



Norwegian University  
of Life Sciences

**Master's Thesis 2023 30 ECTS**  
Faculty of Science and Technology

# **Fish tracking using detection in Aquaculture: A Pilot Study**

Jens Kristian Røed Holmboe  
Data Science



# Preface

This thesis marks my end here at the Norwegian University of Life Science (NMBU) as a master's student. Therefore I would like to thank PhD. Lars Erik Solberg and PhD. Santhosh Kumaran for introducing me to the topic, and Chris Nobel for providing us with data from the Research Council of Norway projects: FASTWELL. It has been an exciting topic, regardless of how challenging it has been at times, and I would like to return to working with computer vision in the future.

There are many people I would like to thank for helping me through writing my thesis. I am truly grateful for all the knowledge, time and supervision my supervisor Professor Oliver Tomic and my co-supervisors, Lars Erik Solberg and Santhosh Kumaran provided me during this thesis. Their discussions and guidance and has motivated me and contributed greatly. Also, a special thanks to my fellow student Mikal Breiteig. Our engaging discussions have been enlightening, and his camaraderie has made this journey even more rewarding.

I would also like to thank my family for being super supportive and taking such a deep interest in my studies, and doing their best to understand what I have been working on. A special thanks go to my brother Jonas who could always put a smile on my face, and my grandpa for checking up on me every week.

Finally, I would like to thank all my fellow students for making my five-year here at NMBU a truly memorable time of my life, leaving me with unforgettable memories I will fondly remember as I move forward in life.



# Abstract

This research addresses the pressing need for robust fish-tracking systems in the aquaculture industry. Recognising the significant role of the fish industry in global food security and sustainable development, our study investigates the potential of these advanced technologies in overcoming challenges in fish welfare monitoring.

Our focus is on developing a robust computer vision-based fish tracking system capable of handling the challenges of underwater fish tracking. Leveraging state-of-the-art detection frameworks, Detectron2 and YOLOv8, and pairing them with the high-performance object tracking algorithm, ByteTrack, to create a foundation for future advancements in fish welfare. This work is motivated by a project by Nofima, which aspires to improve fish welfare through sustained fish tracking. Utilising their provided dataset, we train and refine our tracking system. Our primary success criteria are high-confidence fish detection within the frame and reliable tracking despite challenges such as occlusion.

Our findings indicate promising detection performance, especially YOLOv8, which surpassed Detectron2 across various mean average precision (mAP) thresholds. When combined with ByteTrack, YOLOv8 outperformed the Detectron2-ByteTrack combination in tracking performance.

The combination of YOLOv8 and ByteTrack demonstrates the best performance among the tested models, marking an advancement in fish tracking systems. However, while the results show promise, they also signal a need for further refinement to overcome the challenges inherent in underwater fish tracking fully. This study represents a crucial step towards developing advanced and accurate fish tracking systems, paving the way for improved welfare monitoring and management in the aquaculture industry.

# Contents

<b>1</b>	<b>Background</b>	<b>8</b>
1.1	Introduction . . . . .	8
1.2	Objectives . . . . .	9
1.3	Structure . . . . .	10
<b>2</b>	<b>Theory</b>	<b>11</b>
2.1	Machine Learning Theory . . . . .	11
2.2	Deep Learning with Artificial Neural Networks . . . . .	12
2.2.1	Convolutional Neural Networks . . . . .	17
2.3	Computer vision . . . . .	20
2.3.1	Object detection . . . . .	20
2.3.2	Detection using Detectron2 . . . . .	21
2.3.3	Detection using YOLOv8 . . . . .	23
2.3.4	Object tracking . . . . .	26
2.3.5	Bytetrack . . . . .	28
2.3.6	Evaluation Metrics . . . . .	29
2.3.7	Intersection over Union . . . . .	30
<b>3</b>	<b>Method</b>	<b>35</b>
3.1	Software and Hardware . . . . .	35
3.2	Dataset . . . . .	35
3.2.1	Video annotation and Preprocessing . . . . .	37
3.3	Detection using Detectron2 . . . . .	39
3.4	Detection using YOLOv8 . . . . .	40
3.5	Tracking using ByteTrack . . . . .	40
<b>4</b>	<b>Results and Discussion</b>	<b>44</b>
4.1	Detectron2 and Bytetrack Performance . . . . .	44
4.1.1	Detection Results . . . . .	44
4.1.2	Tracking Results . . . . .	48
4.2	YOLOv8 and ByteTrack Performance . . . . .	50
4.2.1	Detection Results . . . . .	50
4.2.2	Tracking Results . . . . .	52
4.2.3	Comparative analysis . . . . .	56
<b>5</b>	<b>Evaluation and Future Directions</b>	<b>61</b>
<b>6</b>	<b>Conclusion</b>	<b>63</b>

---

<b>A</b>	<b>Tables of Detection and Tracking Dependencies</b>	<b>68</b>
A.1	Detectron2 Dependencies . . . . .	68
A.2	YOLOv8 Dependencies . . . . .	69
A.3	ByteTrack Dependencies . . . . .	70
<b>B</b>	<b>Parameters</b>	<b>71</b>
B.1	Detectron2 parameters . . . . .	71
B.2	YOLOv8 parameters . . . . .	72
B.3	ByteTrack parameters . . . . .	73

# List of Figures

2.1	A visual representation of the artificial intelligence hierarchy, showcasing AI as an overarching field with its various subcategories, including machine learning, deep learning and computer vision. . . . .	11
2.2	The Functionality of a Node in ANN . . . . .	13
2.3	Depicts the corresponding functions and plots for each activation function ReLU, Sigmoid, and Tanh. . . . .	13
2.4	An example of a basic MLP with an input layer with three input units and an additional bias unit. There are two hidden layers, the first with three hidden units and a bias unit, and the second with four hidden units and its own bias unit. The network concludes with two output layers. . . . .	14
2.5	Illustration of backpropagation of the error in an ANN, computing the partial derivatives of the loss with respect to the output layer weights. $L$ represents the loss function, $a^{out}$ denotes the activation function for the neuron in the output layer, and $w_{1,1}^{out}$ refers to its corresponding weights. . . . .	15
2.6	Illustration of backpropagation of the error in an ANN, computing the partial derivatives of the loss with respect to the hidden layer weights. $L$ represents the loss function, $a^h$ denotes the activation function for the neurons in the hidden layer, while $a^{out}$ is for the output layer. $w^{h,i,j}$ refers to the corresponding weights between neuron $i$ in the current hidden layer and neuron $j$ in the next layer, here being $w^{h,1,1}$ . The corresponding weights for the output layer can be denoted as $w_{1,1}^{out}$ . . . . .	16
2.7	A typical Convolutional Neural Networks . . . . .	17
2.8	Application of a Filter Across a 2D Image in a Convolutional Operation . . . . .	18
2.9	Illustration of the max-pooling operation . . . . .	19
2.10	A comparison between image classification and object detection tasks. Image classification (left) assigns a single label to the entire image, while object detection (right) identifies and locates multiple objects within the image, using bounding boxes to represent their positions, scales, and classes. Original images from[24][25]	21
2.11	This illustration demonstrates the architecture of Detectron2, highlighting the interconnected components, including the Backbone, RPN, and ROI Head. The figure showcases how these elements work together within the object detection pipeline to process and analyze input images . . . . .	22
2.12	A visual representation of the YOLOv8 architecture, as interpreted by a community member named RangKing[33]. This illustration provides insight into the model's design and structure, given the absence of an official publication from the Ultralytics team. . . . .	24



---

2.13	An illustration of IoU. Here, the Area of Intersection denotes the area where the two bounding boxes overlap, and the Area of Union represents the combined area of the two bounding boxes. . . . .	30
2.14	The illustration demonstrates how various detections fall into distinct categories. In this case, $\alpha$ represents the IoU threshold of 0.5, with ground truth (GT) bounding boxes marked in red and predicted bounding boxes in blue. Each detection has its own IoU score, which determines its placement into one of the three categories: True Positive (TP), False Positive (FP), or False Negative (FN) based on the IoU score. . . . .	31
3.1	The variability in fish appearance from different angles in the Training and Test datasets . . . . .	36
3.2	Examples of challenges in the underwater environment . . . . .	36
3.3	Annotation scenarios . . . . .	37
3.4	Depiction of the same frame before and after annotation. Fishes in the annotated frame are depicted with a red bounding box surrounding them, occluded fish are surrounded by a dotted red bounding box . . . . .	38
3.5	The preprocessing pipeline to split and convert the annotated dataset into the final dataset . . . . .	39
3.6	ByteTrack tracking pipeline . . . . .	41
3.7	Sample frame from ByteTrack's final output video: Each identified fish is encompassed by a distinct bounding box, accompanied by a tracking ID and detection confidence. . . . .	42
3.8	Illustrates the process of matching ground truth annotations (indicated in red) with the predictions (indicated in blue) to enable the computation of MOT Metrics for the video sequence . . . . .	43
4.1	Detectron2's training performance . . . . .	45
4.2	Comparison of detected frames from the ch6 validation, ch6 test, and ch3 test datasets. . . . .	47
4.3	Examples of severe occlusions in the datasets . . . . .	48
4.4	channel 3 Prolong occlusion lasting 60 frames (almost 2.5 seconds) . . . . .	49
4.5	channel 3 Brief occlusion lasting 5 frames (20 milliseconds) . . . . .	49
4.6	YOLOv8 training performance . . . . .	51
4.7	Comparison example using frame 1 of ground truth and detections in Ch3 test datasets. . . . .	52
4.8	Example of Prolong occlusion lasting 60 frames (almost 2.5 seconds) . . . . .	53
4.9	Example of brief occlusion lasting 5 frames (20 milliseconds) . . . . .	53
4.10	Detection comparison between two threshold parameters that decide high scoring and low scoring detections . . . . .	55
4.11	Prolong occlusion with lost track restoration at 80 frames . . . . .	55
4.12	Detection results from Detectron2 and YOLOv8 compared to the ground truth of the same frame . . . . .	57
4.13	ByteTrack with YOLOv8 on Ch3 . . . . .	59
4.14	ByteTrack with Detectron2 on Ch3 . . . . .	59
4.15	Tracking Detection compared with ground truth on Frame 92 Ch3 . . . . .	60

# List of Tables

3.1	GPU specs for Google Colab Pro . . . . .	35
4.1	Mean Average Precision (mAP) for different IOU thresholds in training and testing for Detectron2 . . . . .	45
4.2	Performance metrics for object tracking using default Bytetrack with Detectron2 as the detector. MOTA (Multiple Object Tracking Accuracy), MOTP (Multiple Object Tracking Precision), IDF1 (ID F1 Score), MT (Mostly Tracked), PT (Partially Tracked), and ML (Mostly Lost). . . . .	48
4.3	Comparison of tracking performance metrics for ByteTrack with Detectron2 on Channels 3 and 6, using various parameter configurations. The results demonstrate the impact of tuning parameters such as threshold (th) for high-scoring and low-scoring detections, Intersection over Union (IoU), and maximum frame gap for lost tracks (frame) on the tracking performance. . . . .	50
4.4	Mean Average Precision (mAP) for different IOU thresholds in training and testing for YOLOv8 . . . . .	51
4.5	Number of Ground Truth and Predicted Fish Detected by YOLOv8 in Frame 1 in Ch3 . . . . .	52
4.6	Performance metrics for object tracking using default ByteTrack . . . . .	52
4.7	Comparison of tracking performance metrics for ByteTrack with YOLOv8 on Channels 3 and 6, using various parameter configurations. The results demonstrate the impact of tuning parameters such as threshold (th) for high-scoring and low-scoring detections, Intersection over Union (IoU), and maximum frame gap for lost tracks (frame) on the tracking performance. . . . .	54
4.8	Average detection in Ch3 . . . . .	54
4.9	Mean Average Precision (mAP) for different IOU thresholds in training and testing for Detectron2 and YOLOv8 . . . . .	56
4.10	Average detection results and mAP scores from Detectron2 and YOLOv8 for the two datasets compared with the ground truth . . . . .	57
4.11	Performance metrics for object tracking using YOLOv8 and Detectron2 with default ByteTracker on Ch3 and Ch6 test datasets . . . . .	58
5.1	Comparative Analysis of ByteTrack and DeepSort Trackers Using the Same Detection Model (YOLOv8) . . . . .	62
A.1	Detectron2 dependencies with their respective versions and purpose of use . . . .	68
A.2	YOLOv8 dependencies with their respective versions and purpose of use . . . .	69
A.3	ByteTrack dependencies with their respective versions and purpose of use . . . .	70
B.1	Parameters and their respective values used in Detectron2 . . . . .	71

---

B.2 Parameters and their respective values used in YOLOv8 . . . . .	72
B.3 Parameters and their respective values used in Byte Tracker . . . . .	73

# Background

## 1.1 Introduction

The fish industry plays a crucial role in global food security, providing essential nutrients and protein sources to billions of people worldwide. Being estimated to be a 401 billion USD industry in 2018. Over the past few decades, the rapid expansion of the aquaculture sector has helped meet the rising demand for fish, accounting for over 50% of fish production for human consumption as the Global fish production was estimated to be about 179 million tons in 2018, 156 million tonnes of it was used for human consumption[1]. Despite its significant contributions to the global food supply, the fish industry faces growing concerns regarding fish welfare. The potential negative consequences of intensive farming practices damaging fish health through overcrowded environments lead to disease spread. Ensuring the well-being of fish in aquaculture systems is not only an ethical imperative but also a key factor in achieving optimal production outcomes and securing the long-term sustainability of the industry.

The United Nations' Sustainable Development Goal 14 (SDG 14)[2][3], "Life Below Water," underscores the importance of conserving and sustainably using oceans, seas, and marine resources for sustainable development. Promoting responsible fish farming practices, including the improvement of fish welfare, is an integral part of this global commitment. By safeguarding the well-being of aquatic species, we can advance the broader objectives of SDG 14, such as maintaining the health and productivity of marine ecosystems, reducing marine pollution, and fostering sustainable fisheries.

Recent advances in computer vision technologies offer promising opportunities to revolutionize the fish welfare industry[4][5]. These cutting-edge technologies can be employed to monitor fish behaviour, identify signs of stress, and assess health conditions in real time, thereby enabling the implementation of interventions to mitigate welfare issues and enhance the overall well-being of fish populations. Adopting computer vision technologies can help manage fish welfare by providing accurate and reliable data on individual fish and group dynamics. This allows for the automation of detecting potential welfare concerns, reducing human error in monitoring, and facilitating more efficient resource allocation in managing fish health and welfare. Furthermore, the application of these technologies not only promotes the ethical treatment of aquatic species but also aligns with the aims of Sustainable Development Goal (SDG) 14 by supporting sustainable and environmentally-friendly aquaculture practices.

However, complex environmental factors create significant challenges for existing computer vision technologies, impeding precise monitoring in underwater fish tracking. These factors include occlusion, where fish overlap or are hidden behind objects; variable lighting conditions

---

due to depth, time of day, and weather; diverse fish behaviours such as rapid swimming, sudden changes in direction, and schooling; water turbidity affecting image clarity; and camera motion caused by water currents or external forces. Collectively, these factors contribute to the complexity of underwater fish tracking and present considerable challenges for developing and applying computer vision technologies in aquaculture, necessitating innovative solutions to improve the accuracy and reliability of fish monitoring.

This thesis aims to provide a solution using computer vision technologies, particularly deep learning algorithms, and their potential applications in enhancing fish welfare within the aquaculture industry. Central to this exploration is the investigation of how our computer vision models can lay the groundwork for accurate fish detection and monitoring by combatting challenges, especially occlusion, which serves as a critical prerequisite for implementing welfare-focused applications. By developing innovative solutions to address these difficulties, our computer vision models could facilitate a more precise understanding of fish behaviour in the future, leading to improved welfare outcomes in aquaculture settings.

In light of the challenges and opportunities presented by computer vision technologies for fish welfare monitoring, the primary focus of this thesis is to develop and evaluate a robust computer vision-based fish tracking system that can effectively address the complexities of underwater fish tracking in aquaculture environments. By leveraging state-of-the-art detection frameworks and object-tracking algorithms, we aim to create a foundational tracker that can lay the groundwork for future advancements in fish welfare. In collaboration with Nofima, we will use their provided data to train and refine our tracking system, ensuring that it is well-suited for addressing welfare-related challenges in the fish industry.

## 1.2 Objectives

The motivation behind this thesis stems from Nofima's project, which endeavors to enhance fish welfare by devising robust computer vision systems capable of facilitating sustained fish tracking. Acknowledging the substantial potential of computer vision technologies in tackling welfare-related issues, our research concentrates on constructing a foundational tracker proficient in managing the intricacies of fish tracking, thereby establishing a basis for future progress in fish welfare.

In the pursuit of a dependable and efficacious tracking system, we shall utilize cutting-edge detection frameworks, such as Detectron2[6] and YOLOv8[7], and combine them with a high-performance object tracking algorithm in ByteTrack[8] to accurately discern and monitor fish. By harnessing these advanced frameworks, we aspire to develop a versatile computer vision system that can adapt to various aquatic environments in accordance with our project's objective. Two criteria must be fulfilled to deem the framework a successful implementation for future fish welfare applications. These two criteria are:

- Reliably detect fishes that are in the frame with high confidence
- Track fish reliably even though challenges such as occlusion

To develop a tracking application capable of achieving these objectives, Of developing a tracking application capable of achieving these objectives, Nofima has furnished us with data that will be employed to train our system.

---

## 1.3 Structure

This master's thesis commences by delving into the theoretical underpinnings and intricacies of the machine learning process associated with computer vision techniques. Chapter 2 introduces and examines the dataset utilised in this study to elucidate the configuration and functionality of our integrated systems. Subsequently, Chapter 3 outlines the obtained results, which are further analysed and expounded upon in Chapter 5. Finally, Chapter 6 summarises the key aspects of the thesis and provides a thorough conclusion to the study.

## Theory

Artificial intelligence (AI) constitutes a broad term encompassing diverse technologies and methodologies that facilitate machines executing tasks traditionally necessitating human cognitive abilities, such as visual perception, speech recognition, decision-making, and language translation [9]. The domain comprises multiple sub-disciplines, including machine learning, natural language processing, and computer vision. AI research aims to develop sophisticated machines capable of performing intricate tasks autonomously, minimising the need for human intervention. The approaches employed in this thesis pertain to supervised deep learning within the context of computer vision [10].

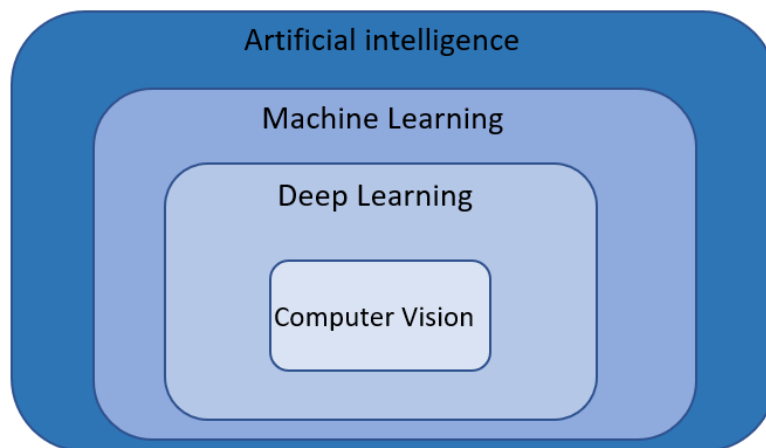


Figure 2.1: A visual representation of the artificial intelligence hierarchy, showcasing AI as an overarching field with its various subcategories, including machine learning, deep learning and computer vision.

This chapter delves into the theoretical foundations of computer vision, explaining its relationship with object detection and object tracking. This thesis utilises two object detection techniques and one object tracking approach, and it will also expound upon their underlying theory.

### 2.1 Machine Learning Theory

Machine learning, a fundamental element of artificial intelligence, utilises data and algorithms to mirror the process of human learning. This allows computers to automatically adapt and improve their performance in various tasks without being explicitly programmed to do so[11][12].

---

The fundamental aim of machine learning is to devise algorithms capable of extracting knowledge from data and applying this acquired understanding to novel scenarios. These algorithms are primarily classified into three distinct categories: *Supervised learning*, *Unsupervised Learning*, and *Reinforcement Learning*[13]. Each category encompasses various methodologies and techniques specifically tailored to address particular types of tasks and data structures.

Supervised Learning entails learning from labelled data, where input data is associated with known output values, also referred to as ground truth. By utilising this labelled dataset, the algorithm generalises and applies its understanding to new, unseen data[14]. In contrast, Unsupervised Learning operates with unlabelled datasets, enabling the algorithm to independently discern patterns and structures within the data without explicit guidance[15]. Reinforcement learning adopts a trial-and-error approach, where the model's decisions are either rewarded or penalised within a specific environment. This method involves an "agent" interacting with its environment to learn decision-making processes that yield the desired outcome without relying on labelled data[13]. In this thesis, we will focus on supervised learning.

## 2.2 Deep Learning with Artificial Neural Networks

Deep Learning is an important subcategory of machine learning, centred around the concept of neural networks with multiple layers, known as deep neural networks. These networks have the capacity to tackle intricate problems by identifying hierarchical patterns and representations within the data. Drawing inspiration from the structure and function of the human brain, deep neural networks have garnered widespread attention due to their efficacy in a wide array of applications. This section closely examines the core theoretical concepts underpinning Deep Learning in the context of Artificial Neural Networks (ANN). By examining the architecture, learning processes, and mathematical principles that shape deep neural networks, a comprehensive understanding of the fundamental theories driving the success of Deep Learning can be established.

### Neuron

The fundamental building block of an Artificial Neural Network (ANN) is the single neuron, responsible for receiving input signals, processing the information, and generating an output. Each neuron typically receives multiple inputs from either the raw data or the outputs of other neurons in the preceding layer. The inputs are multiplied by the corresponding weights, and the weighted inputs are then summed to calculate the net input as in eq 2.1. A bias term is usually added to this net input to influence the neuron's output. The neuron's net input, combined with the bias term, is processed by an activation function that introduces non-linearity, enabling the neuron to model complex relationships in the data. This process is illustrated in Fig. 2.2.

$$Z = \sum (Weight_i * Input_i) + Bias \quad (2.1)$$



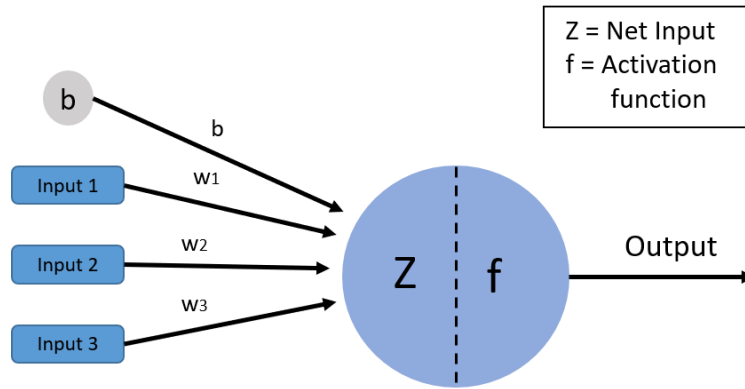


Figure 2.2: The Functionality of a Node in ANN

### Activation function

Activation functions play a crucial role in artificial neural networks by introducing non-linearity, which allows the network to model complex relationships in the data. Without a non-linear activation function, the network’s output would be a linear combination of its inputs, often proving inadequate for accurately representing intricate data patterns. There are several types of non-linear activation functions that are commonly used in artificial neural networks, the most popular being sigmoid[16], rectified linear unit (ReLU)[17] functions, and Hyperbolic Tangent (Tanh). These functions map the net input onto a new range, typically between 0 and 1 for sigmoid, between -1 and 1 for tanh, or between 0 and positive infinity for ReLU, depending on the specific function employed as illustrated in Fig. 2.3. The neuron’s output, determined by the activation function, can be passed to the neurons in the subsequent layer, enabling the network to generate predictions or decisions.

