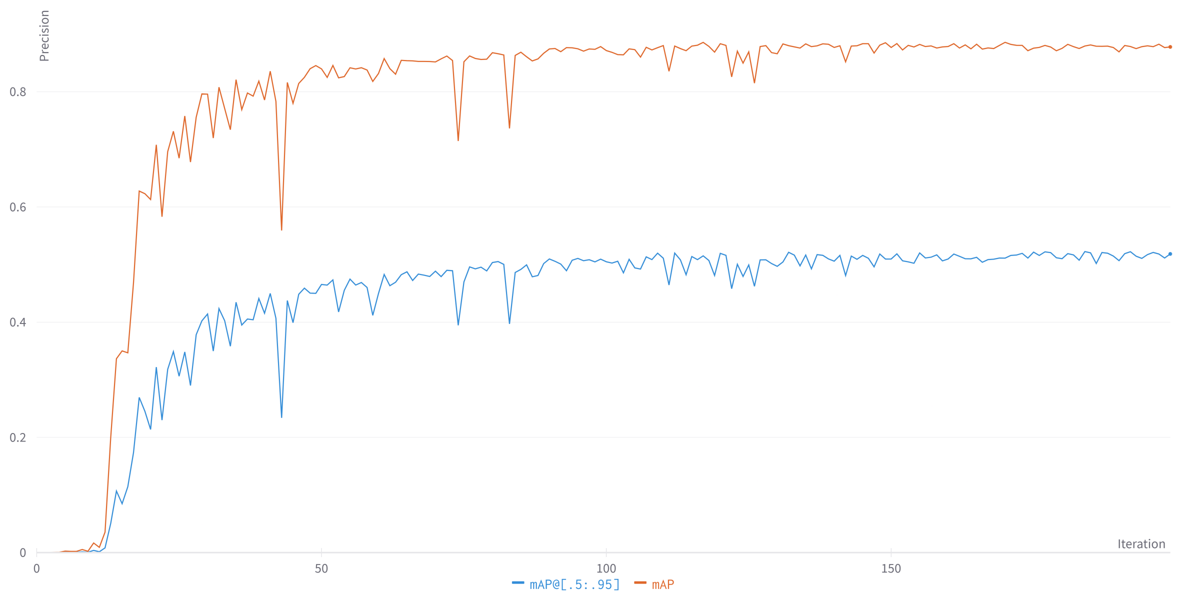

Figure 4.1: Plot of mAP and mAP@[.5,.95] vs iteration number. Source: Weights & Biases

Furthermore, it should be taken into account that training for too many epochs increases the risk of overfitting. In figure 4.2 below, we can see how the objectness loss[2] on the validation set increases slightly as the training proceeds. This is a sign of the model overfitting the training dataset, but the effect is acceptable given that the scale is small.