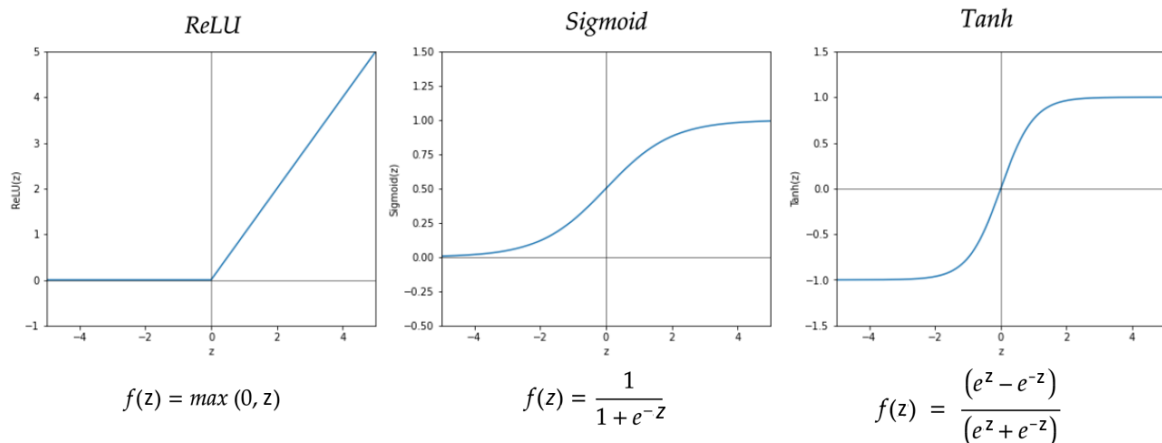


Figure 2.3: Depicts the corresponding functions and plots for each activation function ReLU, Sigmoid, and Tanh.

### Multilayer Perceptron

Building upon the concept of the neuron, the fundamental structure of an ANN consists of a collection of interconnected neurons organised into various layers. These layers may include an

input layer to receive data, one or more hidden layers to perform computations on the data, and an output layer to produce the network's predictions or decisions as portrayed in Fig.2.4. Additionally, other specialised layers may be utilised for specific purposes or within particular types of networks, such as Convolutional layers for processing images or Recurrent layers for sequential data. The connectivity between layers is defined by weights attached to nodes in both the previous and next layers, allowing for the propagation of signals through the network. When the connections between nodes in an ANN are fully connected, the network is often referred to as a multilayer perceptron (MLP)[13]. By combining multiple neurons and layers, ANNs can learn hierarchical representations and process intricate patterns in the data.

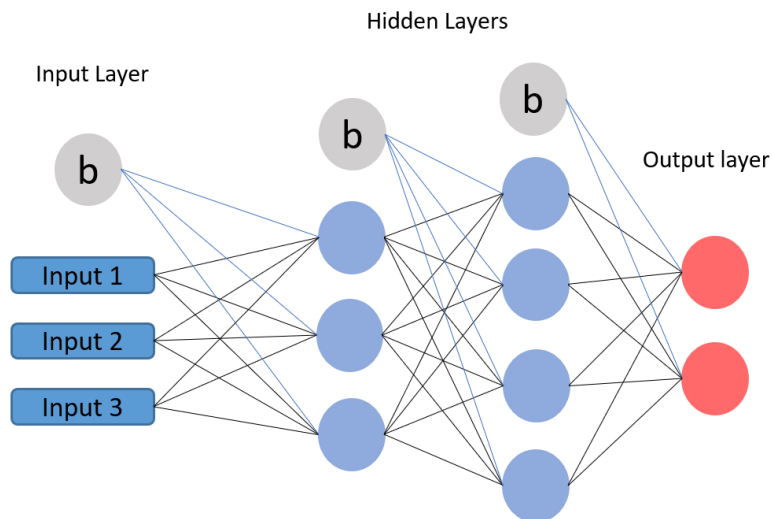


Figure 2.4: An example of a basic MLP with an input layer with three input units and an additional bias unit. There are two hidden layers, the first with three hidden units and a bias unit, and the second with four hidden units and its own bias unit. The network concludes with two output layers.

The learning procedure in an ANN entails the iterative adjustment of the weights and biases associated with the connections between neurons through a series of iterations. The process involves feeding the network with training data, which is then propagated through various layers until a prediction is generated at the output layer. To improve the network's performance, weights and biases between neurons are adjusted in order to minimise the discrepancy between the predicted outputs and the labelled data, also known as the loss. This procedure can be more comprehensively explained by the following sequence of steps:

- **initialisation:** is an essential step in the training process of an ANN, where the weights and biases associated with the connections between neurons are set to initial values before starting the training process. These initial values are typically chosen randomly from a small range, such as a Gaussian or uniform distribution. Proper initialisation is crucial, as discussed by [18], as it can affect the speed of convergence and the final performance of the model whilst also helping to avoid issues such as vanishing or exploding gradients during the training process.
- **Forward Propagation:** is a process in the network during which input data is passed through the network's layers to produce predictions. The input data is initially provided to the input layer and then successively processed by the hidden layers and activation functions. Each neuron in a layer computes a weighted sum of its inputs, adds a bias term, and applies an activation function to the result transforming the initial input value.

This transformed value is then passed on to the neurons in the subsequent layer. The process continues until the output layer is reached, where the final prediction is generated based on the network's current weights and biases.

- **Error Computation:** after forward propagation generates a prediction, the error between the predicted output and the actual target value is calculated using a loss function. Much like there are multiple activation functions, there are different loss functions and the choice depends on the problem type and the network architecture. For example, *mean squared error* is often used for regression problems, while *Binary cross-entropy* is commonly employed for binary classification tasks. The calculated loss is a performance metric that indicates how well the network is performing. This loss is then utilised in the backpropagation step to adjust the weights and biases within the network to minimise the loss and improve the network's predictive accuracy.
- **Backpropagation:** In this step, the goal is to calculate how much each weight and bias in the network contributed to the error computed in the previous step. This is done by calculating the gradients of the loss function concerning the weights and biases for each neuron in the network. Like the name implies this process starts at the output layer and moves backwards through the network.

To do this the gradient for the output layer weights is computed by first calculating the error term for each neuron in the output layer. This error term represents how much a slight change in the neuron's output contributes to the overall error. It is calculated based on the derivative of the activation function used in the neuron and the error computed during the error computation step. Once the error terms for the output layer neurons are calculated, the gradients for the output layer weights can be computed using these error terms. Fig. 2.5 illustrates this process by updating the first weight connection in the output layer.

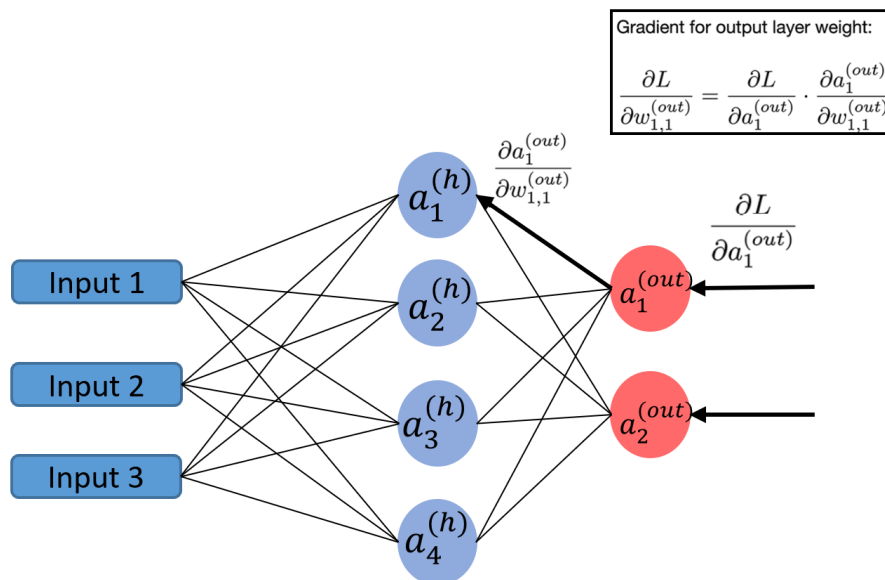


Figure 2.5: Illustration of backpropagation of the error in an ANN, computing the partial derivatives of the loss with respect to the output layer weights.  $L$  represents the loss function,  $a^{out}$  denotes the activation function for the neuron in the output layer, and  $w_{1,1}^{out}$  refers to its corresponding weights.

Then, we move back through the hidden layers, calculating the error term for each neuron in each layer. The error term for a neuron in a hidden layer depends on the error terms of the neurons in the next layer (closer to the output) and the derivative of the activation function used in the current neuron. In this way, we propagate the error information backwards through the network. Effectively computing the gradient of the hidden layers with respect to the weights and biases of the neurons like depicted in Fig. 2.6 where it illustrates the computation of the partial derivative of the loss with respect to the first weight of the hidden layer.

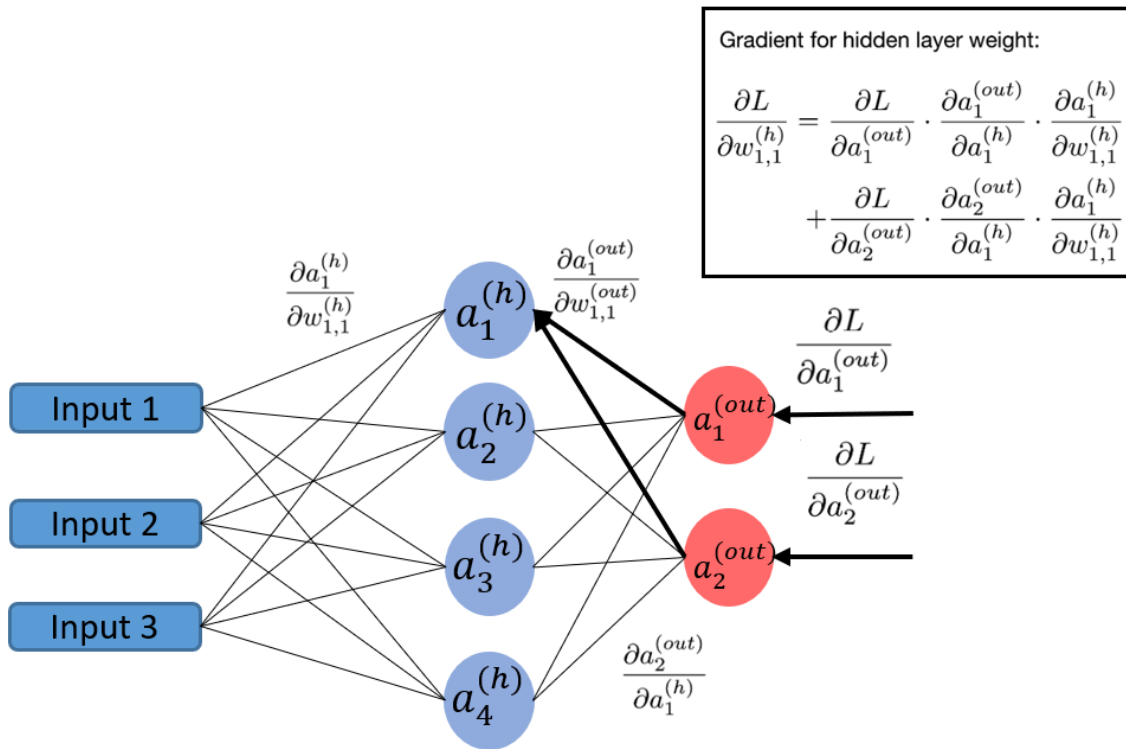


Figure 2.6: Illustration of backpropagation of the error in an ANN, computing the partial derivatives of the loss with respect to the hidden layer weights.  $L$  represents the loss function,  $a^h$  denotes the activation function for the neurons in the hidden layer, while  $a^{out}$  is for the output layer.  $w^{h,i,j}$  refers to the corresponding weights between neuron  $i$  in the current hidden layer and neuron  $j$  in the next layer, here being  $w^{h1,1}$ . The corresponding weights for the output layer can be denoted as  $w_{1,1}^{out}$ .

- **Optimization:** At this stage, the goal is to minimize the loss function by updating the weights and biases based on the gradients computed during backpropagation. The weights and biases are adjusted in the direction of the negative gradient, which helps reduce the loss.

Various optimization algorithms, such as Stochastic Gradient Descent (SGD), Momentum, RMSprop, and Adam, can be used for this purpose. These algorithms differ in the way they adjust the learning rate and utilize previous gradients to update the parameters.

During the optimization step, the computed gradients are multiplied by a learning rate, which is a small positive scalar determining the step size in the direction of the negative gradient. The product of the gradient and the learning rate is then subtracted from the current weights and biases, resulting in updated parameters that should yield a lower loss value in the next iteration.

---

Once an ANN has been trained, it has learned to make predictions or classify inputs based on the patterns and relationships it has captured from the training data. The weights and biases of the network have been adjusted through the training process to minimize the loss function, allowing the ANN to generalize and make accurate predictions on unseen data.

### 2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialized form of ANN introduced in the 1990s by [19] designed to handle grid-like data making them remarkable for image processing, computer vision, and other grid-like data applications. The architecture of CNNs design allows it to mimic the hierarchical structure of the human visual cortex by processing visual information through a series of interconnected layers. [20] This design enables CNNs to learn complex and abstract features from raw input data by building a hierarchy of feature representations.

The primary distinction between CNNs and traditional ANNs, such as MLPs, lies in the architecture of the network and the specific operations performed within the layers. While MLPs are composed of fully connected layers, in which each neuron in a layer connects to every neuron in the previous and subsequent layers, CNNs incorporate alternating *convolutional layers* and *pooling layers* to learn from the data effectively as depicted in Fig. 2.7. Additionally, CNNs use parameter sharing and receptive fields to reduce the number of parameters, making training more efficient and less prone to overfitting.

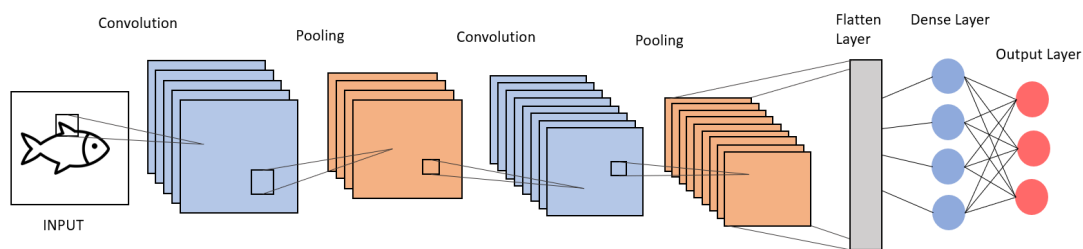


Figure 2.7: A typical Convolutional Neural Networks

Parameter sharing is a technique used in CNNs to diminish the number of parameters required to learn the features. In this approach, the same filter (and its associated weights) is employed across the entire input data, resulting in shared weights for detecting a specific feature. This reduces the number of parameters and promotes better generalisation and more efficient feature learning.

Convolutional layers apply convolutional operations, which help the network learn and extract local features from the input data. These features are represented in the feature maps of each layer, highlighting the spatial relationships and patterns found in the data. Filters, which are small matrices of weights, are crucial in generating feature maps by detecting specific features or patterns, such as edges, corners, and textures, from the input data.

The concept of receptive fields comes into play when discussing how filters in convolutional layers operate on the input data. In a CNN, each filter covers a small region of the input data, termed its receptive field, instead of processing the entire input as in a fully connected layer. This process is illustrated in Fig. 2.8. This approach allows the network to concentrate on local features and reduces the number of connections, rendering the model more computationally efficient.

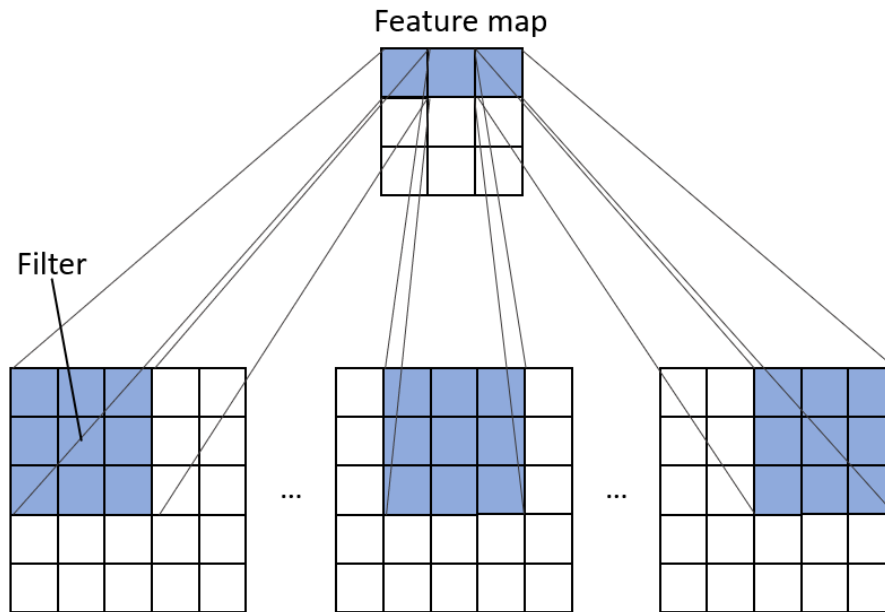


Figure 2.8: Application of a Filter Across a 2D Image in a Convolutional Operation

As the input data progresses through the network, filters in each convolutional layer operate on the original input data or feature maps generated by previous layers. Using a sliding window approach, filters move across the input, performing element-wise multiplication with the corresponding input values within the window, followed by summation. This process creates new feature maps representing the presence and location of the detected features [13]. The sliding window approach allows the network to learn hierarchical features and representations, enabling the recognition of more complex structures as the data passes through successive layers. Filters in deeper layers capture higher-level features and abstractions by combining the lower-level features detected in earlier layers.

Two important parameters that affect the convolution operation and the resulting feature maps are stride and padding. Stride refers to the number of positions the filter moves at each step during the convolution operation. A larger stride yields a smaller feature map with reduced spatial dimensions, whereas a smaller stride produces a larger feature map containing more detailed information.

Padding involves the addition of extra pixels (typically zeros) around the input data's border before executing the convolution operation. This technique helps maintain the input data's spatial dimensions and enables the filter to process data near the edges effectively.

Pooling layers serve multiple purposes in a CNN, including reducing the spatial dimensions of feature maps while preserving crucial information, controlling overfitting by introducing a degree of spatial invariance to the network, and enhancing computational efficiency. Moreover, pooling layers contribute to the network's robustness against minor translations or distortions in the input data by retaining the most prominent features. This is achieved by partitioning the feature maps into non-overlapping regions and applying an aggregation function to each region. The most common aggregation functions are *Max-pooling* and *Average-pooling*, with max-pooling being the predominant choice [13].

Max-pooling employs a pooling window that moves across the input feature map using a sliding

---

window approach similar to the one used in the convolution operation. At each window position, the maximum value within the window is extracted and added to the output feature map, as depicted in Fig. 2.9. Pooling layers, typically positioned after one or more convolutional layers in a CNN, contribute to the network's robustness against minor translations or distortions in the input data by reducing the spatial dimensions of the feature maps.

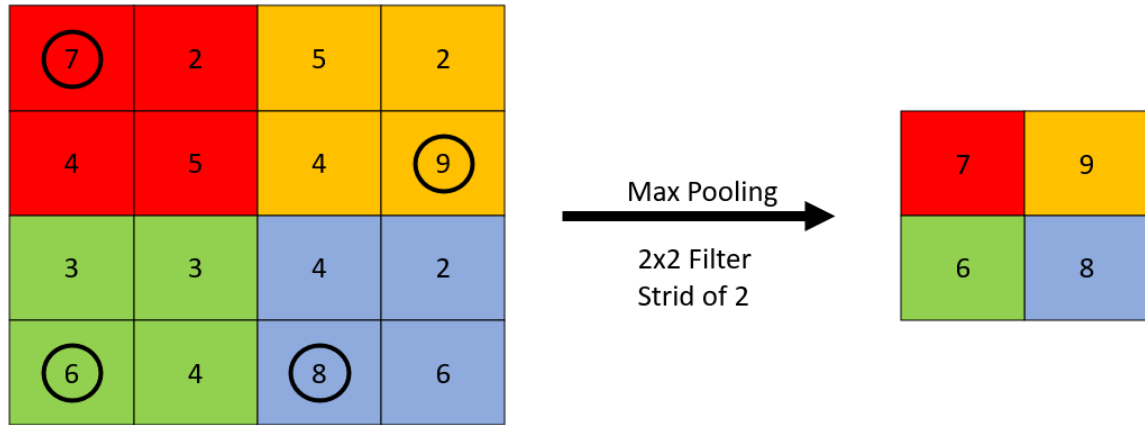


Figure 2.9: Illustration of the max-pooling operation

Average-pooling is similar to max-pooling, but instead of selecting the maximum value within the pooling window, it computes the average of all the values within the window and adds that average value to the output feature map.

Pooling layers are typically placed after one or more convolutional layers in a CNN. By reducing the spatial dimensions of the feature maps, pooling layers help make the network more robust to small translations or distortions in the input data.

The architecture of a CNN comprises these layers alternating between convolutional- and pooling layers, succeeded by fully connected layers and an output layer. This hierarchy enables the network to learn increasingly abstract features at each stage. Lower-level layers capture basic features, such as edges or textures, while subsequent convolutional layers combine these to form more complex and abstract representations, eventually recognizing entire objects. The fully connected and output layers merge the learned features to produce final predictions or classifications[21].

### Transfer learning

Transfer learning is a well-established technique in the field of machine learning, wherein a pre-trained model developed for one task is adapted to perform a distinct yet related task. This approach aims to capitalise on the knowledge acquired from the source task to enhance performance on the target task, particularly in scenarios with limited data availability. Additionally, transfer learning can accelerate training time and enhance performance since the model has already identified valuable features and patterns during its source task. This advantage helps the model achieve convergence more rapidly on the target task than a model trained from the beginning. By utilising the pre-trained model, transfer learning can decrease the computational resources needed to train the model on the target task.

---

When adopting a model for a new target task, the output layers of the model are generally substituted with novel layers that are more suited for the target task. The weights of these new layers are initialised randomly, while the weights of the remaining layers are retained, preserving the features learned during the source task. However, these weights are subject to modification during the fine-tuning process. When retraining the model for the target task, a lower learning rate is usually employed to adjust the weights, enabling weight adjustments while maintaining the learned features from the model's prior task. The decision to update the weights of the other layers is contingent upon the similarity between the source and target tasks and the volume of available data for the target task.

## 2.3 Computer vision

Computer vision is a field of AI that enables computers and systems to extract meaningful information from visual inputs such as digital images or videos[22]. This technology is applicable to various tasks, including image classification, object detection and object tracking. In essence, computer vision allows computers to see, observe, and understand the world in a way similar to human perception.

To achieve this, computer vision relies on vast amounts of data, deep learning algorithms, and specialised neural network architectures such as CNNs. Deep learning algorithms train computers to distinguish images by identifying patterns and features, while CNNs facilitate this process by breaking down images into pixels with associated tags or labels. These labels are then used in convolution operations to extract local features and make predictions about the content of the images.

The convolution process is iterative, with each iteration refining the prediction accuracy until a satisfactory level is reached. This process can be likened to a human attempting to discern an object from a distance, initially recognising the most defining features, such as hard edges and simple shapes, before gradually filling in the details.

In the context of video processing, CNNs can also be adapted to handle sequences of images or video frames, enabling the analysis of temporal relationships between them. In this thesis, two different object-detection methods based on CNNs are used and paired with a single object-tracking algorithm to detect and track fish in a video stream.

### 2.3.1 Object detection

Object detection is a fundamental aspect of computer vision, the interdisciplinary field that enables machines to perceive, interpret, and understand visual information from their surroundings. Object detection's primary goal is to identify and locate objects within images or videos accurately[23]. In contrast to image classification, which assigns a single label to an entire image, object detection seeks to provide detailed information about the presence, position, scale, and class of multiple objects in a scene as illustrated in Fig. 2.10. An essential aspect of object detection is the representation of detected objects, typically using bounding boxes or other forms of annotations that enclose the objects of interest. These annotations facilitate the quantification and evaluation of the detection performance and the visualisation and interpretation of the results.



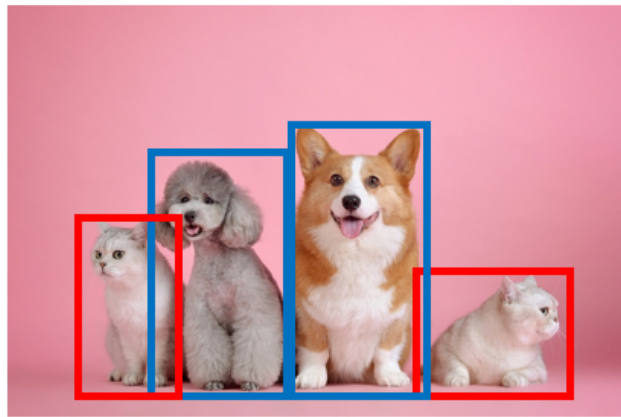
---

## Classification



**CAT**

## Object Detection



**CAT, DOG**

Figure 2.10: A comparison between image classification and object detection tasks. Image classification (left) assigns a single label to the entire image, while object detection (right) identifies and locates multiple objects within the image, using bounding boxes to represent their positions, scales, and classes. Original images from[24][25]

The field of computer vision has witnessed significant advancements in recent years, primarily driven by the development of deep learning techniques, particularly CNNs. These advancements have resulted in substantial improvements in the accuracy and efficiency of object detection systems, enabling their widespread adoption in various domains. Some popular object detection techniques include Faster R-CNN[26], SSD, and YOLO[7][27].

Object detection has numerous practical applications, such as autonomous vehicles, video surveillance, and augmented reality. The development of accurate and efficient object detection systems is crucial for these applications' continued growth and success.

### 2.3.2 Detection using Detectron2

Detectron2 is an open-source object detection and segmentation library developed by Facebook AI Research (FAIR) released in 2019. It is an upgraded version of the original Detectron library and maskrcnn-benchmark, featuring improved performance, flexibility, and usability. It is built using PyTorch, a popular deep learning framework [6].

Detectron2 provides state-of-the-art object detection and segmentation algorithms, including Faster R-CNN, RetinaNet, Mask R-CNN, and DensePose, among others. These models are pre-trained on large datasets like COCO[28] and can be fine-tuned for specific tasks. The library supports a variety of use cases, including instance segmentation, panoptic segmentation, key-point detection, and more.

In the context of object detection, the Faster R-CNN model[26], which is implemented in Detectron2, can be conceptually partitioned into three primary components: the backbone, the Region Proposal Network (RPN)[29], and the Box Head (also known as the Region of Interest (ROI) Head). These components synergistically operate to identify and localise objects within an image.

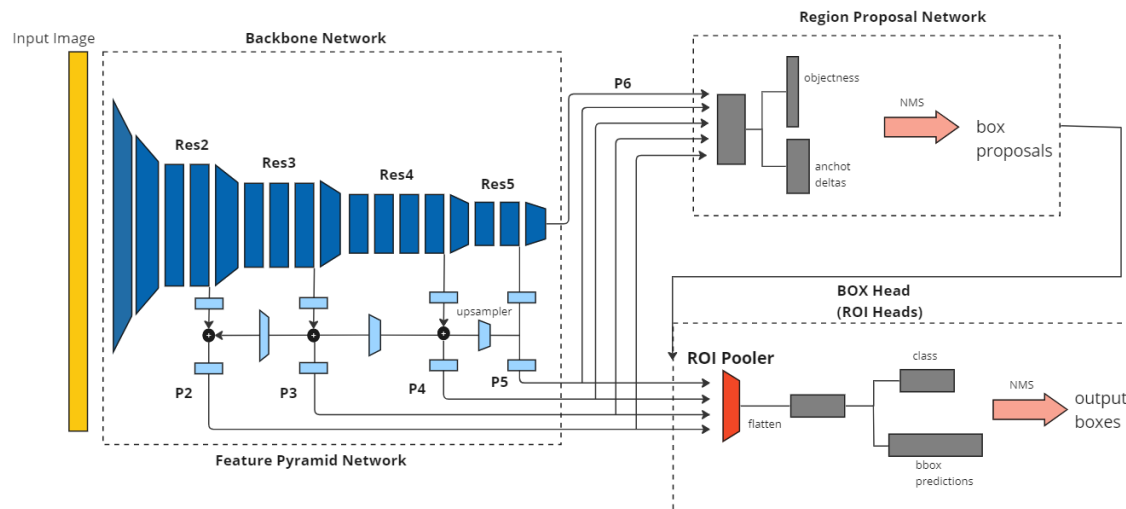


Figure 2.11: This illustration demonstrates the architecture of Detectron2, highlighting the interconnected components, including the Backbone, RPN, and ROI Head. The figure showcases how these elements work together within the object detection pipeline to process and analyze input images

## BackBone

The backbone network is responsible for extracting feature maps from the input image, relying on a deep convolutional neural network (CNN) architecture, such as a ResNet[30], combined with a Feature Pyramid Network (FPN)[31] as depicted in Fig.2.11. This backbone serves as the foundation for identifying and localising objects within an image by processing the input and generating features that capture essential information about the objects. These feature maps capture different levels of abstraction and provide the necessary information for the subsequent stages of the pipeline, such as the Region Proposal Network (RPN) and the ROI head, to perform object detection or segmentation tasks.

The FPN is a key component that enhances the object detection performance by addressing scale variance in the input image. FPN creates lateral connections between feature maps at different layers of the backbone network, linking high-resolution, semantically weak features with low-resolution, semantically strong features. It then performs a top-down pathway, where features from higher layers are upsampled and combined with corresponding lateral connections. This process results in a new feature map set that incorporates high-level semantic information and fine-grained details.

The feature maps generated by the FPN (depicted as P2-P6 in Fig. 2.11) are utilised by both the RPN and the ROI head.

## Region Proposal Network

The RPN is responsible for generating a set of proposed bounding boxes (region proposals) that potentially contain objects within the input image. To achieve this, the RPN utilises feature maps derived from the backbone network as input. The RPN is a separate network consisting of a series of convolutional layers that predict objectness scores and bounding box coordinates for each predefined anchor box in the input feature maps.

---

Anchor boxes are fixed-size bounding boxes with various shapes and sizes, which serve as reference points for object detection. These anchor boxes are distributed evenly throughout the feature maps at different spatial positions, serving as starting points for predicting the actual bounding boxes around objects. The RPN calculates the likelihood of each anchor box containing an object (objectness score) and refines the anchor box to fit the objects better using bounding box regression.

The objectness score indicates how likely each anchor is to contain an object, and these scores are used to rank the anchor boxes. To reduce the number of overlapping region proposals, the RPN applies non-maximum suppression. This technique removes proposals with a lower objectness score with a high overlap (measured using IoU) and other proposals with higher objectness scores. After this suppression, the remaining anchor boxes serve as the region proposals and are potential bounding boxes containing objects. These region proposals are then sent to the ROI head.

### **Box Head**

The Box Head constitutes the final stage of the object detection pipeline in Detectron2, tasked with predicting the object class and refining the bounding box coordinates for each region proposal generated by the RPN. The Box Head is integral to the production of precise object detection outcomes.

The Box Head receives outputs from both the backbone and the RPN. It uses the region proposals from the RPN and corresponding feature maps from the backbone to extract fixed-size features for each region proposal using the Region of Interest (ROI)[32] Align technique. The Box Head processes these resampled feature maps through fully connected layers to capture abstract information about the objects.

Separate fully connected layers are utilised for classification (object class prediction) and bounding box regression (coordinate refinement). The bounding box regression is class-specific, meaning that the network predicts separate bounding box adjustments for each region proposal for each class. When the final detection is made, the class-specific bounding box adjustments are applied to the region proposal to obtain the final refined bounding box coordinates for that specific class. This results in the final output of the Box Head.

### **2.3.3 Detection using YOLOv8**

YOLOv8, the latest addition to the renowned YOLO (You Only Look Once) family of object detection, image classification, and instance segmentation models, was developed by the Ultralytics team[7], who also pioneered YOLOv5[27]. Launched on January 10, 2023, this state-of-the-art model represents the most recent iteration in the YOLO series.

At the time of writing, the Ultralytics team has yet to release an official paper on YOLOv8, as the model remains under active development. The team continues to introduce new features to the framework and is expected to provide long-term support, given their established track record of supporting their projects. While the team has expressed interest in publishing a paper in the near future, they have prioritized addressing community feature requests as more beneficial for the community. Despite the lack of an official paper, the community has actively engaged with the development team to gain a deeper understanding of the latest YOLO model. A prime example is Fig. 2.12, developed by a user named RangKing, which visually represents the YOLOv8 architecture.

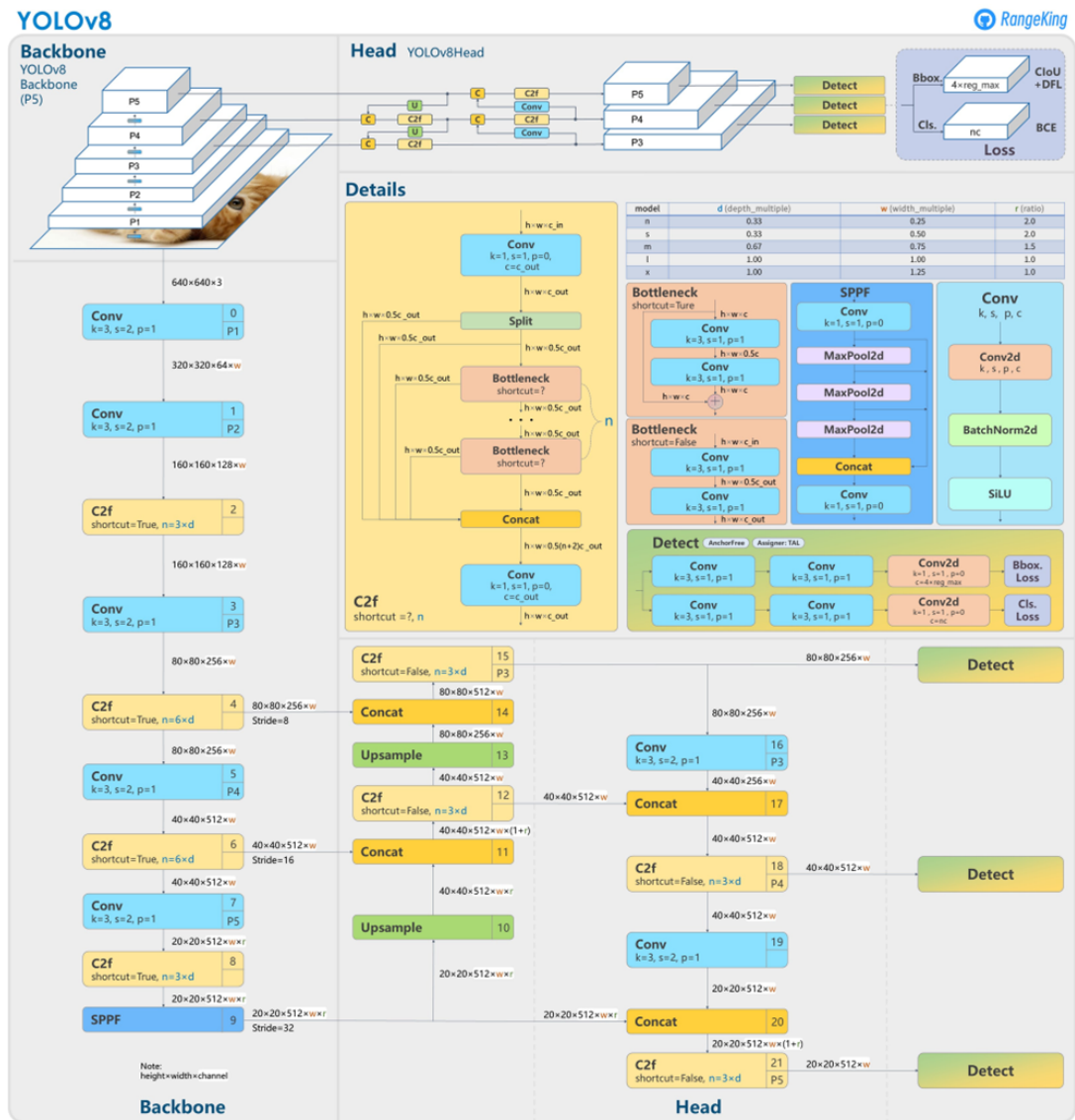


Figure 2.12: A visual representation of the YOLOv8 architecture, as interpreted by a community member named RangKing[33]. This illustration provides insight into the model’s design and structure, given the absence of an official publication from the Ultralytics team.

## Backbone

YOLOv8 utilises the same backbone architecture as its predecessor YOLOv5, namely CPS-Darknet53. The backbone is a modified version of the Darknet53 architecture that incorporates the Cross Stage Partial Network (CPS)[34] to enhance information and gradient flow between the layers in the network and improve detection performance and faster training time.

The backbone is responsible for feature extraction by generating different feature maps by processing the input images throughout its different layers. The layers that make up the CPS-Darknet are a series of convolutional layers responsible for extracting features and learning from the input data, skip-connections, to maintain better gradient flow during training and preserving features that could be lost at deeper layers of the network, and CSPBlocks used by the CSPNet component of the network.

The CSPNet uses a partitioning technique as the CSPBlocks on the feature maps that happen

---

at various different stages in the network. At each stage, the feature maps are divided into two, one part is sent directly to the next stage (the next CSP block), while the other part is processed by layers being transformed, and new feature maps are generated until the next CSPBlock where the two parts are recombined. The recombination of feature maps allows the network to maintain better gradient flow during training and enables direct communication between layers at different stages. This helps the network access and combine low-level and high-level features more effectively, improving its ability to learn the hierarchical structure of the data.

After the feature maps are generated by the CSPDarknet53 backbone, they are passed through the SPPF layer, an optimised version of the SPP layer[35] with fewer floating-point operations (FLOPs). The SPPF layer helps capture spatial information at different scales by pooling feature maps at multiple window sizes. This allows the network to better handle objects of varying sizes and scales in the input images. These feature maps are then passed to the PAN-FPN neck of the model.

### **PAN-FPN Neck**

The neck of the Model is a PAN-FPN architecture, which combines the Path Aggregation Network (PAN) [36] for Instance Segmentation and the Feature Pyramid Network (FPN)[31] for object detection. The PAN-FPN neck is used to fuse multi-scale feature maps generated by the backbone, creating a more comprehensive and robust representation of the input image. It employs a top-down and lateral connections approach for this fusion process, effectively combining low-level and high-level features. This enhances the model's ability to detect objects of varying sizes and scales, allowing it to capture fine-grained details and contextual information from the image more effectively. As a result, the model's detection performance improves across a wide range of object categories and scales. The fused feature maps are then sent to the head to be processed for the final output.

### **Head**

The head of the model is responsible for generating the final predictions based on the processed feature maps it receives from the neck. Unlike the traditional Coupled Head architecture, which predicts object class and bounding box coordinates simultaneously within a single branch, YOLOv8 adopts a Decoupled-Head approach. Within this architecture, the tasks of predicting object classes and bounding box coordinates are separated into two parallel branches. One branch is dedicated to determining object classes, while the other focuses on predicting bounding box coordinates. This division of tasks allows for more efficient and specialised processing of the input feature maps, ultimately enhancing the model's detection performance.

The branch responsible for predicting object classes processes the feature maps from the neck by using its convolutional layers and outputs a probability distribution over the possible object categories for each location in the feature maps. This processing enables the extraction of class-specific information from the input feature maps. The output consists of a probability distribution for each location in the feature maps, representing the likelihood of the presence of different object categories at that particular location. By generating these probability distributions, the model can effectively identify and classify objects within the input image, even when they occur at different scales or positions.

It processes the feature maps from the neck using its convolutional layers and outputs a set of coordinates representing the location and size of the bounding boxes. The anchor-free detection eliminates the need for predefined anchor boxes and instead relies on a dense sampling strategy,

---

where the model predicts object locations and scales directly from the feature maps at different scales.

The predicted class probabilities and bounding box coordinates from two branches are then combined to generate the final object detections. This stage includes thresholding the class probabilities to filter out low-confidence predictions, applying non-maximum suppression to remove duplicate detections by comparing the predicted bounding boxes' overlaps, and adjusting the bounding box coordinates to match the original input image scale.

### 2.3.4 Object tracking

Object tracking refers to the process of locating and following a moving object or multiple objects over time using sensor measurements such as radar, sonar, ultrasound, or a camera. The primary goal of object tracking is to identify and follow the movement of an object over time, even when the object is occluded by other objects, changes in appearance, or undergoes changes in lighting conditions[37]. It is a fundamental task in computer vision with applications in various fields, such as surveillance, autonomous driving, robotics, and human-computer interaction. In this thesis, the focus will be narrowed down to how it is applied to video sequences filmed with a camera. Thus making the goal to identify objects in a video sequence and track their movement across multiple frames.

Object tracking systems typically use computer vision techniques to extract features from the object of interest and then track those features over time. These features can include colour, shape, texture, and motion. The system then uses algorithms to predict the location of the object in subsequent frames of the video.

However, these systems face several challenges, including occlusion, appearance changes, and camera motion. Occlusion occurs when an object is temporarily hidden by other objects in the scene, making it difficult for the tracking system to maintain continuity. Appearance changes can occur due to lighting conditions, changes in pose, or changes in the object's size. Camera motion can also pose a challenge, as it can cause the object to move in unexpected ways, making it difficult to predict its future location accurately. Due to the complex scenarios in videos, detectors are prone to make mistakes resulting in imperfect predictions.

One widely accepted paradigm within Multiple Object Tracking (MOT) is Tracking-by-Detection. This approach begins by detecting the object of interest in the initial video frame using object detection algorithms such as YOLO or Faster R-CNN. The object is then tracked across subsequent frames using various tracking algorithms, such as the Kalman filter[38] and correlation filter. In conjunction with this, low confidence detection boxes are filtered out by managing trade-offs between true positive and false positive detections in an attempt to overcome the aforementioned challenges in MOT.

#### **Kalman filters**

Kalman filters are a powerful tool for estimating the state of a system based on noisy measurements. They were invented by Rudolf E. Kálmán[38] in the 1960s and are widely used today in a variety of fields. In computer vision, Kalman filters are widely used for object tracking, motion analysis, and camera pose estimation. The filter can predict the future position, velocity, and acceleration of an object based on its current state and can update the state based on the observed measurements. Kalman filters are especially useful in cases where the measurements are noisy or incomplete.

---

The Kalman filter is a recursive algorithm that uses a set of equations to estimate the state of a system. The state of a system is a set of variables that describe the system's behaviour at a particular time. In many cases, the state of a system is not directly measurable but can be inferred from noisy measurements of other variables. The method is based on the principles of Bayesian statistics, which provide a way to update the probability of a hypothesis based on new evidence. In the context of the Kalman filter, the hypothesis is the state of the system, and the evidence is the noisy measurements.

The Kalman filter has two main components: the prediction step and the update step. The prediction step uses the system model to project the current state of the system forward in time, while the update step uses measurements to correct the predicted state.

**Prediction Step:** In the prediction step, the Kalman filter uses the system model to project the current state of the system forward in time. This is achieved by multiplying the previous state estimate by the state transition matrix  $A$  and adding the effect of any control input using the control input matrix  $B$ . The result is the predicted state of the system at the next time step as depicted in Equation 2.2.

$$\hat{x}_k = A_k \hat{x}_{k-1} + B_k u_k \quad (2.2)$$

Here,  $\hat{x}_k$  is the predicted state at time  $k$ ,  $\hat{x}_{k-1}$  is the estimated state at time  $k-1$ .  $A_k$  represents the state transition matrix, whereas  $B_k$  is the control input matrix and  $u_k$  is the control input at time  $k$ .

Additionally, the filter predicts the uncertainty or covariance of the estimated state using the error covariance prediction equation 2.3. The prediction step provides an estimate of the future state of the system based on the current state and the system model.

$$P_k = A_k P_{k-1} A_k^T + Q_k \quad (2.3)$$

Where  $P_k$  is the error covariance matrix  $Q_k$  is the process noise covariance matrix.

**Update Step:** In the update step, the Kalman filter uses the measurement data to correct the predicted state estimate. The filter computes the Kalman gain, which is a weighting factor that determines how much weight is given to the predicted state estimate versus the measurement. The Kalman gain is calculated using the error covariance of the predicted state estimate and the measurement noise covariance as shown in Equation 2.4. Next, the filter updates the state estimate by adding the product of the Kalman gain and the difference between the predicted state estimate and the actual measurement. This results in an improved estimate of the system state. Finally, the filter updates the error covariance matrix based on the Kalman gain and the measurement noise covariance. This step reduces the uncertainty in the state estimate and prepares the filter for the next prediction step.

$$K_k = P_k H_k^T (H_k P_k H_k^T + R_k)^{-1} \quad (2.4)$$

where the Kalman gain is represented as  $K_k$ .  $H_k$  being the measurement matrix, whilst  $R_k$  is the measurement noise covariance matrix.