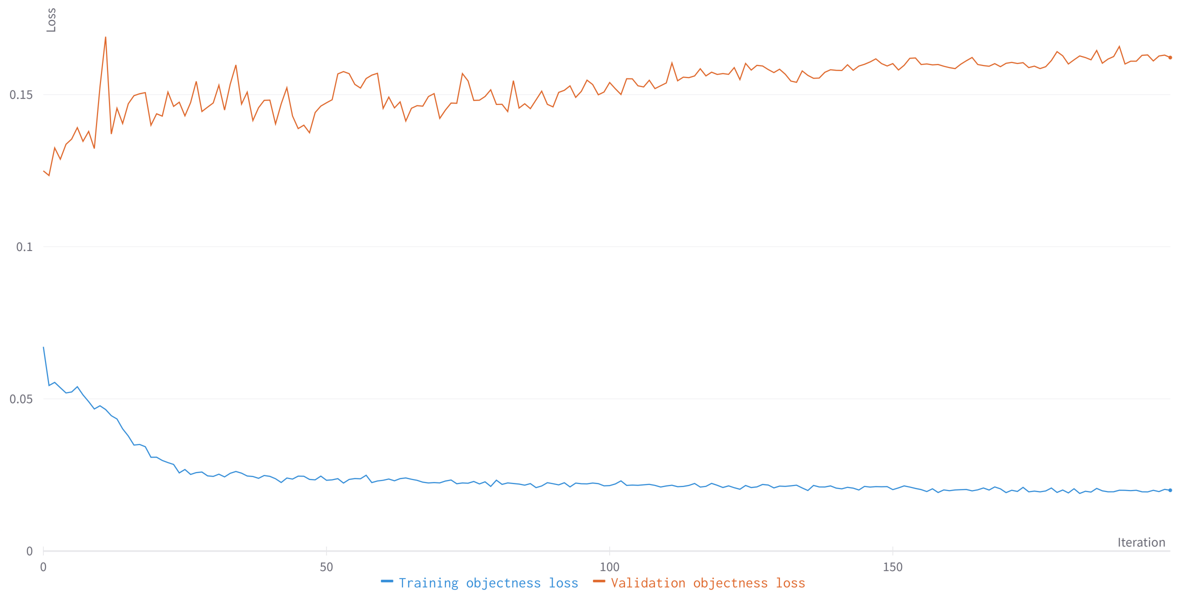

Figure 4.2: Plot of objectness loss in training and validation datasets vs iteration number. Source: Weights & Biases

[2]Objectness loss is a value that represents the error made by the detector in predicting that an object is there

4.3.4 Testing

Once trained, a model is ready for testing, which is performed by running the `train.py` script inside the main YOLO folder, with the following command:

```
python test.py --data stereo_dataset/data.yaml --task test --img 1280 --batch 32 --conf 0.001
    --iou 0.65 --device 0 --weights runs/train/yolov713/weights/best.pt --name yolov7_A_B_C
```

Many of the options are the same found for the training script 4.3.2, below are listed those that weren't already explained:

`--task` - This option instructs the script to look for the test path in the yaml file. By default, `test.py` scans the validation set.

`--conf` - Confidence threshold for making predictions. Such a low value implies that every prediction will be calssified as an object.

`--batch` - Determines the IoU threshold to be used for NMS (see 2.2.1).

`--weights` - For the test script this parameter provides the path to the weights file of the model to be tested. It will reference the best weights file for a given training.

`--name` - Name of the folder where the test results should be saved.

The test process creates as outputs different files (confusion matrix, f1-score curve, precision-recall curve, etc.) but what we are most interested in are the performance metrics on the test dataset, which are displayed as follows:

```
Class       Images      Labels      P           R           mAP@.5      mAP@.5:.95
all         132         1092        0.683       0.709       0.726       0.374
```

These values represent an example of an output. In the next chapter test results for every trained model will be gathered and discussed.

4.4 Inference

Inference is the process of feeding the trained model with new, unlabelled data that can be either a video, an image or a stream. The detector makes predictions about the objects to be detected and draws bounding boxes around them. Inference is started by running the script `detect.py`:

```
python detect.py --weights runs/train/yolov710/weights/best.pt --conf 0.25 --img-size 1280 --
    source stereo_dataset/Videos/'left_150609 Sc M9 8Meter High-0 No6.m4v'
```

The parameters are a subset of those listed for testing, except for the `-- source` option, which specifies the file (or the stream source) to do detection on.

In this dissertation, inference was never performed by itself, but as explained in the next section, inference and tracking were run simultaneously.

4.5 TRACKING

Tracking can be defined as the problem of estimating the trajectory of an object in the image plane as it moves around a scene. In other words, a tracker assigns consistent labels to the tracked objects in different frames of a video. Strongsort, the tracking algorithm that was chosen for this project works by obtaining possible object regions in every frame by means of an object detection algorithm, and then the tracker corresponds objects across frames (Yilmaz, Javed, and Shah, 2006). It follows that inference and tracking are performed at the same time. StrongSORT uses YOLO as object detector, which makes the implementation easier since the detector models were already trained and ready to be used.

After cloning the StrongSORT GitHub repository, the tracker was run with the command:

```
python track_v7.py --source yolov7/stereo_dataset/left_mix.m4v --yolo-weights yolov7/runs/
    train/yolov718/weights/best.pt --strong-sort-weights weights/osnet_x0_25_msmt17.pt --img-
    size 1280 --device 0 --save-txt --save-vid --count --draw --conf-thres 0.5
```

Some of the options are the same used to run the train and test scripts (see 4.3.2 and 4.3.4). The other options are:

`--yolo-weights` - Path to the file containing the YOLO weights used for the detection part.

`--strong-sort-weights` - Path to the file containing the StrongSORT weights used for the tracking part. Default StrongSORT weights were used in this implementation.

`--save-txt` - Saves the results in a text file containing all the information about the predicted classes and bounding boxes, similarly to what shown in table 3.1.

`--save-vid` - Saves inference and tracking results to a video file.

`--count` - Instructs the script to count the number of detections for each object class. The counter is displayed in the output video. In the original code all the object occurrences are counted from the start of the video, but this was edited to display the number of detections only in the current frame to enable an easier comparison of different tracking runs.

`--draw` - Instructs the script to draw lines that follow the trajectory of every labeled detection.

`--conf-thres` - Confidence threshold for creating bounding boxes. The tracker will not draw bounding boxes for predictions with confidence lower than this threshold.

5

Results & Discussion

5.1 Detection results

Overall, 9 training runs were completed and for each model obtained, testing was carried out on the corresponding dataset.

5.1.1 Training results

W&B enables to plot the evolution of the performance metrics for different training runs on the same chart. By analysing the mAP@[.5,.95] graph reported in figure 5.1 in particular, where differences within model precision are much more obvious, it is possible to make some important observations. Model number referencing can be found in figure 5.1.

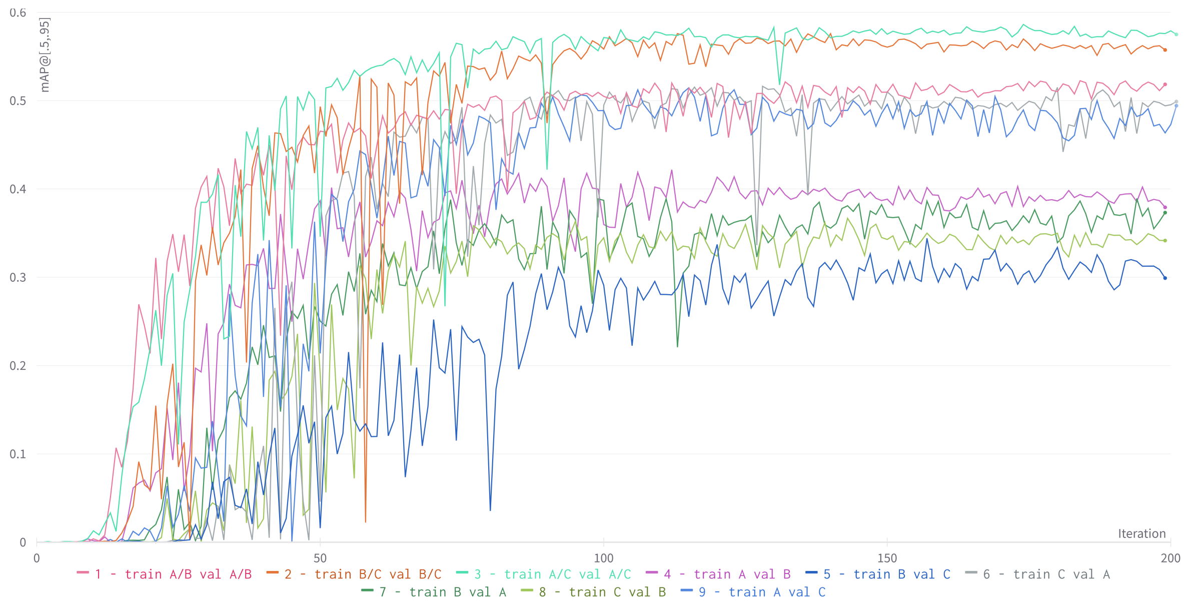

Figure 5.1: Plot of mAP@[.5,.95] vs iteration number for all 9 training runs. The legend reports for each plot, the model number as defined in table 4.1 and the training and validation sets used. Source: Weights & Biases

- The models which delivered the best validation results (n.1, 2 and 3) were those that were trained on a mixed set. In particular models n.2 and 3, which had video C mixed in the training set, performed slightly better than model n.3 that did not.

- Models n.6 and 9, which were trained on video C and validated on video A (and vice versa) performed much better than all the other models that were trained on data from a single video. In fact, their performance was comparable to that of model n.1.

- The models that performed worse were n.8 and 5. Given that from previous observations models seemed to be performing better when they were trained or validated on video C, this result was surprising. On the contrary, a lower precision was expected from models n.4 and 8 since video A and video B present similarities, as already stated in Section 3.2.

This analysis would suggest that a good general rule to obtain a robust detector that generalises well would be to include video C in the training or validation set, but testing results discussed in the next section do not prove this hypothesis.