Next, the filter updates the state estimate by adding the product of the Kalman gain and the difference between the predicted state estimate and the actual measurement. This results in an improved estimate of the system state. Finally, the filter updates the error covariance matrix based on the Kalman gain and the measurement noise covariance. This step reduces the

---

uncertainty in the state estimate and prepares the filter for the next prediction step and can be given as Equation 2.5

$$\hat{x}_k = \hat{x}_k + K_k(y_k - H_k\hat{x}_k) \quad (2.5)$$

where  $y_k$  is the measured output at time  $k$ .

In computer vision, the measurement matrix  $H_k$  is used to map the state of the system to the observed measurements. For example, if the state of the system represents the position and velocity of an object, then the measurement matrix  $H_k$  can be used to project the position of the object onto an image plane.

Overall, the prediction and update steps of the Kalman filter work together to provide an accurate estimate of the system state. The prediction step uses the system model to estimate the future state, while the update step uses measurement data to refine the estimate. This iterative process results in an estimate that is progressively refined over time, resulting in an accurate and reliable estimate of the system state.

### Hungarian algorithm

The Hungarian algorithm[39], also known as the Kuhn-Munkres algorithm, is a combinatorial optimisation algorithm that can be used to solve the assignment problem. The assignment problem is the problem of assigning a set of agents to a set of tasks, such that the total cost or benefit of the assignments is minimised or maximised. The algorithm is a simple and efficient algorithm that can solve the assignment problem in polynomial time. The algorithm works by iteratively constructing a matrix of costs or benefits and then finding an optimal assignment that minimises or maximises the total cost or benefit and consists of three main steps:

1. **Row and Column Reduction:** the algorithm finds the smallest element in each row of the cost or benefit matrix and subtracts it from every element in the row. It then does the same for each column of the matrix.
2. **Assignment:** This steps consists of an initial assignment that satisfies the constraints of the problem. This can be done by selecting the smallest element in each row or column of the reduced matrix and assigning the corresponding agent to the corresponding task. If there are multiple agents that can be assigned to the same task or multiple tasks that can be assigned to the same agent, the algorithm selects one arbitrarily.
3. **Improvement:** finally algorithm attempts to improve the assignment by finding a better assignment that reduces the total cost or maximizes the total benefit. This is done by searching for a set of uncovered zeros in the matrix and attempting to construct a new assignment that includes these zeros. If a new assignment can be constructed, the algorithm updates the assignment and returns to step 2. If no new assignment can be constructed, the algorithm terminates and returns the current assignment as the optimal solution.

#### 2.3.5 Bytetrack

Bytetrack is a multi-object-tracking (MOT) algorithm developed by Yifu Zhang et al.[8] that aims to solve the common problem of occluded objects by a generic association method, tracking objects by associating them with almost every detected bounding box, instead of only using high-scoring detections that are related to the object, as low confidence detection could be a



---

result of the existing object being occluded. Thus filtering out these objects could lead to errors and fragmented trajectories regarding MOT.

To be able to perform object tracking Bytetrack takes in a video sequence, detection score threshold  $\tau$  and weights from an object detection algorithm. These weights are fed to the algorithm's in-built object detector YOLOX[40], with subsequently proceeds to do detections on a given frame in the video sequence.  $\tau$  detection default value is at 0.6 unless otherwise specified, and detections will be divided into one of two categories  $D_{high}$  and  $D_{low}$ .  $D_{high}$  being detection above the threshold, whilst  $D_{low}$  being detection whose score are lower then  $\tau$ .

After the detection has been separated, Kalman filters are adopted to predict the new locations of each track  $T$  in the current frame. It first associates high score detection boxes  $D_{high}$ , and the predicted tracks  $T$ , including the lost tracks  $T_{lost}$ . The similarity is computed using the intersection over union (IoU)[41] between the two types of boxes before adopting the Hungarian algorithm to finalize the matching based on the similarity between them. Detections and tracks that were not matched are kept.

A second association it then performed, but this time using detection from  $D_{low}$  and the remaining tracks that were kept from the first association. These matches are computed using IoU, as the low-scoring detection boxes usually contain motion blur or serious occlusion. After the matching, the remaining unmatched tracks are kept as  $T_{remain}$  whilst we dispose of the unmatched low-scoring detection boxes, as they are regarded as background noise at this point. These two associations are performed on every frame to preserve the identity of the track. Tracking trajectory lost during the second association is named  $T_{lost}$  and is kept for 30 frames and will only be deleted if they can't be matched with a trajectory within those frames. Finally, initializing new tracks for the unmatched high-scoring detections from the first association  $D_{remain}$ . This yields the final output of a given frame with the corresponding bounding boxes for each observation in the frame and the identity of its track. Note that the  $T_{lost}$  is not a part of these tracks rather they are stored for the next frame.

### 2.3.6 Evaluation Metrics

Building upon the established theories of object detection and tracking, it is essential to employ suitable evaluation metrics to assess the performance of various algorithms in these tasks. Evaluation metrics play a pivotal role in the theoretical and practical aspects of computer vision, contributing to its ongoing advancement. This section introduces the evaluation metrics that will be used to measure the performance of detection and tracking algorithms, highlighting their importance and relevance in computer vision.

Evaluation metrics are indispensable for multiple reasons. Firstly, they enable us to compare different methods fairly and objectively, taking into account factors such as accuracy, computational efficiency, and real-time performance. These comparisons allow us to discern the strengths and weaknesses of various approaches and select the most appropriate method for a specific application.

Secondly, evaluation metrics serve as essential tools for optimizing existing models. By identifying areas of weakness or inefficiency, researchers can fine-tune algorithm parameters, modify designs, and improve the overall performance of a given method. This iterative process leads to continuous enhancements in object detection and tracking techniques, fostering the development of state-of-the-art solutions.

---

Lastly, evaluation metrics help uncover research gaps and areas for further improvements. By quantitatively assessing the performance of existing methods, researchers can recognize the limitations and challenges that need to be addressed, thus guiding future research efforts and innovations in the field.

In the following sections, we will delve into the specific evaluation metrics used for assessing object detection and tracking performance. These metrics will be instrumental in guiding our analysis, enabling a comprehensive understanding of the effectiveness and reliability of various algorithms within the computer vision domain.

### 2.3.7 Intersection over Union

Intersection over Union (IoU)[41] is widely used in computer vision applications. IoU measures the similarity between two bounding boxes or regions by checking the ratio of which they overlap. It is therefore, beneficial for assessing the accuracy of object localisation, as it is commonly employed to compare predicted bounding boxes to ground truth annotations.

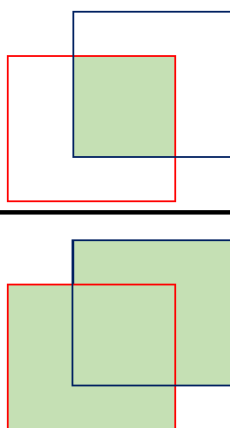
$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$


Figure 2.13: An illustration of IoU. Here, the Area of Intersection denotes the area where the two bounding boxes overlap, and the Area of Union represents the combined area of the two bounding boxes.

IoU values range from 0 to 1, with higher values indicating greater similarity between the two regions, while a lower value signifies less similarity or poor localization accuracy. In many computer vision tasks, a threshold value for the IoU score, most commonly 0.5, is employed to determine if a prediction is considered correct or not. Based on the IoU value and the chosen threshold, the predictions can be classified as true positives (TP), false positives (FP), or false negatives (FN) as illustrated in Fig. 2.14. True negatives (TN) are not relevant in this setting because they represent the absence of an object, which is not a part of bounding box evaluation. These values, excluding TN, can then be utilized to compute various performance metrics, such as precision, recall, and F1 score, to evaluate the overall performance of an object detection or tracking algorithm.

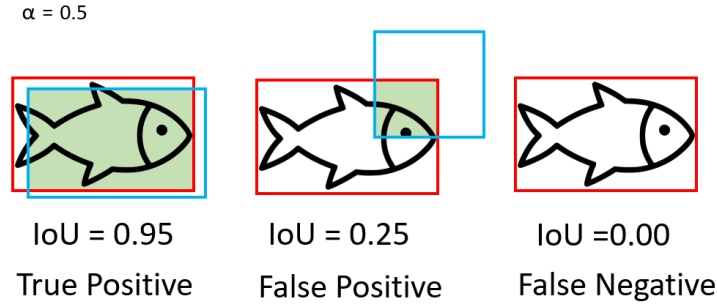


Figure 2.14: The illustration demonstrates how various detections fall into distinct categories. In this case,  $\alpha$  represents the IoU threshold of 0.5, with ground truth (GT) bounding boxes marked in red and predicted bounding boxes in blue. Each detection has its own IoU score, which determines its placement into one of the three categories: True Positive (TP), False Positive (FP), or False Negative (FN) based on the IoU score.

### Object Detection Metrics

In object detection, the accurate and reliable evaluation of algorithmic performance is of paramount importance for both model development and comparison. The mean Average Precision (mAP) metric has emerged as a widely adopted and robust measurement tool. Its popularity stems from its ability to provide a comprehensive score that encapsulates both precisions and recall across all object classes and varying levels of localization accuracy.

To compute mAP, Average Precision (AP) is first calculated for each object class separately. AP represents the area under the precision-recall curve based on the precision and recall metrics. Precision (Eq. 2.6) is a measure of how accurate the positive predictions are, while recall (Eq. 2.7) quantifies the proportion of actual positives that were identified correctly.

$$Precision = \frac{TP}{TP + FP} \quad (2.6)$$

$$Recall = \frac{TP}{TP + FN} \quad (2.7)$$

The precision-recall curve plots the precision against recall at different IoU thresholds. This curve illustrates the trade-off between precision and recall, where higher precision values correspond to fewer false positives, and higher recall values correspond to fewer false negatives.

Once the AP for each object class is determined, we calculate the mAP by taking the mean of these AP values. This involves summing the AP values for each specific class  $i$  and then dividing by the total number of object classes  $N$ , as shown in the equation 2.8:

$$mAP = \frac{1}{N} \sum AP_i \quad (2.8)$$

Hence, the AP and mAP values are identical when dealing with a single class. The mAP value ranges from 0 to 1, where a higher score indicates the detection algorithm's ability to identify objects with high precision and recall across various classes and localisation accuracy. As a result, it offers a comprehensive insight into the algorithms overall performance. This performance is usually measured over specific IoU thresholds, with mAP at an IoU of 50% (mAP50) being the most common[23]. The mAP50 evaluates the detector's performance at a moderate localisation accuracy, where a predicted bounding box must have at least 50% overlap with the ground truth box to be considered correct. On the other hand, mAP50-95 is used to assess the detection performance across a range of IoU thresholds, from 50% to 95%, with a step size of 5%. This metric provides a more robust evaluation of the detection algorithm's ability to localise objects with varying degrees of precision.

---

## Multi Object Tracking Metrics

Evaluating the performance of Multi-Object Tracking (MOT) algorithms presents a unique challenge due to varying interpretations of essential aspects that need to be measured. To address this issue and establish a standardised set of metrics for assessing tracking performance, the CLEAR MOT Metrics[42] and the Multi-Target, Multi-Camera (MTMC)[43] were introduced. These metrics offer a consistent framework for comparing and evaluating different MOT methods. In this thesis, we utilise the *py-motmetrics* library[44], which supports both the CLEAR MOT Metrics and MTMC evaluation standards. This library offers an extensive range of metrics, from which we have selected specific ones to assess the tracking performance. By employing these chosen metrics, we can conduct a comprehensive and objective analysis of various multi-object tracking algorithms, ensuring that our findings adhere to widely accepted evaluation criteria.

- **Multi Object Tracking Accuracy (MOTA)**

It measures the overall tracking accuracy, considering various factors such as false positives, false negatives, and identity switches. MOTA[42] provides a single number that combines these factors, making it a convenient metric for comparing the performance of different tracking algorithms.

$$MOTA = 1 - \frac{\sum_t FN_t + \sum_t FP_t + \sum_t IDS_t}{\sum_t GT_t} \quad (2.9)$$

where:

- $\sum_t FN_t$  stands for the sum of false negatives (missed detections) over all frames.
- $\sum_t FP_t$  stands for the sum of false positives (incorrect detections) over all frames.
- $\sum_t IDS_t$  stands for the sum of identity switches (when a tracker incorrectly changes the identity of an object) over all frames.
- $\sum_t GT_t$  represents the total number of ground truth objects (actual objects) over all frames.

MOTA scores range from  $-\infty$  to 1, where a score of 1 indicates perfect tracking performance, while negative values imply poor tracking performance due to excessive false positives, false negatives, or identity switches.

- **Multiple Object Tracking Precision (MOTP)**

While MOTA[42] measures the overall tracking accuracy, MOTP focuses on the localisation precision of the tracked objects. In other words, it evaluates how accurately the tracking algorithm can estimate the position of objects in the scene.

$$MOTP = \frac{\sum_t d_t}{\sum_t M_t} \quad (2.10)$$

- $\sum_t d_t$  represents the sum of the distances between the matched ground truth and predicted bounding boxes over all frames.
- $\sum_t M_t$  stands for the total number of matched object detections over all frames.

MOTP is calculated as the average distance between the ground truth bounding boxes and the predicted bounding boxes for all correctly matched object detections across all frames. MOTP values typically range from 0 to 1, where lower values indicate better localisation precision and higher values signify lower precision.

---

- **ID F1**

ID F1 is another evaluation metric used for multi-object tracking algorithms developed by [43]. It is based on the F1 score, a popular measure for classification tasks that considers both precision and recall. The ID F1 score focuses on measuring the tracker’s ability to maintain consistent object identities throughout the tracking process. ID F1 can be broken into the following equations:

$$\mathbf{IDP} = \frac{IDTP}{IDTP + IDFP} \quad (2.11)$$

$$\mathbf{IDR} = \frac{IDTP}{IDTP + IDFN} \quad (2.12)$$

where:

- IDTP (Identification True Positives) represents the number of correct matches between ground truth and predicted object identities.
- IDFP (Identification False Positives) represents the number of predicted object identities that do not match any ground truth identities.
- IDFN (Identification False Negatives) represents the number of ground truth object identities that were not matched with any predicted identities.

It gives us identification precision (IDP) and identification recall (IDR), when multiplied, gives us the IDF1.

$$\mathbf{IDF1} = IDP \times IDR \quad (2.13)$$

The ID F1 score ranges from 0 to 1, where a higher score indicates better performance in maintaining consistent object identities. A score of 1 signifies perfect tracking of identity preservation, while a score of 0 indicates no correct matches between ground truth and predicted object identities. This metric is particularly useful in situations where it is important to maintain object identities throughout the tracking process accurately.

- **Number of Identity Switches (Num Switches)**

Num Switches is an evaluation metric for multi-object tracking algorithms that quantify the number of times a tracker incorrectly changes the identity of an object during the tracking process. This metric is particularly useful for assessing a tracker’s ability to maintain consistent object identities, especially in scenarios where objects may intersect or become occluded.

A lower number of identity switches indicates better performance in preserving object identities throughout the tracking process, while a higher number of switches implies that the tracker is struggling to maintain consistent identities for the objects being tracked.

- **Tracking Lifespan** Tracking life span metrics, such as Mostly Tracked, Partially Tracked, and Mostly Lost, are used to assess the performance of multi-object tracking algorithms by analyzing the consistency of object tracking over time. These metrics provide insights into the tracker’s ability to maintain object identities and the robustness of the tracking algorithm.

- **Mostly Tracked(MT):** counts the number of objects that the tracking algorithm is able to follow for a significant portion of their visible presence, specifically more than 80% of the time. A high MT count indicates the algorithm’s ability to consistently keep objects within its tracking scope for most of their presence in the scene.

- 
- **Partially Tracked(PT):** counts the number of objects that the tracker is able to follow for a moderate portion of their visible presence, specifically between 20% and 80% of the time. A high PT count suggests that the tracking algorithm might have intermittent difficulty keeping objects within its tracking scope, possibly due to challenges such as occlusions, object interactions, or other complexities in the scene.
  - **Mostly Lost:** counts the number of objects the tracking algorithm follows for only a small portion of their visible presence, specifically less than 20% of the time. A high ML count indicates that the tracking algorithm frequently loses objects from its tracking scope.

To thoroughly assess tracking performance, it is essential to consider all these metrics in conjunction. Each provides unique insights into the performance of a multi-object tracking algorithm. We can comprehensively understand the algorithm's strengths and weaknesses by analysing these metrics together. This holistic assessment enables more informed decisions when selecting or developing tracking methods for specific applications, ultimately leading to improved tracking performance and more robust computer vision systems.

# Chapter 3

## Method

This chapter outlines the research methods used in this study. To start off, we will cover the hardware and software resources that were used throughout the project, including the necessary libraries and dependencies for the object detectors and tracking algorithms.

Next, is to go into detail about the dataset we worked with. This includes how we annotated the data and what steps we took to preprocess it. Finally, we will discuss how we integrated our object detection models and tracking algorithm. By providing a clear picture of the methods used in our research, this chapter sets the stage for further discussions and analyses in the chapters to follow.

### 3.1 Software and Hardware

Starting with the software resources, we relied on two different detection frameworks, Detectron2 and Yolov8, and the Bytetrack tracking algorithm. We aimed to use these tools for tracking fish over extended periods within a video sequence. These complex frameworks involve numerous dependencies, listed in detail in Appendix A.

To accommodate the considerable computational requirements of image processing, we opted to use Google Colab Pro’s GPUs for training our models rather than relying on our laptops. The specific specifications associated with these GPUs are elaborated upon in Table 3.1

Table 3.1: GPU specs for Google Colab Pro

System	GPU Name	GPU RAM
Google Colab Pro	Tesla V100-SXM2	16GB
Google Colab Pro	A100-SXM4	40GB

### 3.2 Dataset

The dataset utilized in this thesis was provided by Chris Noble at Nofima. It was initially collected for the 2021 salmon study, *FASTWELL*, and solely comprises salmon footage. The footage originates from six research tanks, each inhabited by 30-40 fish. Underwater cameras stationed within these tanks captured 12 videos, two from each tank. Every video features 5 minutes of footage, recorded at 25 frames per second (FPS) with a resolution of 576p. Although this offers reasonable clarity, it does not yield the sharpness of higher-resolution footage or the smoothness of videos with a higher FPS.

Since the data were not originally intended for computer vision purposes, the data had to be repurposed for this study. The necessary data annotation process is time-consuming, and given the limited scope of a 30-credit master's thesis, annotating footage from all channels was not feasible. As a result, we chose Channel 6 as the single video channel for training the detection models, while footage from Channel 3 was selected to serve as a separate test set.

These channels were selected due to their ability to capture the challenging and dynamic aspects of the underwater environment, such as poor lighting, reflection, distortion, and various degrees of occlusion, with Fig. 3.2 showing some of these challenges. At the same time, they offer clear views of multiple fish from different perspectives, as shown in Fig 3.4.

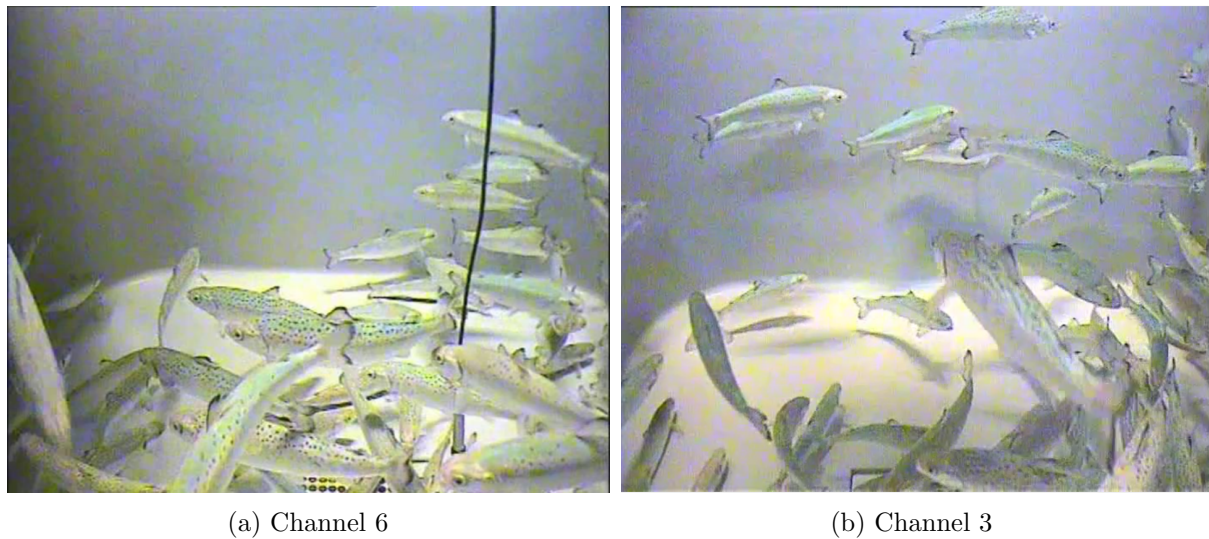


Figure 3.1: The variability in fish appearance from different angles in the Training and Test datasets

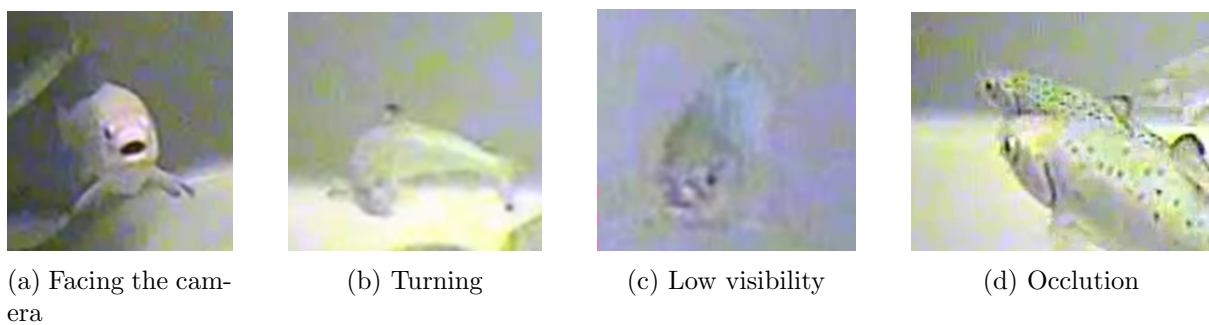


Figure 3.2: Examples of challenges in the underwater environment

Despite their similarities, there are notable differences between the two channels in the lighting conditions. The footage from Channel 3 is generally brighter, whereas Channel 6's footage contains more shadowed areas shown in Fig. 3.4.