5.1.2 Testing results

Model	**Train. set**	**Val. set**	**Test set**	**mAP@.5**	**mAP@.5:.95**
1	Video A/B	Video A/B	Video C	0.793	0.489
2	Video B/C	Video B/C	Video A	0.847	0.436
3	Video A/C	Video A/C	Video B	0.816	0.382
4	Video A	Video B	Video C	0.794	0.503
5	Video B	Video C	Video A	0.824	0.387
6	Video C	Video A	Video B	0.812	0.344
7	Video B	Video A	Video C	0.662	0.316
8	Video C	Video B	Video A	0.844	0.52
9	Video A	Video C	Video B	0.837	0.396
Mean				0.803	0.419
Standard deviation				0.0565	0.0721

Table 5.1: Test results including the mean and the standard deviation of the precision scores

Testing results do not show a precise pattern, though some comments can be made:

- By looking at mAP, the models that are performing better are the ones trained/validated on videos B-C, and A-C (n.2, 5, 8, and 9), which suggests that the third video should be used for training. Note that model n. 8, which had one of the worst performance during training, produced the second best mAP score.

- On the contrary, the models (n.1, 4, 7) that were trained and validated on videos A and B (and the other way around) seem to perform the worst, confirming that detection video C is "hard" for a model trained on the first two, where fish are never very close.

- The mAP@.5:.95 metric appeared to be rather inexplicable given that some models provided a high mAP and a much lower mAP@.5:.95 while for some others it was the contrary.

Recalling that the cross-validation approach was taken in order to estimate the performance of the detector in a non-optimistic situation, the mean and the standard deviation of the scores have to be discussed. With regard to the mAP score, the results are satisfying. In fact the predicted value will fall with a probability of 95% in the range [0.690; 0.916], which is high enough for OD.

The expected value of the mAP@.5:.95 which falls with probability 95% in the interval [0.275; 0.563], should be higher but is acceptable.

5.2 Inference & Tracking results

Since inference and tracking were performed jointly by the tracker, the final empirical observations on the correctness and completeness of the detections had to be made on the output of the tracker. For this final stage, the two models that performed best during testing were selected. Given the volatility of the mAP@.5:.95 score, this decision was made by considering the mAP score only, thus model n.2 and 8 were picked.

The data used in this process was completely new and consisted of a montage of short video clips taken from the last batch of data provided by Sintef. The videos were trimmed to extract the most interesting bits (those with more fish swimming in the foreground) and then merged together into a video using ffmpeg. The data was then fed to the tracker by running the script twice with the options illustrated in 4.5. The only difference was the `--yolo-weights` that was set each time to the proper model path. The two output videos were eventually stacked on top of each other to enable performance to be compared in real time. Below are reported two screenshots from the final video along with considerations on inference and tracking performance. The lines drawn over the video keep track of the trajectory of every predicted object and allow an easier visualisation of the tracking performance.

Figure 5.2 displays a situation where both the detector and the tracker are working well. The setting is "easy" since there are just a few fish, all of which are far from one another, which makes it less complex for the tracker to distinguish one fish from another. The detector is working really well, as it is able to detect even the fish on the left whose tail is barely visible. Given that during the labelling process we chose not to put ground truths on such spurious instances, this is an indication that the detector is robust and can generalise well. Furthermore, the closest fish is correctly not detected, since it is clearly not a salmon, thus we are not interested in tracking it.

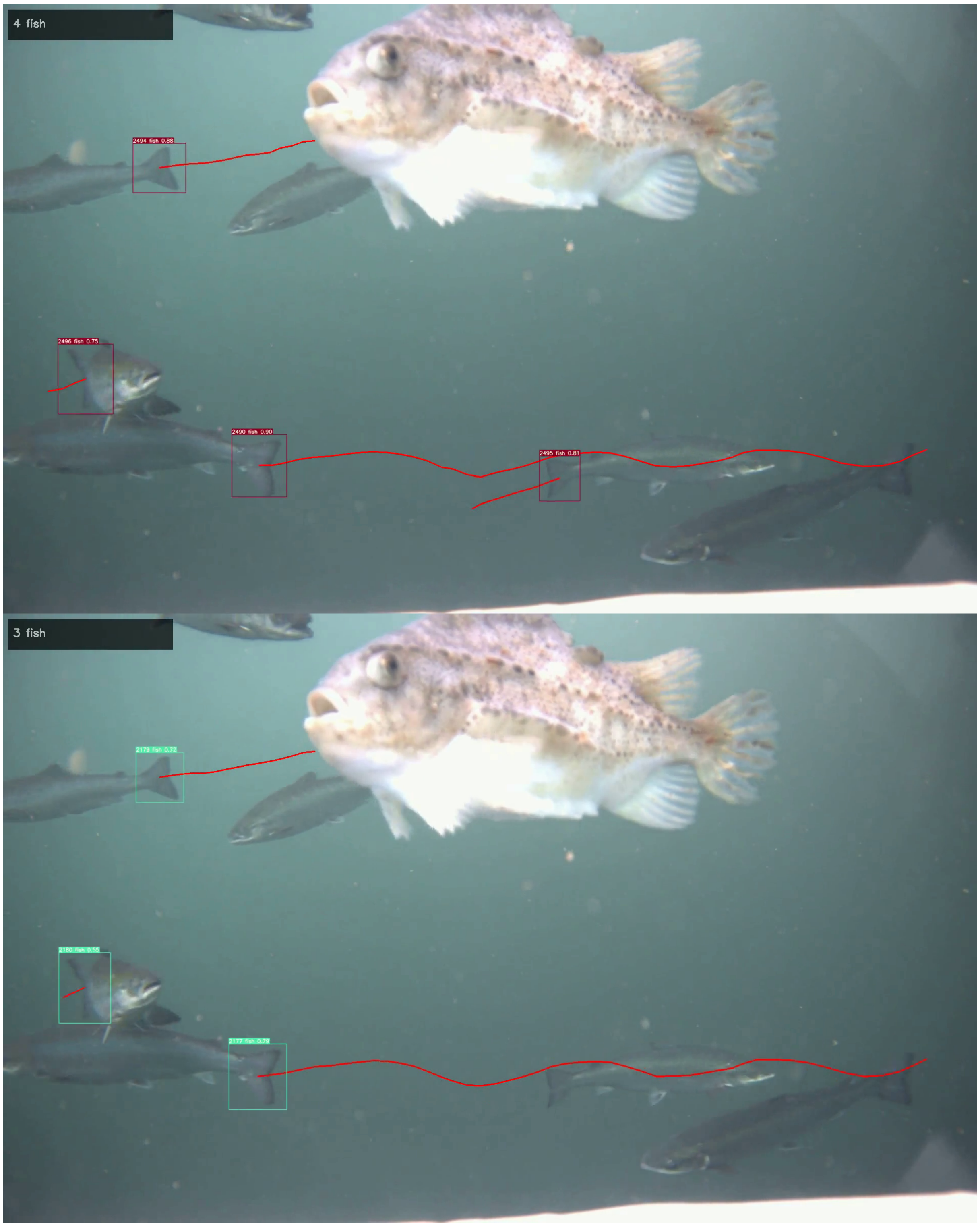

Figure 5.2: Tracking/inference example n.1. Detection is performed by model n.2 on the upper half and by model n.8 on the lower half.

The frame displayed in figure 5.3 presents a rather complicated situation for the tracker. The detector is still working well, especially the one on the upper picture (model n.2), in fact the closest fish are all correctly detected, except one tail that is detected twice. The tracker, though, is having trouble identifying correctly each detection and telling one fish from another. This leads to trajectories of different fish being mixed together, with a lot of random lines being drawn. What makes it so difficult for the tracker to work properly is mainly the fact that fish are swimming all in different directions, intersecting their trajectories.

Figure 5.3 also allows us to compare the detection performance of models n. 2 and 8. The ability of the first model to detect fish is slightly better. There are several frames where model n.8 is able to detect more fish, but in the vast majority of cases the opposite happens. In this case, less detections imply less leads for the tracker to follow, thus less wrong trajectories.