Having examined and discussed the datasets composition and inherent challenges, the following sections will delve into the essential procedures of video annotation and data preprocessing. These critical steps transform the raw footage into a structured format, paving the way for the development of a robust and accurate tracking algorithm.



---

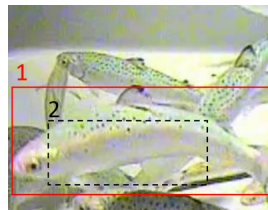
### 3.2.1 Video annotation and Preprocessing

The video annotation process demands strict attention to ensure consistency, accuracy, and the ability to handle challenges like occlusion or fish leaving the frame. To address these complexities, we established a set of guidelines for annotating the data. These guidelines ensure that the target fish are clearly defined and the annotations are reliable. Here are the established annotation guidelines:

1. As long as the fish is visible in the frame, it shall have the same tracking ID.
2. If a fish leaves the frame, there is no way of determining if the same one returns. Therefore, every fish that enters the frame will be given a new tracking ID.
3. When a fish is visible but partially occluded, it should be annotated using an occluded bounding box, as in Fig. 3.3a
4. If a fish is fully occluded and cannot be identified, it should not be annotated. Illustrated in Fig. 3.3b
5. When 50% or more of the fish is visible, and if this is the only visible part, it should be annotated with a normal bounding box. An example of this is shown in Fig. 3.3c



(a) Two fish, one with a normal bounding box, the other is marked with the red dotted box as it is partially occluded.



(b) Fish 2, is fully occluded by fish 1. Therefore it should not be annotated. The black dotted box indicates full occlusion.



(c) 50% of the fish is visible, and the rest is fully occluded.

Figure 3.3: Annotation scenarios

Additionally, the annotation format has been carefully considered to ensure the data is compatible with the computer vision algorithms used. The annotation format should provide a standardised and machine-readable representation of the annotated data, enabling efficient data processing and analysis.

With established rules, the final annotation format was decided to be bounding box annotation in the COCO JSON file format[45]. Bounding box is a widely supported format and is supported by both detection algorithms used in this thesis, unlike key-point detection which is not supported by YOLOv8 at the time of writing. Bounding boxes are capable of capturing fish accurately throughout the frames and can be marked as occluded as can be depicted in Fig. 3.4.

The COCO JSON format is also capable of retaining this information as the JSON file includes "image.id", "tracking\_id", "class", "occlusion", and "bbox" (bounding box). The latter consists of x and y making up the centre and height, and width of the box. This format is also the same annotation format that Detectron2 accepts. However, YOLOv8 does not, so a conversion

---

to the YOLO format is needed. This conversion was possible through the use of the Roboflows conversion platform[46].

The YOLO format also retains the same information as the JSON file except for occlusion tags. The information is also stored differently than it is in the COCO JSON format, where everything is stored in a single JSON file. YOLO stores the information in multiple text files, one for every frame. The exclusion of the occlusion tag is not a major issue. This is because many other object detection algorithms, including Detectron2 and YOLOv8, are incapable of handling occlusion during detection. Consequently, occlusion tags in the annotations become obsolete as they are read as a normal bounding box by these detectors. However, including these tags can be used for future work as further advancements within the computer vision fields will make them more compatible.

Moving on to the practical application of these formats, the right tools are essential for creating efficient and effective annotations. Initially, we considered using the annotation tool CVAT, as it could produce annotations that fulfilled our criteria. However, manual annotation is a labour-intensive and time-consuming process. Even with tools to speed it up, such as interpolation, it would take too long to do it ourselves. Thus, the decision was made to outsource the annotation to an annotation studio to ensure the process would be as efficient as possible. With financial support from the Faculty of Science and Technology at NMBU and NOFIMA, totalling 1200 USD, we collaborated with LabelYourData for the annotation process. The annotation studio used an annotation tool built on top of CVAT, thus allowing us to ensure that the annotation standards were maintained throughout the process. However, we could only request annotations for up to 60 seconds due to budgetary constraints, which were divided between the two videos, with 50 seconds for channel 6 and 10 seconds for channel 3.

The final annotations, as illustrated in Fig. 3.4b, comprise two sets of bounding boxes, one for when the fish is occluded and another for when it is not. These bounding boxes are assigned to each fish and associated with a distinct tracking ID, persisting throughout all frames as long as the fish remains visible.

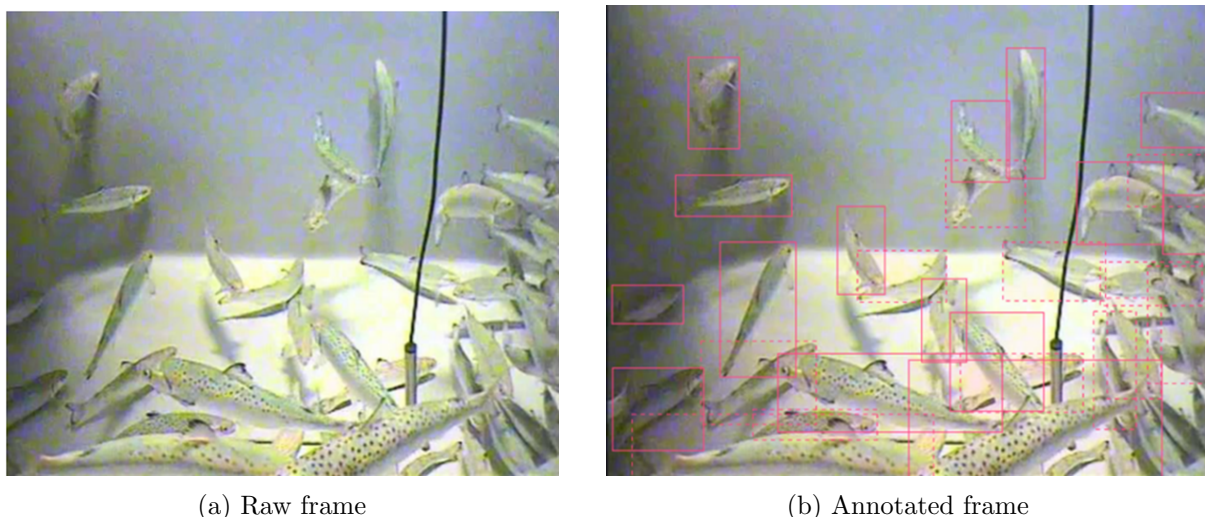


Figure 3.4: Depiction of the same frame before and after annotation. Fishes in the annotated frame are depicted with a red bounding box surrounding them, occluded fish are surrounded by a dotted red bounding box

The annotated dataset from Channel 6 is split into three parts: a training set composed of the first 35 seconds, a validation set spanning the next 5 seconds, and finally, a testing set consisting

of the last 10 seconds of the video. All these datasets, including an additional 10-second segment from channel 3, are initially preserved in the COCO JSON format and additionally converted into the YOLO format with the assistance of Roboflow, giving us our final dataset as depicted in Fig. 3.5.

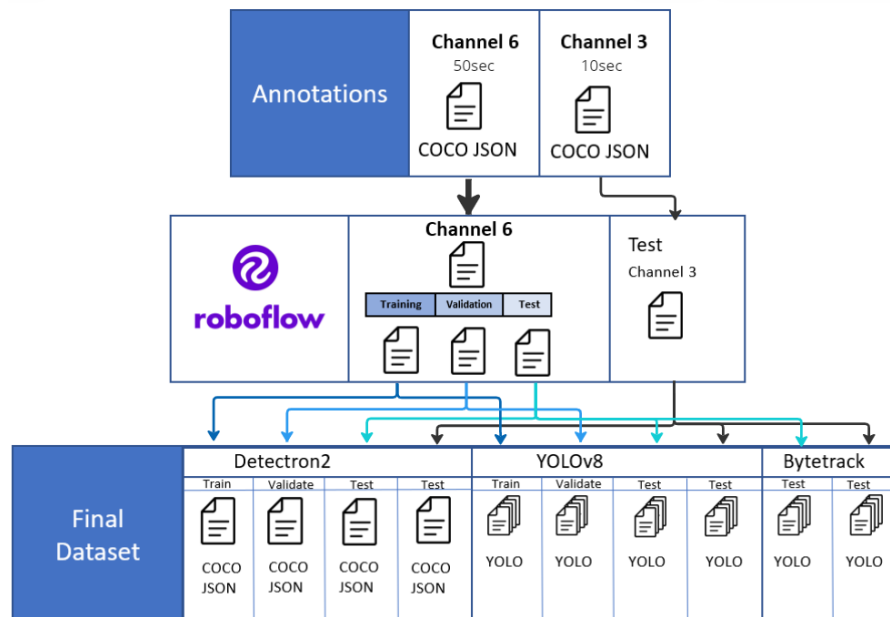


Figure 3.5: The preprocessing pipeline to split and convert the annotated dataset into the final dataset

Having successfully transformed the final annotations into the appropriate formats and partitioned them accordingly, the stage is now set for these annotations to be fed into the object detectors, thereby facilitating the execution of object detection tasks.

### 3.3 Detection using Detectron2

With the data in play, the first detector used for our object detection task was the Detectron2 framework. Using the pre-trained *faster\_rcnn\_R\_50\_FPN\_3x* model leveraging the Faster R-CNN structure with a ResNet50 backbone as the model’s balanced mix of complexity and computational performance, its ability to detect objects at various scales should translate well for fish detection. To tailor the framework for our specific use case of fish detection, we configure the expected number of classes for the ROI heads to one, considering that our dataset comprises only a single class.

In our implementation, we also use Detectron2s built-in support for data augmentation techniques to enhance the model’s robustness and generalisation capabilities. During training, the input images undergo a series of transformations, including flipping, brightening and colour jittering. As a result, the model becomes more invariant to variations in object appearance, orientation, and illumination, which are common in underwater environments. By employing these data augmentation strategies and integrating them into our training pipeline, we further hope to improve the performance of the trained model for fish detection. This approach ensures that the model can better adapt to diverse underwater conditions, increasing the overall accuracy and effectiveness of the fish detection system.

---

Additionally, we have opted for the default parameters provided by Detectron2 for our model. Although fine-tuning these parameters could potentially enhance the model’s performance, such an optimisation process is generally time-consuming and computationally demanding. A detailed overview of the different parameters that were used in our Detectron2 model is provided in Appendix B.1. By using these default parameters, we efficiently allocated our computational resources, allowing us to concentrate on other critical aspects of our research. However, the default configuration does not compute the mAP values so an additional scrip was made to evaluate the detection performance.

### 3.4 Detection using YOLOv8

For this study, we also employed the recently released YOLOv8 model. Similar to previous YOLO versions, this model is available in various sizes: *Nano*, *Small*, *Medium*, *Large*, and *Extra Large*, all trained on the ImageNet dataset. Smaller models with fewer layers offer faster inference times but may compromise accuracy while reducing memory footprint and computational requirements. On the other hand, larger-sized models feature more layers and increased complexity, leading to better accuracy but slower inference times. These models also have more parameters, increasing memory and computational demands but potentially improving overall performance. In this study, we selected the YOLOv8 small model, which balances computational efficiency and detection accuracy.

To enhance YOLOv8’s ability to generalise well across various scenarios, we adopted the mosaic data augmentation approach[47], which combines objects from different images into a single mosaic image. This technique encourages the model to learn to detect objects of varying sizes, aspect ratios, and appearances, consequently increasing the diversity of the training dataset and improving the model’s ability to handle small objects.

YOLOv8 offers various adjustable parameters to enhance the model’s performance. In our study, we opted to use the default parameters for our model, as extensive hyperparameter tuning can be both time-consuming and computationally demanding. An overview of the different parameters can be found in Appendix B.2. By relying on these default parameters, we could allocate our resources more efficiently and focus on other aspects of our research.

### 3.5 Tracking using ByteTrack

Building upon the detectors discussed in the previous sections, we now shift our focus to the tracking algorithm implemented in this study. As covered in Sec. 2.3.5, ByteTrack relies on YOLOX for detection. To facilitate our preferred detection models, we employ the Supervision framework[48], which acts as a bridge, facilitating the integration between the chosen detection models and ByteTrack. This process is depicted in Fig. 3.6.

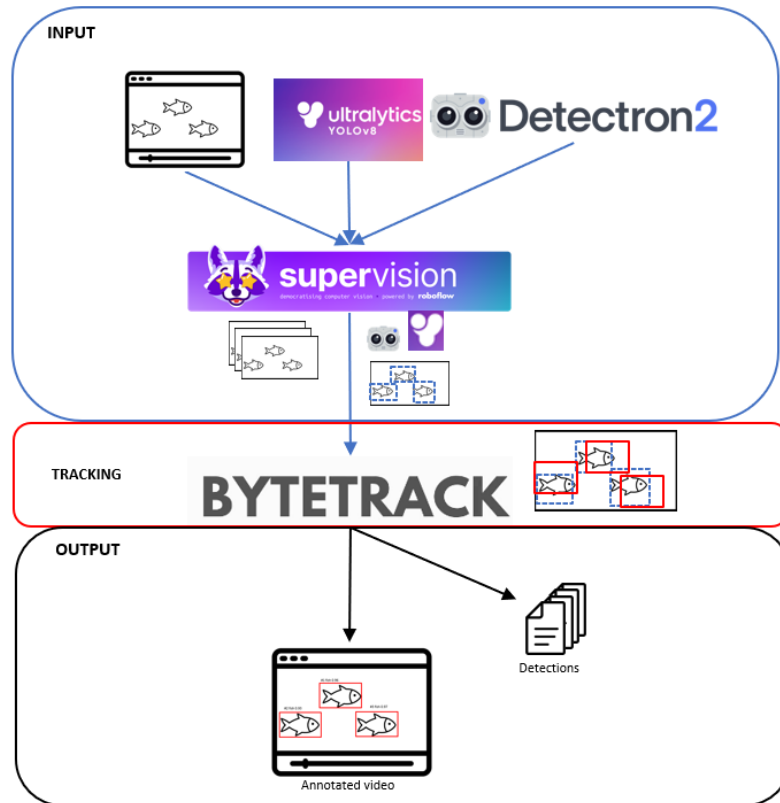


Figure 3.6: ByteTrack tracking pipeline

As depicted in Fig. 3.6, the Tracking pipeline involves three distinct phases. In the input phase, the Supervision framework manages the memory-intensive nature of video data. Rather than simultaneously loading all frames, it reads and processes each frame individually, significantly easing memory demand. Each frame is handheld by a detector which produces detections.

During the tracking stage, ByteTrack receives these detections and predicts their locations in the subsequent frame. Detections from two consecutive frames are matched using Intersection over Union (IoU), and tracking IDs are assigned accordingly.

The final phase produces an output that includes an annotated video with the predictions and a series of text files containing the detection information. The annotated video mirrors the input video but features bounding boxes for each detected fish, along with tracking IDs and confidence scores, as depicted in Fig. 3.7. Each text file, one for every frame in the video, contains bounding box coordinates in the YOLO format and the corresponding tracking ID. This information is used later to evaluate ByteTrack's tracking performance.

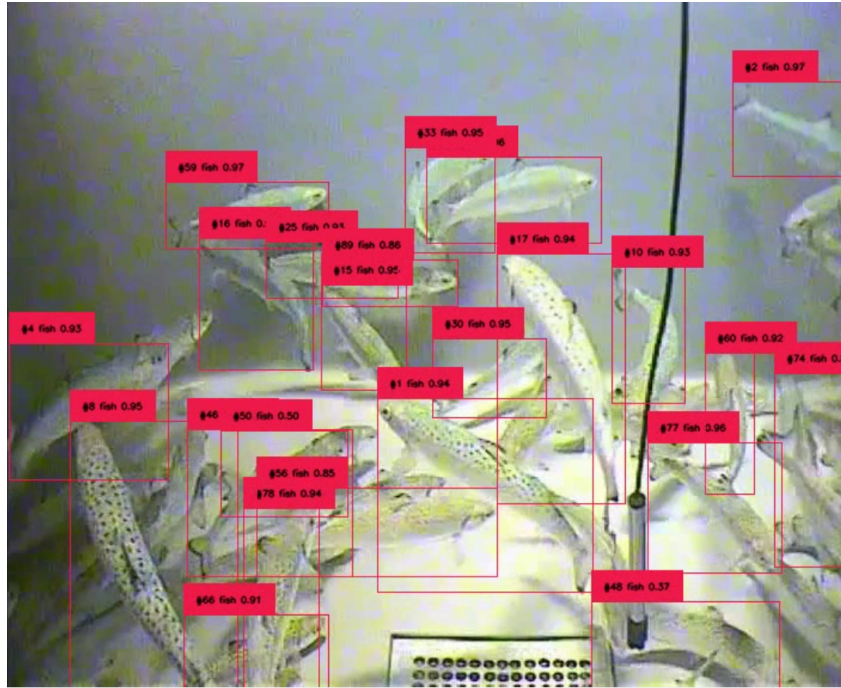


Figure 3.7: Sample frame from ByteTrack’s final output video: Each identified fish is encompassed by a distinct bounding box, accompanied by a tracking ID and detection confidence.

### Evaluation of ByteTrack

To assess ByteTrack’s tracking performance, we utilize the `py-motmetrics`[44] library to access the evaluation metrics discussed in subsection 2.3.7. Evaluating the performance requires matching ByteTrack’s predictions with the ground truth annotations from the same video feed, which can be challenging due to potential differences in tracking IDs between ground truth and predicted boxes. To resolve this issue, a prediction ID is mapped to a ground truth ID using the IoU as illustrated in Fig. 3.8. The IoU must exceed a threshold of 0.5, and after meeting this condition, the detection with the highest IoU is matched to the corresponding ground truth ID. Once the mapping is complete, the MOT metrics can be computed.

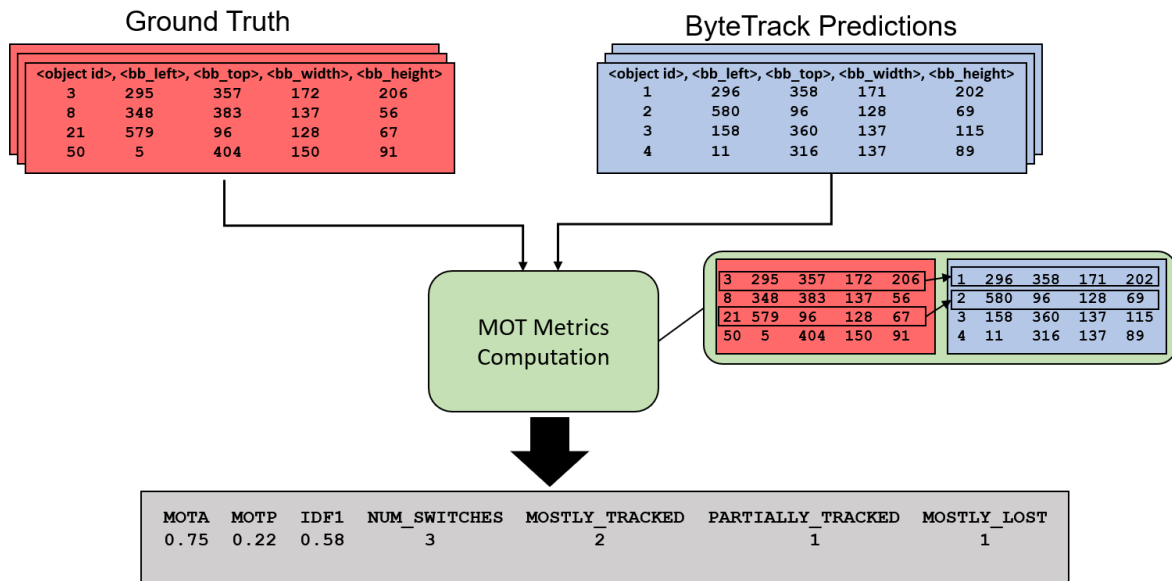


Figure 3.8: Illustrates the process of matching ground truth annotations (indicated in red) with the predictions (indicated in blue) to enable the computation of MOT Metrics for the video sequence

## Results and Discussion

In this results chapter, the object detection capabilities of Detectron2 and YOLOv8 are investigated in the context of fish detection. Their performance is evaluated individually, focusing on their ability to detect fish accurately and efficiently, a necessity for subsequent tracking applications. To measure their efficacy, specific evaluation metrics covered in subsection 2.3.6 are employed, alongside an analysis of the detectors' loss functions, providing insights into the training process.

After evaluating detection performance, each detection framework is paired with our tracking algorithm, ByteTrack, to establish a robust fish detection and tracking system capable of handling the unique challenges in fish tracking.

Upon successful implementation of the two iterations, the focus shifts to assessing the performance of the tracking results. A comparative analysis of the tracking outcomes for the two iterations is conducted at this stage. This evaluation offers valuable insights into the effectiveness of the integrated systems and aids in determining the most suitable combination for our specific fish detection and tracking application.

Throughout this results chapter, the performance of the two selected detectors is thoroughly examined, scrutinising their synergy with the tracking algorithm. The effects of diverse hyperparameters on the performance of the tracking algorithm are also explored, indicating whether the tracker shows signs of improvement. As the chapter progresses, potential factors contributing to the observed outcomes are discussed, shedding light on the underlying dynamics of our fish detection and tracking system. The chapter concludes with a comparison of the tracking performance between the two integrated systems, providing valuable insights into their relative strengths and weaknesses.

### 4.1 Detectron2 and Bytetrack Performance

#### 4.1.1 Detection Results

In this section, we assess the performance of Detectron2 for fish detection. We monitor the trends in total loss, validation loss, and mAP values at various IoU thresholds during the training iterations, as depicted in Fig. 4.14. Notably, the mAP50 and mAP50-95 values reach the plateau around mAP 75% and 45%, respectively.

As the training progresses, the total loss and validation loss decrease, suggesting that the model is learning to minimise the discrepancies between its predictions and the ground truth.



Around 1500 iterations, these losses stabilise, indicating that the model has reached a point of convergence where further training might not lead to significant improvements. Simultaneously, the mAP values, particularly mAP50 and mAP50-95, flatten out and reach the plateau, suggesting that the model’s ability to detect objects at different IoU thresholds has also reached its limit. Consequently, additional training may not result in substantial enhancements in detection performance.