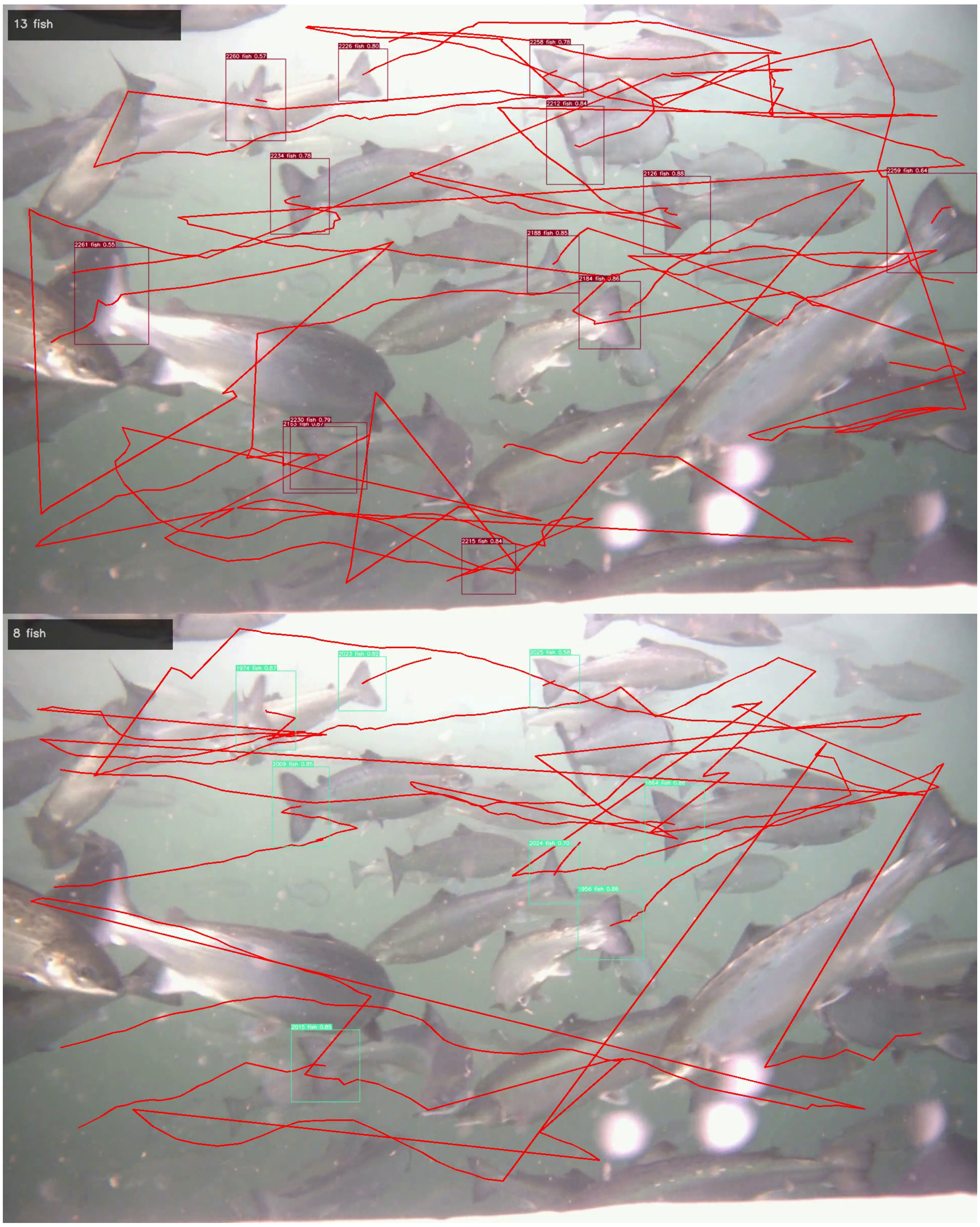

Figure 5.3: Tracking/inference example n.2. Detection is performed by model n.2 on the upper half and by model n.8 on the lower half.

To summarise, empirical observation of the tracked footage revealed that the detector works very well on almost all situations where fish are not too far from the camera, but the tracking algorithm only works well when there is a limited number of fish inside the camera's field of view and they are all swimming in the same direction.

6 Conclusion and Future Work

6.1 Conclusion

This thesis aimed at exploring the feasibility of using Computer Vision techniques and deep learning algorithms to detect and track fish in aquaculture cages, with the final purpose being the use of 3D camera footage to measure fish-population parameters, such as distance and swim speed. This values can be recorded to be later used for the statistical analysis on fish behavior. The final goal of the project carried out by NTNU and Sintef, where this work fits in, is to automatically assess how the presence of a Remotely Operated Vehicle (ROV) inside a fish cage produces changes in such behavior. The main contribution of this thesis is to explore a robust method for detecting and tracking fish by using YOLOv7 for the detection part and a YOLOv7-based algorithm for the tracking. Despite the limited dataset, results prove YOLO to be a strong and powerful detector that, after training on a dataset which offers enough variability, ensures precise and correct detection on new data.