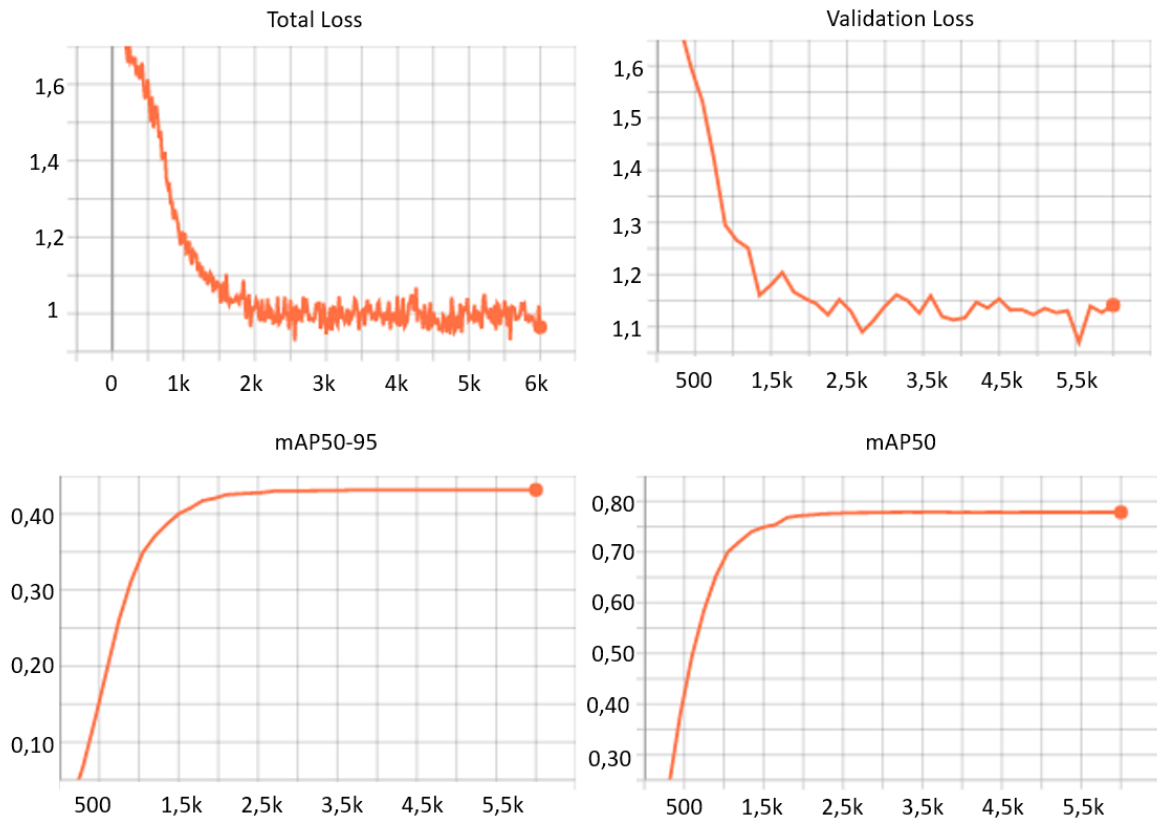


Figure 4.1: Detectron2’s training performance

In addition to examining the loss and mAP trajectories during the training process, the final model’s performance was evaluated on both the validation (ch6) and testing (ch3 and ch6) datasets. The mAP values corresponding to distinct IoU thresholds are presented in Table 4.1.

Table 4.1: Mean Average Precision (mAP) for different IOU thresholds in training and testing for Detectron2

Metric	Validation ch6	Test ch6	Test ch3
mAP50-95	0.431	0.395	0.462
mAP50	0.778	0.687	0.769
mAP75	0.438	0.419	0.512

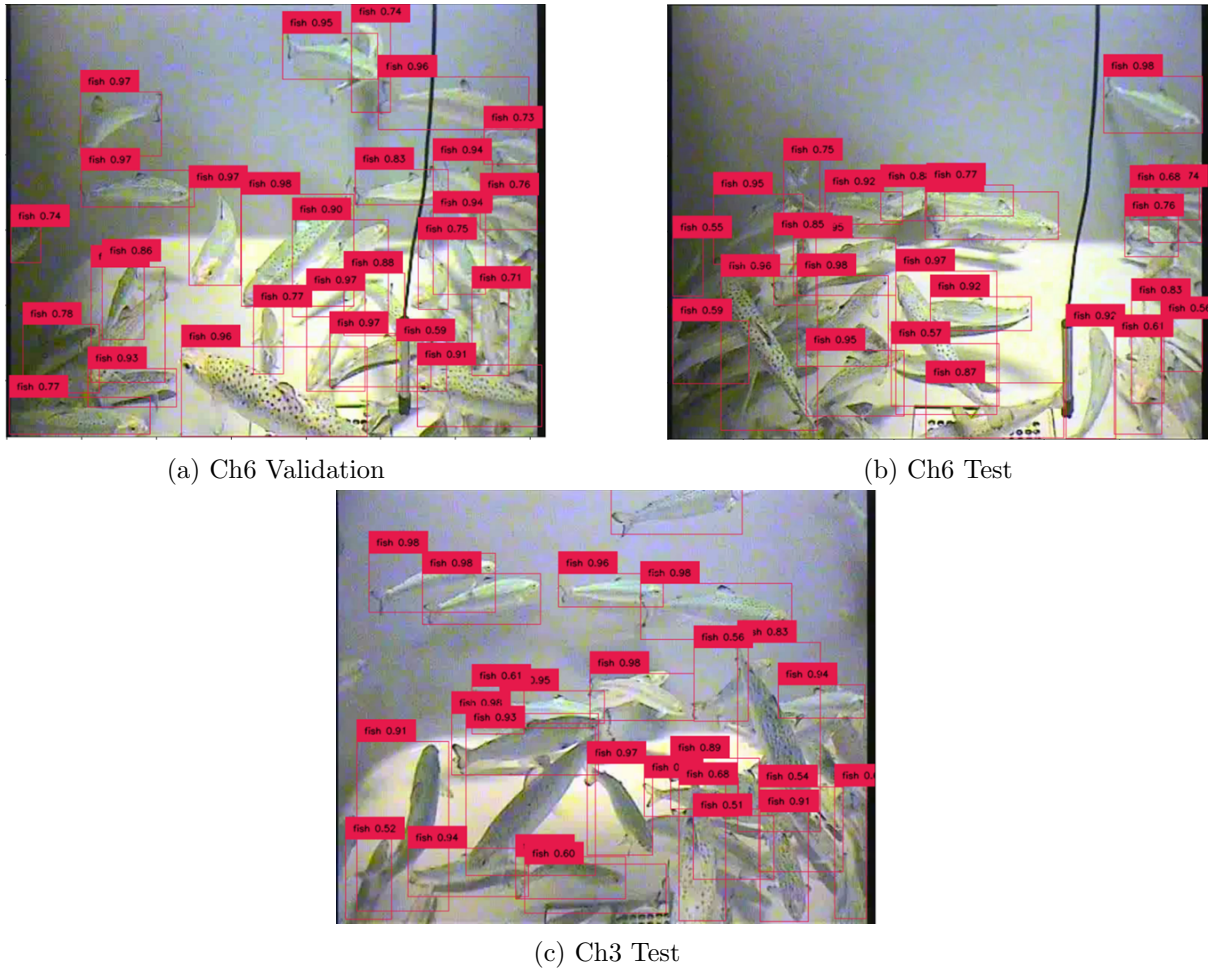
The model’s performance in detecting fish varies across different IoU thresholds and datasets. For instance, the model exhibits better detection capability at the mAP50 level, while the performance drops considerably at the more stringent mAP50-95 level. This illustrates the model can detect fish reasonably well when the IoU requirement is less strict. Still, its performance declines when a higher level of overlap between predictions and ground truth is expected.

---

Notably, the model performed better on the ch3 test set than the validation and test datasets from ch6, particularly at higher IoU thresholds. This was somewhat unexpected, as models typically perform better on data that mirrors their training set. Although both datasets were collected in similar environments, depicting comparable scenarios, the footage from ch6 exhibited more clustered fish throughout its duration. This clustering could challenge the detector's ability to localise individual fish with certainty. On the other hand, the brighter lighting conditions in the ch3 dataset could have facilitated the model's ability to distinguish individual fish better, thereby contributing to its better detection performance on this dataset.

Furthermore, the model's detection performance on the testing ch6 dataset shows a decline compared to the validation dataset, particularly at the mAP50 level. However, the mAP75 value remains fairly stable, indicating a similar performance in the model's ability to detect objects with a 75% IoU threshold on the test dataset compared to the validation dataset.

The differences in performance between the two testing datasets are due to variations in factors such as fish behaviour and environmental conditions. These aspects could influence the model's capacity to generalise across datasets, which is paramount for real-world applications. To further investigate the factors contributing to the performance discrepancies between the datasets, we visually compared detected frames from each dataset, as shown in Fig. 4.2. These comparisons reveal environmental conditions, fish behaviour, and image quality differences. We suspect that these observed differences could have impacted the model's detection performance, although additional investigation would be required to establish this relationship definitively.



(a) Ch6 Validation

(b) Ch6 Test

(c) Ch3 Test

Figure 4.2: Comparison of detected frames from the ch6 validation, ch6 test, and ch3 test datasets.

The environments across all three datasets are largely similar, with occlusions of varying degrees present. A distinctive feature of the ch3 dataset (Fig. 4.2c) is its brighter lighting condition compared to the ch6 validation and test datasets. Additionally, the absence of tank-cast shadows in the ch3 dataset could be contributing to the model’s enhanced performance on this dataset. However, the model appears to struggle more with detecting fish in areas with higher fish density, as indicated by the lower confidence scores in these scenarios across all datasets. This observation becomes particularly prevalent as the ch6 footage contains these crowded areas for longer than ch3. Consequently, the higher fish density in ch6 appears to be the key factor contributing to the performance discrepancy between the datasets.

The model also struggles with more severe occlusions, even when fish are not densely clustered. This observation is more evident when examining the zoomed-in sections of the Validation Ch6 and Test Ch3 datasets, as shown in Fig. 4.3. Both fish are visible and detected despite minor occlusion in the Ch6 case (Fig. 4.3a). However, in the Ch3 example (Fig. 4.3b), although both fish are visible, the overlapping is more pronounced, resulting in more severe occlusion. Consequently, Detectron2 fails to detect the fish being occluded.

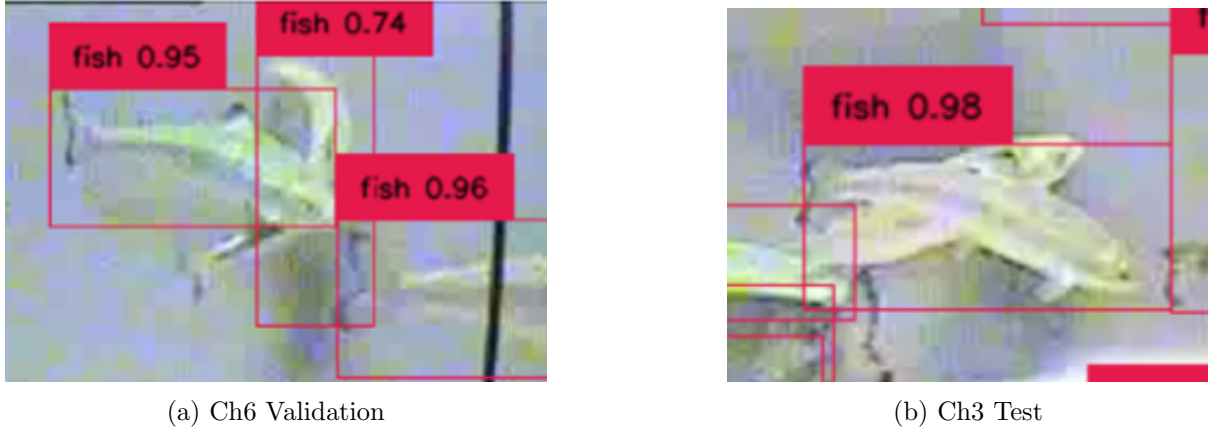


Figure 4.3: Examples of severe occlusions in the datasets

The observations from the frame analysis align with our findings regarding the higher mAP values and support the notion that the model demonstrates a reasonable fish detection performance for less strict IoU thresholds. As the IoU threshold increases, the model’s ability to detect fish accurately becomes more limited. This is particularly evident in the presence of severe occlusions and fish clusters, scenarios that inherently require stricter thresholds. In addition, lower image resolution may exacerbate these challenges, resulting in the loss of details, which can be crucial for accurate detection.

Despite these limitations, our tracking algorithm will employ the trained Detectron2 model for fish detection. In the following sections, we will analyze the performance of this combined system, aiming to gain insights into its effectiveness and to determine the Detectron2 model’s suitability for our specific application.

#### 4.1.2 Tracking Results

Building upon the evaluation of Detectron2 as a detector, we now present the tracking results for the integrated system with ByteTrack, as shown in Table 4.2. The default settings of ByteTrack have been employed, and we aim to understand the effectiveness of the combined system for our specific fish detection and tracking application.

Table 4.2: Performance metrics for object tracking using default Bytetrack with Detectron2 as the detector. MOTA (Multiple Object Tracking Accuracy), MOTP (Multiple Object Tracking Precision), IDF1 (ID F1 Score), MT (Mostly Tracked), PT (Partially Tracked), and ML (Mostly Lost).

Metric	MOTA	MOTP	IDF1	Num Switches	MT	PT	ML
<i>Ch3<sub>default</sub></i>	0.603	0.207	0.588	111	28	28	5
<i>Ch6<sub>default</sub></i>	0.538	0.209	0.538	98	23	32	7

The performance metrics in Table 4.2 reveal that the default configuration of ByteTrack provides similar tracking performance across both ch3 and ch6 datasets. The slight differences in the MOTA and MOTP scores between the two datasets are small, indicating that ByteTrack maintains consistent tracking performance across different environments. The IDF1 scores are slightly higher for the ch3 dataset compared to ch6, suggesting a somewhat better performance in preserving identities in the ch3 dataset. This is an important aspect of the tracking performance, particularly for applications where maintaining consistent identities over time is critical.

Looking at the 'Mostly Tracked' (MT) and 'Partially Tracked' (PT) metrics, we observe comparable values across both datasets, which suggest successful tracking across most of the video sequences in both channels. The 'Mostly Lost' (ML) figures being low for both datasets are encouraging, indicating the algorithm's resilience against temporary tracking failures, which can occur due to occlusions or fish interactions.

Upon further examination of the tracking videos, it also becomes apparent that the model can handle occlusion to a certain extent. Nevertheless, maintaining the correct tracking ID proves inconsistent, particularly when severe occlusions occur for an extended period. Fig. 4.4 illustrates an example from Channel 3, where two fish with tracking IDs 6 and 1 are followed. In Fig. 4.4a, both fish are visible before fish 6 occludes fish 1 in Fig. 4.4b. However, when fish 1 re-emerges 60 frames (2.5 seconds) later, the tracking ID has changed to 185. On the other hand, the model successfully regains the correct tracking ID when occlusions are brief, as demonstrated in Fig. 4.5. In this instance, the two fish highlighted in Fig. 4.5a become occluded by another fish in Fig. 4.5b, but they manage to recover their initial tracking IDs in Frame 105, as shown in Fig. 4.5c.

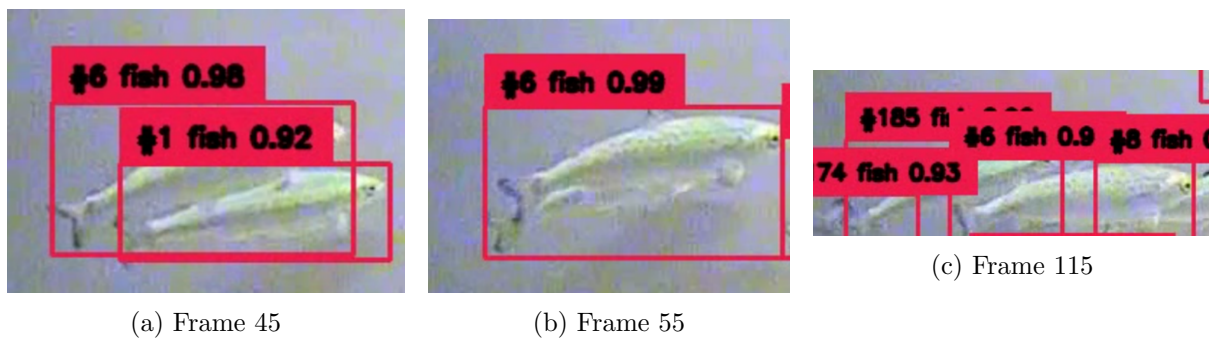


Figure 4.4: channel 3 Prolong occlusion lasting 60 frames (almost 2.5 seconds)

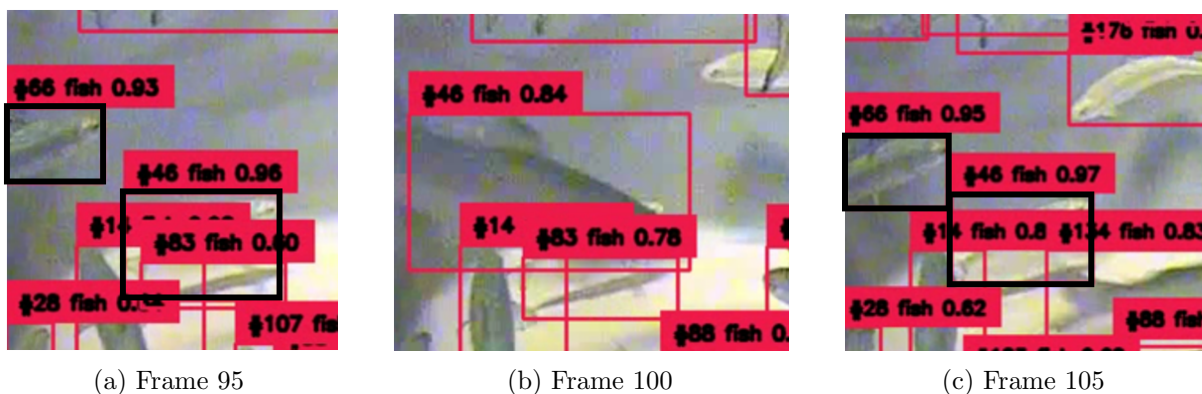


Figure 4.5: channel 3 Brief occlusion lasting 5 frames (20 milliseconds)

The model's tracking performance on the datasets underscores a number of challenges in tracking objects across different scenarios. While the model has demonstrated some capacity to handle occlusion, the inconsistency in maintaining the correct tracking ID during prolonged occlusions remains a significant issue. Additionally, issues such as identity switches and lost tracks become more prevalent under these conditions. These challenges suggest potential areas for improvement in our fish detection and tracking application, as explored through the algorithm tuning presented in Table 4.3.

Table 4.3: Comparison of tracking performance metrics for ByteTrack with Detectron2 on Channels 3 and 6, using various parameter configurations. The results demonstrate the impact of tuning parameters such as threshold (th) for high-scoring and low-scoring detections, Intersection over Union (IoU), and maximum frame gap for lost tracks (frame) on the tracking performance.

Metric	MOTA	MOTP	IDF1	Num Switches	MT	PT	ML
<i>Ch3<sub>default</sub></i>	0.603	0.207	0.588	111	28	28	5
<i>Ch3<sub>th0.25</sub></i>	-0.033	+0.002	-0.023	+49	+6	-4	-2
<i>Ch3<sub>th0.8</sub></i>	+0.044	-0.008	+0.036	-35	-10	+6	+4
<i>Ch3<sub>IoU0.25</sub></i>	-0.240	-0.015	-0.331	+1070	-22	+21	+1
<i>Ch3<sub>frame80</sub></i>	-0.003	-0.001	-0.000	+53	+6	-4	-2
<i>Ch3<sub>frame20</sub></i>	+0.002	0.000	+0.005	+46	+6	-4	-2
<i>Ch3<sub>IoU0.25Th0.8</sub></i>	-0.188	-0.029	-0.310	+496	-24	+11	+13
<i>Ch3<sub>IoU0.5Th0.5</sub></i>	-0.010	-0.008	-0.123	+241	-8	+8	0
<i>Ch6<sub>default</sub></i>	0.538	0.209	0.538	98	23	32	7
<i>Ch6<sub>th0.25</sub></i>	-0.024	+0.001	-0.010	+51	+1	0	-1
<i>Ch6<sub>th0.8</sub></i>	+0.001	-0.010	+0.022	-48	-9	0	+9
<i>Ch6<sub>IoU0.25</sub></i>	-0.260	-0.014	-0.348	+1094	-18	+11	+7
<i>Ch6<sub>frame80</sub></i>	-0.002	0.000	-0.008	+63	+1	0	-1
<i>Ch6<sub>frame20</sub></i>	+0.003	-0.000	+0.006	+35	+1	0	-1
<i>Ch6<sub>IoU0.25Th0.8</sub></i>	-0.213	-0.027	-0.330	+439	-20	+3	+17
<i>Ch6<sub>IoU0.5Th0.5</sub></i>	-0.033	-0.008	-0.115	+272	-5	+4	+1

After evaluating the tracker using different configuration values, it is observed that the performance does not improve substantially with the parameters mentioned above. While the variations in the MOTA, MOTP, and IDF1 scores are relatively minor compared to the default setup, a considerable fluctuation is observed in the number of identity switches. This fluctuation becomes particularly pronounced at lower IoU thresholds, where the system experiences a notable increase in these switches, indicating a decline in tracking performance. A reduction in the MOTA score further evidences this decline.

Further investigation of the videos resonates with these observations as there is little to no improvement in the tracking videos whilst still being unable to handle longer occlusions. Thus, tuned versions of this tracker still face the same problems as the default settings.

## 4.2 YOLOv8 and ByteTrack Performance

### 4.2.1 Detection Results

In this subsection, we analyze the training results of the YOLOv8 model, focusing on the key performance metrics and trends observed during the training process. Fig. 4.6 depicts the trajectories for validation loss, total loss, mAP50, and mAP50-95 metrics throughout the training process. As the number of epochs increases, both the validation loss and total loss exhibit a decreasing trend, signifying that the model is effectively learning from the training data.

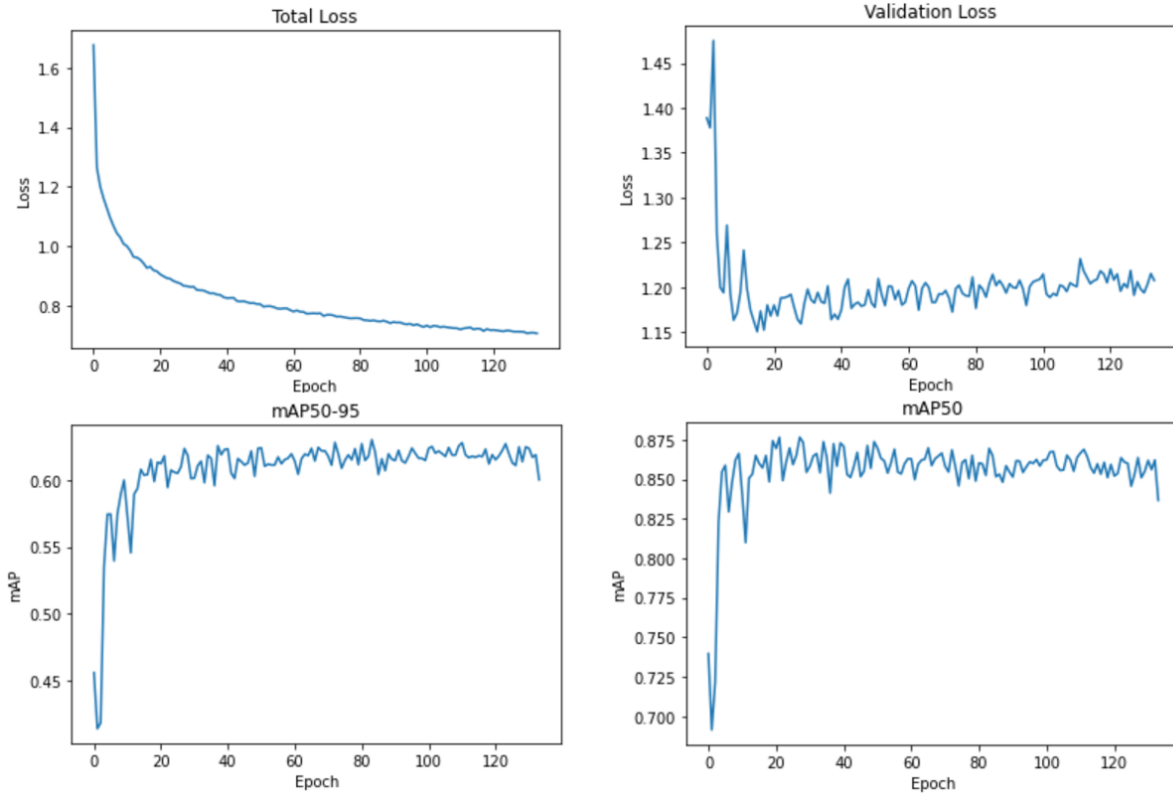


Figure 4.6: YOLOv8 training performance

As the training progresses, the mAP values increase before stabilising and fluctuating around 80% and 60%, suggesting the model is converging. The difference between the two mAP metrics suggests that the detector performs better when employing less stringent IoU thresholds. Furthermore, a gradual decrease in total loss is observed, while the validation loss increases after 20 epochs, suggesting potential overfitting. However, early stopping is implemented, halting the training process when no significant improvements in performance are detected over 50 consecutive epochs and saving the best-performing model to prevent overfitting. The optimal model, saved at epoch 84, is used for subsequent analysis. The performance of this model across different datasets is presented in Table 4.4.

Table 4.4: Mean Average Precision (mAP) for different IOU thresholds in training and testing for YOLOv8

Metric	Validation ch6	Test ch6	Test ch3
mAP50-90	0.630	0.619	0.617
mAP50	0.862	0.862	0.89
mAP75	-	-	-

The results generally indicate that the YOLOv8 model effectively generalises across all three datasets, as shown by the high similarity across different mAP thresholds. However, the lower mAP score indicates the model could improve its detection accuracy.

This is illustrated in Fig. 4.7, where the model exhibits relatively high confidence in areas with less occlusion. However, the model appears to struggle with detection in more complex areas, particularly in clusters of fish. In Table 4.5 associated with Fig. 4.7, we see that it only detects

23 out of 28 fish. When comparing this to the ground truth frame, the missed detections tend to occur in areas with more occlusion or when fish are exiting the frame. An example of one such missed detection can be found in the middle of the frame, as depicted in Fig. 4.7b.

Table 4.5: Number of Ground Truth and Predicted Fish Detected by YOLOv8 in Frame 1 in Ch3

Ground Trut	Predictions
28	23

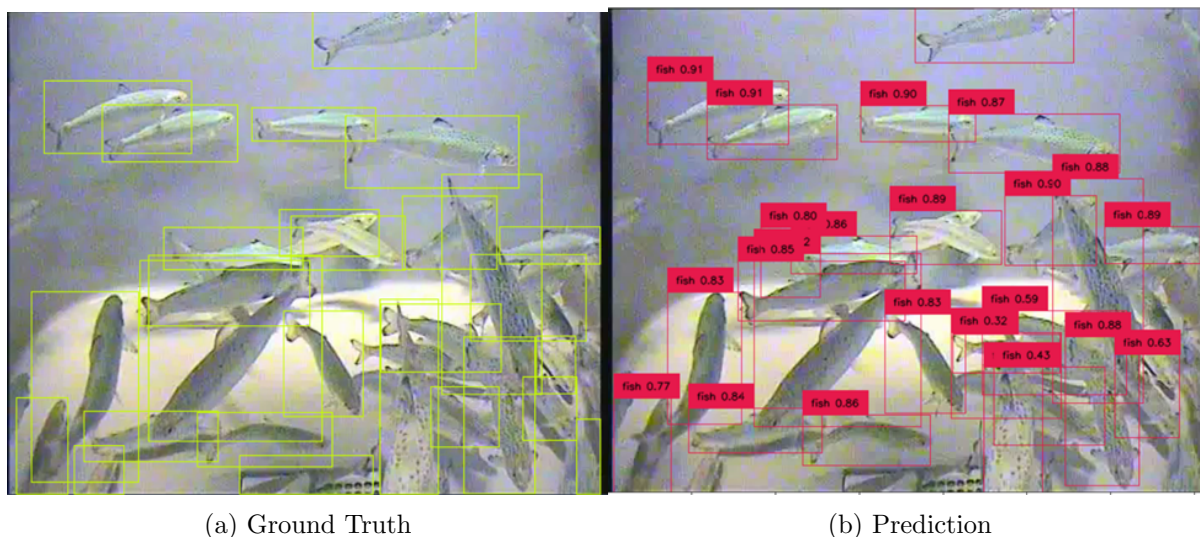


Figure 4.7: Comparison example using fram 1 of ground truth and detections in Ch3 test datasets.

With a deeper understanding of YOLOv8’s detection performance, we will employ this object detector with the ByteTrack algorithm for further object tracking. The aim is to create an integrated tracking system capable of robustly detecting and tracking fish in video sequences. The effectiveness of the YOLOv8 model in contributing to accurate and reliable fish tracking will be evaluated as part of this integrated system.

#### 4.2.2 Tracking Results

The object tracking performance metrics presented in Table 4.6 demonstrate the effectiveness of the YOLOv8 detector combined with the ByteTrack tracking algorithm on the two test datasets, Ch3 and Ch6. Both datasets exhibit similar MOTA, MOTP, and IDF1 scores, suggesting that the YOLOv8 model combined with ByteTrack maintains a consistent tracking accuracy and identity preservation level across different conditions.

Table 4.6: Performance metrics for object tracking using default ByteTrack

Metric	MOTA	MOTP	IDF1	Num Switches	MT	PT	ML
<i>Ch3<sub>default</sub></i>	0.657	0.168	0.597	93	25	30	6
<i>Ch6<sub>default</sub></i>	0.652	0.158	0.624	73	33	24	5

However, some differences emerge when we examine the number of identity switches. The Ch6 dataset has fewer identity switches than the Ch3 dataset, indicating a more stable tracking



performance. The difference in MT and PT instances compared to Ch3 further supports this observation. These differences can be ascribed to various factors, such as disparities in occlusion, fish behaviour alterations, or environmental conditions between the two datasets. Nevertheless, the results subtly indicate the necessity for further enhancements.

Further analyses of the videos also illustrate how the algorithm manages these challenges, such as severe and brief occlusions. Instances of these can be seen in Ch3, with examples showcased in Fig. 4.8 and Fig. 4.9.

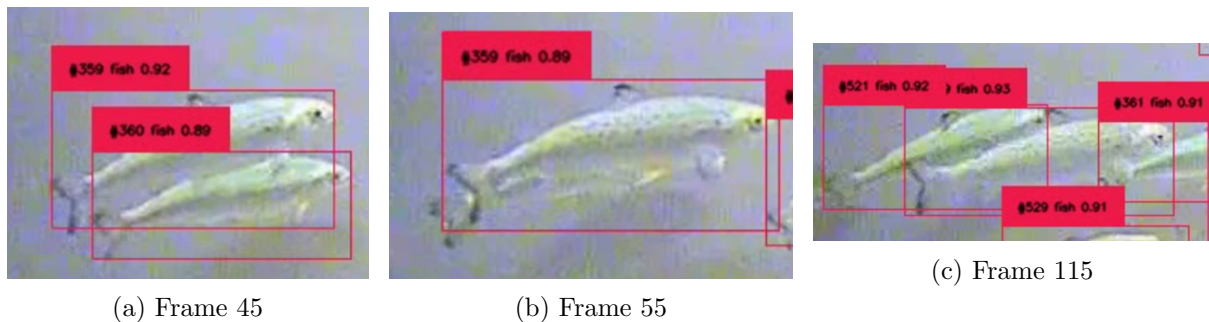


Figure 4.8: Example of Prolong occlusion lasting 60 frames (almost 2.5 seconds)

The prolonged occlusion scenario is depicted in Fig. 4.8 follows two fish in frame 45 (4.8a) where fish 359 fully occludes fish 360 at frame 55(4.8b) lasting for a duration of 60 frames, before finally reappearing in frame 115(4.8c)but with a new tracking ID.

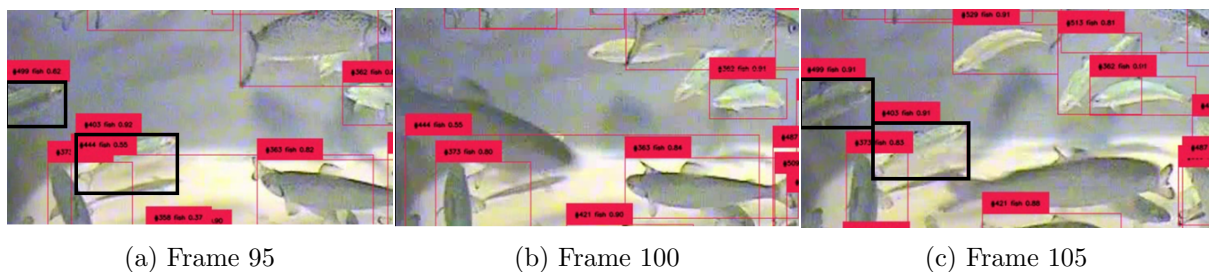


Figure 4.9: Example of brief occlusion lasting 5 frames (20 milliseconds)

The other example follows the highlighted fish in frame 95(4.9a), where both fish are fully occluded by a fast-moving fish in frame100(4.9b). However, both fish retain their original tracking IDs when fully visible in frame105(4.9c). Another notable observation is that YOLOv8 cannot detect or track this fast-moving fish.

To investigate the possibility of enhancing tracking performance, further tuning of the ByteTrack algorithm was carried out. The outcome of this tuning process is presented in Table 4.7, which displays the comparable differences between the fine-tuned tracker and the default settings for each dataset.

Table 4.7: Comparison of tracking performance metrics for ByteTrack with YOLOv8 on Channels 3 and 6, using various parameter configurations. The results demonstrate the impact of tuning parameters such as threshold (th) for high-scoring and low-scoring detections, Intersection over Union (IoU), and maximum frame gap for lost tracks (frame) on the tracking performance.

<b>Metric</b>	<b>MOTA</b>	<b>MOTP</b>	<b>IDF1</b>	<b>Num Switches</b>	<b>MT</b>	<b>PT</b>	<b>ML</b>
<i>Ch3<sub>default</sub></i>	0.657	0.168	0.597	93	25	30	6
<i>Ch3<sub>th025</sub></i>	0.000	+0.005	+0.013	+26	+3	0	-3
<i>Ch3<sub>th0.8</sub></i>	-0.106	-0.023	-0.027	-59	-14	+2	+12
<i>Ch3<sub>IoU0.25</sub></i>	-0.305	-0.021	-0.356	+965	-18	+10	+8
<i>Ch3<sub>frame80</sub></i>	-0.000	+0.000	+0.021	+26	+3	0	-3
<i>Ch3<sub>frame20</sub></i>	+0.001	+0.000	+0.007	+30	+3	0	-3
<i>Ch3<sub>IoU0.25Th0.8</sub></i>	-0.344	-0.051	-0.340	+129	-21	-5	+26
<i>Ch3<sub>IoU0.5Th0.5</sub></i>	-0.038	-0.008	-0.122	+166	-5	+3	+2
<i>Ch6<sub>default</sub></i>	0.652	0.158	0.624	73	33	24	5
<i>Ch6<sub>th025</sub></i>	-0.045	+0.003	-0.066	+77	+3	0	-3
<i>Ch6<sub>th08</sub></i>	-0.067	-0.015	+0.004	-31	-25	+13	+12
<i>Ch6<sub>IoU0.25</sub></i>	-0.287	-0.016	-0.361	+1057	-29	+25	+4
<i>Ch6<sub>frame80</sub></i>	-0.004	0.000	-0.008	+85	+3	0	-3
<i>Ch6<sub>frame20</sub></i>	0.000	-0.001	+0.001	+73	+2	+1	-3
<i>Ch6<sub>IoU0.25Th0.8</sub></i>	-0.342	-0.041	-0.352	+197	-32	+4	+28
<i>Ch6<sub>IoU0.5Th0.5</sub></i>	+0.055	+0.007	+0.103	+172	-11	+12	-1

The fine-tuning process of the tracking algorithm did not yield any significant enhancements in tracking performance. A notable adjustment was the modification of the detection threshold in ByteTrack from the default 0.6 to 0.8. This alteration led to a marginal decline in the overall performance, but concurrently resulted in a considerable reduction in the number of switches, decreased the count of MT, and raised the quantity of ML.

This outcome suggests that the algorithm has become more rigorous and may experience difficulties in tracking fish over longer durations. As a result, the remaining MT might primarily consist of shorter trajectories. The elevated threshold could potentially cause the model to lose track of fish more easily in complex situations, such as occlusions or abrupt changes in movement, leaving only short-duration trajectories as MT.

However, upon reviewing the video feeds, it is apparent that the increase in mostly lost can be attributed to fewer fish being detected, as illustrated in Fig4.10 and the associated Table 4.8. The heightened deviation induced by the higher threshold can complicate the task for the tracking algorithm to form associations with a higher detection.

Table 4.8: Average detection in Ch3

<b>Threshold (th)</b>	<b>Detections</b>
0.8	18
0.6	26

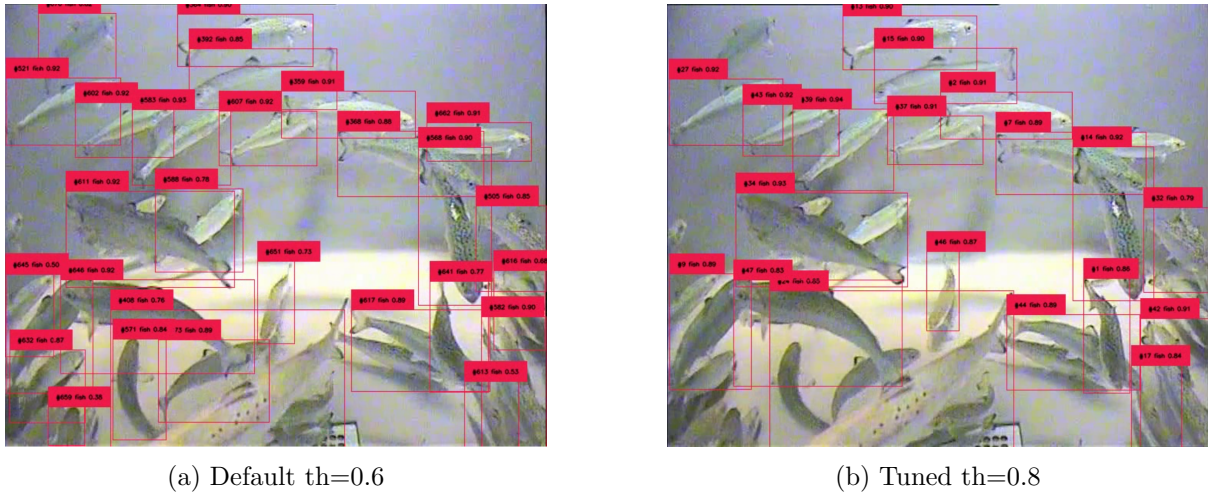


Figure 4.10: Detection comparison between two threshold parameters that decide high scoring and low scoring detections

An additional parameter modification of significance involves adjusting the IoU threshold from a higher default value of 0.8 to a lower threshold of 0.25. This modification led to a notable deterioration in performance across all evaluated metrics. This is due to the lower IoU allowing for low-scoring detections to be considered, thereby increasing the potential for identity switches.

Although the observations in the table do not suggest any significant alterations in performance when increasing the *frame* parameter, a closer analysis of the video reveals an enhanced ability to handle prolonged occlusions, as illustrated in Fig. 4.11. This figure presents the same example as Fig. 4.8 but with the adjusted frame parameter from 30 to 80. Contrary to the default tracker, the fine-tuned tracker can recover the correct tracking ID from frame 45(4.11a) in the subsequent frame 115(4.11c) after the prolonged occlusion.

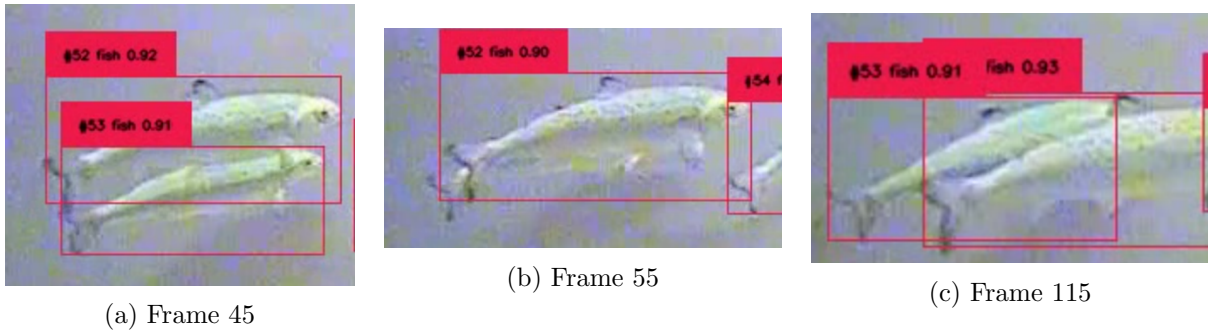


Figure 4.11: Prolong occlusion with lost track restoration at 80 frames

Insights from these observations underscore the critical role that parameter selection plays in achieving optimal tracking performance. Unsuitable combinations can significantly degrade the system’s efficacy. The results also highlight that fine-tuning the tracker can provide a valuable approach to overcoming the unique challenges encountered when tracking fish in these environments.

However, it’s important to note that fine-tuning doesn’t necessarily guarantee improved performance across all metrics. The default tracker configuration still demonstrates the most robust results overall. Yet, there is a compelling case for the tracker configuration with a frame buffer

of 80. This configuration exhibits an enhanced ability to handle prolonged occlusions, a common challenge in the given scenario.

Despite the advantages, increasing the frame buffer size is not without its potential pitfalls. While it provides the system with a greater ability to maintain object tracking during extended occlusions, it also increases the probability of more mismatches or false positives. This is because the tracker has a wider window of frames to consider when trying to re-identify an occluded object, which can lead to an incorrect trajectory or identity switch.

### 4.2.3 Comparative analysis

This thesis aims to present a tracking model capable of being the foundation for fish welfare applications through computer vision. We have presented and analysed two distinct frameworks for achieving this goal. The first framework combines the object detection capabilities of Detectron2 with the tracking performance of ByteTrack, while the second framework utilises YOLOv8 as the object detector and ByteTrack for tracking purposes. Having individually discussed the performance of each framework in terms of detection and tracking, we will now compare the results of both frameworks. This comparison will enable us to determine which of the two proposed solutions offers the most promising performance in underwater fish tracking applications.

#### Detection

The detection results from the two frameworks are presented in Table 4.9. The two detectors demonstrated varying performance across the two channels, with YOLOv8 outperforms Detectron2 in terms of the mean average precision (mAP) across all datasets and IOU thresholds. Looking at the mAP50-95 and mAP50 metrics, YOLOv8 consistently yields higher scores. The consistency in the performance of YOLOv8 across different datasets and thresholds suggests that YOLOv8 is more precise in detecting fish and maintains this precision regardless of changes in the dataset. This consistency makes YOLOv8 more reliable for detection tasks in different environmental conditions.

On the other hand, Detectron2's performance, especially in the mAP50 metric, drops significantly in the test ch6 dataset. This indicates that certain conditions or attributes might negatively influence Detectron2's ability to detect fish in this dataset. This variability in Detectron2's performance poses a risk when the model is deployed in a real-world setting where data variations are expected.

Table 4.9: Mean Average Precision (mAP) for different IOU thresholds in training and testing for Detectron2 and YOLOv8

Detector	Metric	Validation ch6	Test ch6	Test ch3
Detectron2	mAP50-95	0.431	0.395	0.462
	mAP50	0.778	0.687	0.769
YOLOv8	mAP50-90	<b>0.63</b>	<b>0.619</b>	<b>0.617</b>
	mAP50	<b>0.862</b>	<b>0.862</b>	<b>0.89</b>

To further investigate the cause of this performance difference, we also examined the detections produced by both frameworks in the tank and compared them to the ground truth. The detections for each framework are visualised in Fig. 4.14 with Table 4.10 showing the average detections for the videos.