YOLOv7 was trained on a number of images ranging from 292 to 526, depending on the dataset configuration. This was enough to establish precise bounding boxes around the tails of the fish, but with varying confidence. This can only be said for the two models that yielded the best test results, since they were the only ones used for tracking and inference. The other models were expected to perform worse.

It should be noted that generally the suggested dataset split ratio within training, validation and test sets are respectively 70%, 20% and 10%, which was not the case in this dissertation. The reason for this is that we wanted to represent a non-optimal situation to get a final performance estimation that was realistic, rather than optimistic. Besides, having such separation within the training, validation and testing sets yielded further insights on the impact of dataset composition and variability on model performance.

Overall, YOLO has proved to be detecting fish consistently and has been therefore deemed a good choice for a tracking algorithm to be based on. StrongSORT, on the other hand, did not produce satisfying results. The main issue is the inconsistency of the tail-label associations, which causes a lot of random lines to be drawn on the output video.

6.2 Future Work

It should be taken into consideration that tracking fish tails is a difficult task, given the lack of distinctive features between different instances, which makes it difficult for the tracker to label each tail unambiguously. Future work should be directed primarily to improving the tracking performance. Different approaches can be taken to complete this task:

- Further exploration of the tracker's architecture and tuning of the hyper-parameters. Different pre-trained models[1] of the tracker should be tested as well.
- Experimenting with other tracking algorithms.
- Changing labelling technique. Teaching the detector to recognise fish by labelling the entire body or just the head, which may display more specific traits, might produce better tracking results. This should be kept as a last option, given that it would require re-labelling of every image in the dataset. Furthermore, as noted in 3.3, labelling the entire body could be counter-productive and result in worse detection performance.

More effort should be directed to getting a higher confidence score for the YOLO predictions as well. One necessary operation that is definitely going to improve this aspect is expanding the dataset by labelling more footage, but as for the tracker, tuning the hyper-parameters can provide better performance also from the detector.

[1]Models available at `https://kaiyangzhou.github.io/deep-person-reid/MODEL_ZOO`

References

FAO (June 2022). *The State of World Fisheries and Aquaculture 2022*, pp. 1–1. DOI: 10.4060/CC0461EN.

Kasper-Eulaers, Margrit et al. (2021). “Short communication: Detecting heavy goods vehicles in rest areas in winter conditions using YOLOv5”. In: *Algorithms* 14 (4). ISSN: 19994893. DOI: 10.3390/a14040114.

Hou, Lei and Xue Pan (Feb. 2022). “Aesthetics of hotel photos and its impact on consumer engagement: A computer vision approach”. In: ISSN: 0261-5177. DOI: 10.1016/J.TOURMAN.2022.104653.

Zion, Boaz (Oct. 2012). “The use of computer vision technologies in aquaculture A review”. In: *Computers and Electronics in Agriculture* 88, pp. 125–132. ISSN: 0168-1699. DOI: 10.1016/J.COMPAG.2012.07.010.

Thorset, Jonas and Eivind Heldal Stray (2021). “Detection and Tracking of Feeding Pellets in Salmon Pens”.

Zakoldaev, Danil A. and Vorobeva Alisa A (2021). “Machine Learning Methods Performance Evaluation”. In: *Turkish Journal of Computer and Mathematics Education* 12 (2).

Hui, Jonathan (Mar. 2018a). *mAP (mean Average Precision) for Object Detection | by Jonathan Hui | Medium*. URL: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173.

— (Mar. 2018b). *Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3*. URL: https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088.

Bandyopadhyay, Hmrishav (Oct. 2022). *YOLO: Real-Time Object Detection Explained*. URL: https://www.v7labs.com/blog/yolo-object-detection.

Wang, Chien-Yao, Alexey Bochkovskiy, and Hong-Yuan Mark Liao (2022). *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*.

Jin, Kristoffer Chi Rong (2022). “Deep-learning algorithms for estimation of fish-population parameters from video data”.

Yilmaz, Alper, Omar Javed, and Mubarak Shah (Dec. 2006). “Object tracking”. In: *ACM Computing Surveys (CSUR)* 38 (4), p. 45. ISSN: 03600300. DOI: 10.1145/1177352.1177355. URL: https://dlnext.acm.org/doi/10.1145/1177352.1177355.

Acknowledgments

A special thanks goes to Stian, my peer at NTNU, who provided me with basically everything I needed to start working on this project and answered to all my questions.

Thank you to my supervisor Damiano.