Table 4.10: Average detection results and mAP scores from Detectron2 and YOLOv8 for the two datasets compared with the ground truth

Source	Dataset	Average Detections	mAP50-95	mAP50
Ground Truth	Ch3	30	-	-
	Ch6	29	-	-
Detectron2	Ch3	29	0.462	0.769
	Ch6	27	0.395	0.687
YOLOv8	Ch3	24	0.617	0.89
	Ch6	26	0.619	0.862

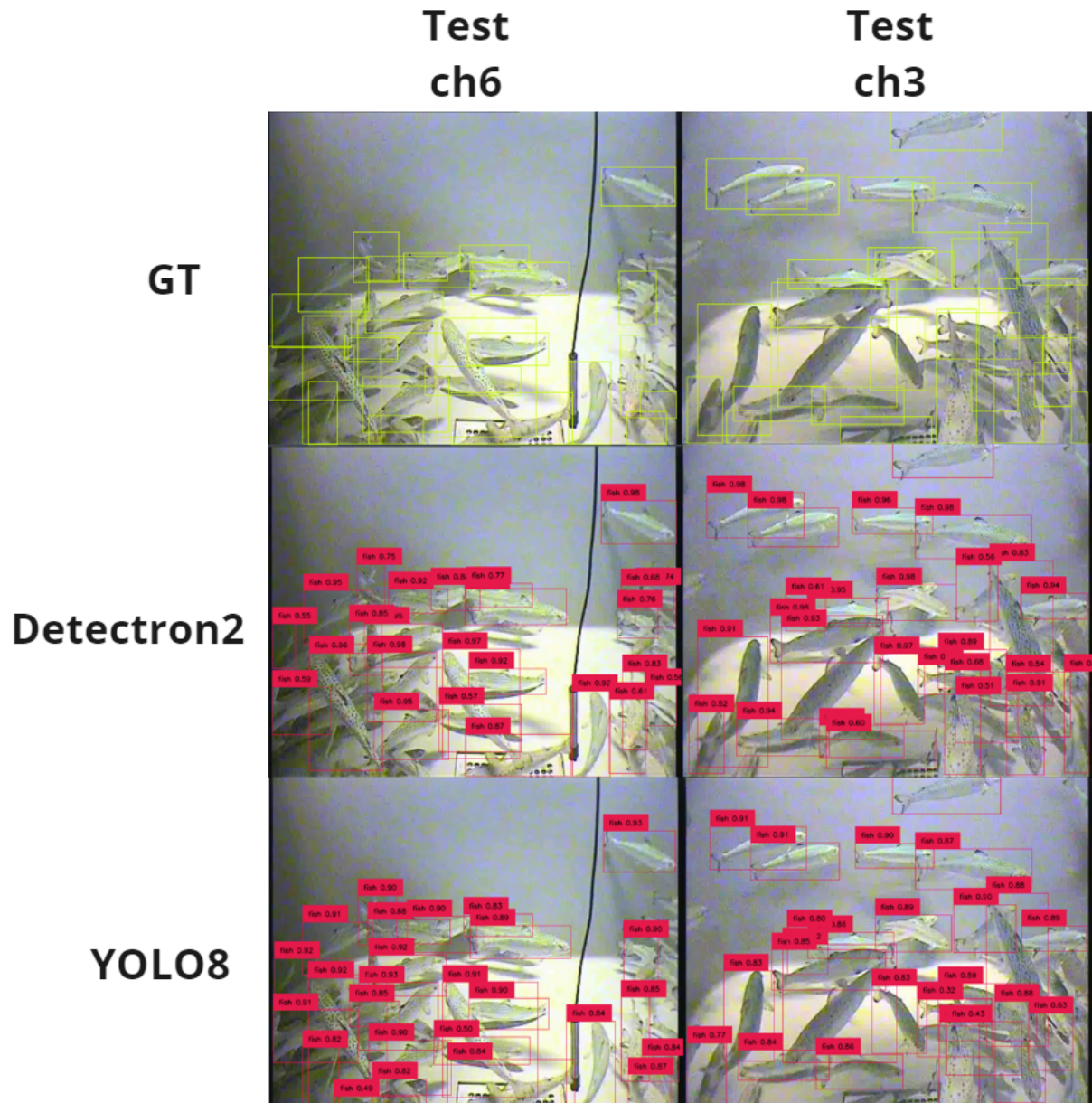


Figure 4.12: Detection results from Detectron2 and YOLOv8 compared to the ground truth of the same frame

Detectron2 detects more fish and demonstrates higher confidence in detections when the fish is

entirely visible, but some of these detections are false positives. On the other hand, YOLOv8 performs consistently and maintains high confidence even in areas with significant occlusions. Despite some high confidence values in Detectron2’s detections, the comprehensive scores in Table 4.9 confirm that YOLOv8 consistently outperforms Detectron2 in detection accuracy. Thus, YOLOv8 is the superior choice for detecting fish in this context.

Various factors may contribute to the observed differences between the two detectors. Both frameworks are based on PyTorch, but significant distinctions may lie in the underlying architectures and data augmentation strategies, leading to variations in performance.

Detectron2 employs an anchor-based approach that might be more sensitive to fluctuations in object size and aspect ratios. This could lead to less stable performance when dealing with footage where fish appearances and sizes drastically change. On the other hand, YOLOv8 employs an anchor-free method, which might offer greater adaptability to changes in object characteristics. This adaptability could account for its more consistent performance across different datasets where such drastic changes occur.

Furthermore, YOLOv8 employs mosaic augmentation, effectively increasing the diversity of object appearances, sizes, and orientations within a single training instance. Consequently, this augmentation strategy may enable YOLOv8 to learn more robust features, ultimately enhancing its detection performance across various scenarios. On the other hand, Detectron2 relies on conventional data augmentation techniques, such as image brightening, flipping, and cropping. While these methods can improve the framework’s ability to generalise to different conditions, they may not provide the same diversity and complexity as mosaic augmentation, potentially resulting in less consistent detection performance across diverse underwater environments.

With the superior detection performance of YOLOv8 and an understanding of the possible reasons behind the observed differences, we will compare the tracking performance of the two frameworks. By examining the tracking results, we can determine the best approach for underwater fish tracking applications, considering both detection and tracking aspects.

## Tracking

Having established that YOLOv8 demonstrates a better detection performance, we then move to compare the tracking performance of the two detection models, both paired with the default ByteTrack settings on the ch3 and ch6 test datasets. We aim to examine if the underlying detector impacted the overall tracking performance and to determine the most suitable combination for underwater fish tracking applications.

Table 4.11 presents the performance metrics for both the trackers with the best results highlighted. The table shows that the tracker paired with the YOLOv8 detector consistently outperforms the one paired with Detectron2 across all key metrics, such as MOTA, MOTP, and IDF1.

Detector	Dataset	MOTA	MOTP	IDF1	Num Switches	MT	PT	ML
Detectron2	Ch3	0.603	0.207	0.588	111	<b>28</b>	28	5
	Ch6	0.538	0.209	0.538	98	23	32	7
YOLOv8	Ch3	<b>0.657</b>	<b>0.168</b>	<b>0.597</b>	<b>93</b>	25	30	6
	Ch6	<b>0.652</b>	<b>0.158</b>	<b>0.624</b>	<b>73</b>	<b>33</b>	24	5

Table 4.11: Performance metrics for object tracking using YOLOv8 and Detectron2 with default ByteTracker on Ch3 and Ch6 test datasets

A more in-depth analysis of the tracking videos further highlights the performance differences between the two trackers. This is exemplified in Fig. 4.14 and Fig. 4.13, where each figure consists of three frames that are zoomed in to examine an occlusion scenario. The sequence follows the detections highlighted with black bounding boxes from Frame 95 to Frame 100, during which another moving fish occludes both fish and finally to Frame 105, where both fish become visible again.

In Frame 100 (4.13b), the tracker utilizing YOLOv8 fails to detect the fast-moving fish. However, the highlighted detections regain their tracking IDs in the subsequent Frame 105 (4.13c).

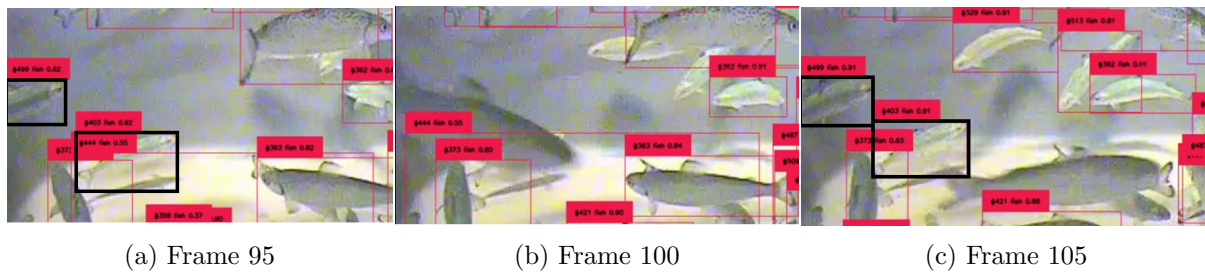


Figure 4.13: ByteTrack with YOLOv8 on Ch3

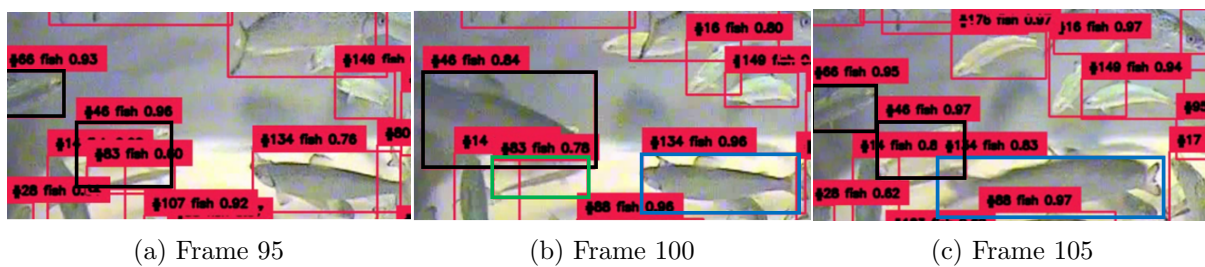


Figure 4.14: ByteTrack with Detectron2 on Ch3

In contrast, the tracker incorporating Detectron2 successfully detects the fast-moving fish and recovers the correct tracking IDs for the highlighted fish from Frame 95 to Frame 105. However, upon closer examination of Fig. 4.14, it becomes apparent that the detected fish in Frame 100 (4.14b) adopts the tracking ID from one of the highlighted fish in Frame 95 (4.14a). This ID switch occurs between Frame 100 and Frame 105, as the tracking ID linked to the blue bounding box portrayed in Frame 100 (4.14b) shifts to the fast-moving fish in Frame 105 (4.14c). Furthermore, the tracker paired with Detectron2 seems to follow the shadow of another fish, most noticeably visible in Frame 100 (4.14b), as depicted by the green bounding box. These tendencies could correspond with Detectron2's detection performance, which is earlier in detecting fish than YOLOv8. This is further accentuated in Fig. 4.15, where the tracker paired with Detectron2 can detect fish earlier than the tracker paired with YOLOv8. However, these earlier detections appear to lead to incorrect tracking ID assignments and false positive detections, as observed in Fig. 4.14b.



(a) YOLOv8



(b) Ground Truth



(c) Detectron2

Figure 4.15: Tracking Detection compared with ground truth on Frame 92 Ch3

The preceding analysis of both trackers reveals that the default configurations exhibit challenges in handling occlusions over extended durations. Further adjustments to ByteTrack’s parameters demonstrated favourable outcomes exclusively for the tracker associated with YOLOv8. Apart from this, no additional tuning appears to impact either tracker favourably. Consequently, this suggests that the disparities in tracking performance predominantly arise from the inherent characteristics of the respective detector models. When paired with ByteTrack, this combination also performed better across the board, being tuned to handle occlusion scenarios better. Conversely, Detectron2, despite having faster detections, tended to produce more false positive detections and incorrect tracking ID assignments, negatively impacting its tracking performance by introducing more uncertainty. Therefore, pairing ByteTrack and YOLOv8 is better for achieving accurate fish tracking in underwater environments.



## Evaluation and Future Directions

The preceding chapter established that the most effective framework incorporated YOLOv8 as the detection component and ByteTracker as the tracking aspect. Although this framework presented the best results of the two combinations tested, its performance still needs to be addressed before a suitable tracker for the targeted objective is attained. This chapter will examine potential enhancements to the framework that could further augment its performance, which could not be implemented due to time constraints. Additionally, alternative approaches that may hold promise and contribute to tracking applications will be discussed.

The tracking combination exhibited notable performance, effectively detecting and maintaining tracks in the face of various challenges associated with fish tracking. Nonetheless, there remains room for further improvement in the framework. As mentioned in the Methods chapter, subsection 3.4, the datasets had a resolution of 576p, which, while adequate, might be considered subpar given the increased availability of higher resolutions. This resolution limitation could contribute to the framework's difficulty in handling ID switches, as it complicates the distinction between fish and their background or other fish. A higher resolution would enable a better representation of environmental features and fish characteristics and potentially enhance the detection of YOLOv8.

The pre-trained architecture has undergone adaptation, refinement, and evaluation using datasets featuring fish from controlled experimental tanks, which may potentially limit its generalizability to other real-world settings. Consequently, the model could encounter difficulties adapting to situations involving varied environments or distinct camera perspectives. To address this limitation, supplementing the dataset with more diverse data may enhance the model's capacity to cope with a broader range of scenarios, thereby mitigating the potential influence of dataset bias.

Additionally, it was observed that the detection models significantly impacted the tracking algorithms' capacity to localise and monitor objects accurately. This observation suggests that further hyperparameter tuning of the detector might improve performance. Another alternative approach to consider involves exploring different detection models, potentially contributing to improving the overall effectiveness of the tracking framework.

While the ByteTrack algorithm demonstrated commendable performance in handling fish-tracking scenarios, there were certain challenges to which it did not adequately adapt. Even experimenting with tuning parameters, no significant performance improvements were observed. This situation prompts us to question the potential limitations of the tracking algorithm.

A concurrent master thesis investigating the same objective using identical data utilized YOLOv8 with the same parameters configuration as the detector but paired it with the DeepSort Tracking algorithm. The primary distinctions between DeepSort and ByteTrack lie in their respective tracking methodologies. DeepSort employs a combination of appearance and motion features for tracking, emphasising the object’s appearance while concurrently filtering out low-scoring detection boxes. ByteTracker does not filter out this detection using high-scoring and low-scoring detection boxes.

The tracking results derived from both algorithms are presented in Table 5.1, demonstrating that ByteTrack consistently outperforms DeepSort across both datasets. This finding suggests that ByteTrack’s tracking method may better suit these environments than DeepSort. However, there could still be another tracker methodology that could still be better suited for the challenge of fish tracking. A potential tracker that could be adopted could be the inbuilt tracker of YOLOv8, which was added to the model too late for us to explore its potential.

Table 5.1: Comparative Analysis of ByteTrack and DeepSort Trackers Using the Same Detection Model (YOLOv8)

Tracker	Dataset	MOTA	MOTP	IDF1	Num Switches	MT	PT	ML
ByteTrack	Ch3	<b>0.657</b>	<b>0.168</b>	<b>0.597</b>	<b>93</b>	<b>25</b>	30	6
	Ch6	<b>0.652</b>	<b>0.158</b>	<b>0.624</b>	<b>73</b>	<b>33</b>	24	5
DeepSort	Ch3	0.598	0.192	0.455	188	25	30	6
	Ch6	0.556	0.185	0.466	213	30	24	5

Another potential avenue to explore could be using transformers[49] to enhance the tracking system. With their ability to model global dependencies and self-attention mechanisms, transformers could capture the relationships between distant parts of the image and weigh the importance of different parts of the image differently. This could help improve performance, better adapt to diverse environments, and address the limitations observed in the current framework, particularly in complex situations such as occlusions. However, it’s important to note that transformers typically require a large amount of training data to perform optimally, which might be a consideration depending on the dataset’s size and diversity available for training

Only when the framework can overcome its challenges and is deemed efficient enough to serve as a groundwork tracker for further welfare applications can it truly be considered a viable solution for monitoring aquatic species. If these criteria are met, the framework could be employed for various applications, such as monitoring fish behaviour, assessing the health of aquatic populations, and informing sustainable management practices in aquaculture and conservation.

## Conclusion

The main objective of this thesis was to build a tracking application capable of being the foundation for future fish welfare applications. Our approach involved pairing two distinct detection frameworks, Detectron2 and YOLOv8, with the ByteTrack algorithm. Both models were trained, validated, and tested using the same dataset, which comprised videos of fish from Nofima's research tanks.

The detection models demonstrated effective performance on the datasets, with YOLOv8 surpassing Detectron2 in detection performance across various mAP thresholds. Subsequently, the tracking performance was assessed by pairing each detector with the ByteTrack algorithm. In this evaluation, the ByteTracker combined with YOLOv8 outperformed the version linked to Detectron2 for both tested datasets.

Nonetheless, there is potential for further improvement in both models. Future research could explore fine-tuning hyperparameters of the detector, incorporating additional training data, or experimenting with alternative network architectures to enhance tracking and detection capabilities. Moreover, investigating other tracking algorithms could yield a better result than we obtained in challenging environments.

In conclusion, while the detection performance achieved in this study can be considered satisfactory, the tracking performance still requires further refinement. The combination of YOLOv8 and ByteTrack demonstrated the best performance among the tested models, but additional work is needed to overcome the existing tracking challenges. This work serves as a preliminary step towards the development of advanced and accurate fish tracking systems, ultimately contributing to improved fish welfare monitoring and management.

# Bibliography

- [1] Ifad Fao, W UNICEF, et al. “Wfp, Who”. In: *The state of food security and nutrition in the world 2340* (2020).
- [2] Naveen Kumar Arora and Isha Mishra. “United Nations Sustainable Development Goals 2030 and environmental sustainability: race against time”. In: *Environmental Sustainability 2.4* (2019), pp. 339–342.
- [3] Annie Sturesson, Nina Weitz, and Åsa Persson. “Sdg 14: Life below water”. In: *A Review of Research Needs. Technical annex to the Formas report ForskningfÅ r Agenda 2030* (2018).
- [4] Shreesha S et al. “Computer Vision Based Fish Tracking And Behaviour Detection System”. In: *2020 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*. 2020, pp. 252–257. DOI: 10.1109/DISCOVER50404.2020.9278101.
- [5] Dong An et al. “Application of computer vision in fish intelligent feeding system—A review”. In: *Aquaculture Research* 52.2 (2021), pp. 423–437.
- [6] Yuxin Wu et al. *Detectron2*. <https://github.com/facebookresearch/detectron2>. 2019.
- [7] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *YOLO by Ultralytics*. Version 8.0.0. Jan. 2023. URL: <https://github.com/ultralytics/ultralytics>.
- [8] Yifu Zhang et al. “ByteTrack: Multi-Object Tracking by Associating Every Detection Box”. In: (2022).
- [9] Stuart Russell. *Artificial Intelligence: A Modern Approach, eBook, Global Edition*. Pearson Education, Limited, 2016.
- [10] IBM. *What is artificial intelligence (AI)?* 2020. URL: <https://www.ibm.com/topics/artificial-intelligence>.
- [11] Eda Kavlakoglu. *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What’s the Difference?* 2020. URL: <https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks>.
- [12] IBM. *What is machine learning?* URL: <https://www.ibm.com/topics/machine-learning>.
- [13] Sebastian Raschka and Vahid Mirjalili. “Python Machine Learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2”. In: 2 (2019). DOI: <https://doi.org/10.1002/nav.3800020109>.

- 
- [14] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. “Supervised Learning”. In: *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*. Ed. by Matthieu Cord and Pádraig Cunningham. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 21–49. ISBN: 978-3-540-75171-7. DOI: 10.1007/978-3-540-75171-7\_2. URL: [https://doi.org/10.1007/978-3-540-75171-7\\_2](https://doi.org/10.1007/978-3-540-75171-7_2).
- [15] Zoubin Ghahramani. “Unsupervised Learning”. In: *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*. Ed. by Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 72–112. ISBN: 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9\_5. URL: [https://doi.org/10.1007/978-3-540-28650-9\\_5](https://doi.org/10.1007/978-3-540-28650-9_5).
- [16] Jun Han and Claudio Moraga. “The influence of the sigmoid function parameters on the speed of backpropagation learning”. In: *From Natural to Artificial Neural Computation*. Ed. by José Mira and Francisco Sandoval. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 195–201. ISBN: 978-3-540-49288-7. DOI: [https://doi.org/10.1007/3-540-59497-3\\_175](https://doi.org/10.1007/3-540-59497-3_175).
- [17] Kuniyuki Fukushima. “Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements”. In: *IEEE Transactions on Systems Science and Cybernetics* 5.4 (1969), pp. 322–333. DOI: 10.1109/TSSC.1969.300225.
- [18] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feed-forward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterton. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: <https://proceedings.mlr.press/v9/glorot10a.html>.
- [19] Yann LeCun et al. “Handwritten digit recognition with a back-propagation network”. In: *Advances in neural information processing systems* 2 (1989).
- [20] Thomas Serre, Lior Wolf, and Tomaso Poggio. “Object recognition with features inspired by visual cortex”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 2. Ieee. 2005, pp. 994–1000.
- [21] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks”. In: *CoRR* abs/1311.2901 (2013). DOI: <https://doi.org/10.48550/arXiv.1311.2901>. arXiv: 1311.2901. URL: <http://arxiv.org/abs/1311.2901>.
- [22] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.
- [23] Mark Everingham et al. “The pascal visual object classes (voc) challenge”. In: *International journal of computer vision* 88 (2010), pp. 303–338.
- [24] MabelAmber. *cat\_pet\_Feline*. 2018. URL: <https://pixabay.com/photos/cat-pet-feline-animal-fur-3591348/>.
- [25] huoadg5888. *Pets\_cats\_dogs*. 2018. URL: <https://pixabay.com/photos/pets-cat-dog-animals-mammals-3715733/>.
- [26] Ross Girshick. “Fast R-CNN”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.
- [27] Glenn Jocher et al. *ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation*. Version v7.0. Nov. 2022. DOI: 10.5281/zenodo.7347926. URL: <https://doi.org/10.5281/zenodo.7347926>.
- [28] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: <http://arxiv.org/abs/1405.0312>.

- 
- [29] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: <http://arxiv.org/abs/1506.01497>.
- [30] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: <https://doi.org/10.48550/arXiv.1512.03385>. arXiv: 1512.03385 [cs.CV].
- [31] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2017. DOI: <https://doi.org/10.48550/arXiv.1612.03144>. arXiv: 1612.03144 [cs.CV].
- [32] Ross B. Girshick. “Fast R-CNN”. In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083. URL: <http://arxiv.org/abs/1504.08083>.
- [33] RangKing. *Model\_structure\_of\_YOLOv8*. 2023. URL: <https://github.com/ultralytics/ultralytics/issues/189>.
- [34] Chien-Yao Wang et al. “CSPNet: A New Backbone that can Enhance Learning Capability of CNN”. In: *CoRR* abs/1911.11929 (2019). arXiv: 1911.11929. URL: <http://arxiv.org/abs/1911.11929>.
- [35] Kaiming He et al. “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.9 (2015), pp. 1904–1916. DOI: 10.1109/TPAMI.2015.2389824.
- [36] Shu Liu et al. “Path Aggregation Network for Instance Segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.
- [37] Alper Yilmaz, Omar Javed, and Mubarak Shah. “Object tracking: A survey”. In: *Acm computing surveys (CSUR)* 38.4 (2006), 13–es.
- [38] Rudolph Emil Kalman. “A new approach to linear filtering and prediction problems”. In: (1960). DOI: <https://doi.org/10.1115/1.3662552>.
- [39] Harold W Kuhn. “The Hungarian method for the assignment problem”. In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97. DOI: <https://doi.org/10.1002/nav.3800020109>.
- [40] Zheng Ge et al. “Yolox: Exceeding yolo series in 2021”. In: *arXiv preprint arXiv:2107.08430* (2021). DOI: <https://doi.org/10.48550/arXiv.2107.08430>.
- [41] Seyed Hamid Rezaatofighi et al. “Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression”. In: *CoRR* abs/1902.09630 (2019). arXiv: 1902.09630. URL: <http://arxiv.org/abs/1902.09630>.
- [42] Keni Bernardin and Rainer Stiefelhagen. “Evaluating multiple object tracking performance: the clear mot metrics”. In: *EURASIP Journal on Image and Video Processing* 2008 (2008), pp. 1–10.
- [43] Ergys Ristani et al. “Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking”. In: *CoRR* abs/1609.01775 (2016). arXiv: 1609.01775. URL: <http://arxiv.org/abs/1609.01775>.
- [44] Christoph Heindl, Toka, and Jack Valmadre. *py-motmetrics*. <https://github.com/cheind/py-motmetrics>. Version 1.4.0. 2023.
- [45] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: <http://arxiv.org/abs/1405.0312>.
- [46] Dwyer B, Nelson J, and Solawetz J. *Roboflow [Software]*. <https://roboflow.com/>. Version 0.1.0. 2022.
- [47] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. In: *CoRR* abs/2004.10934 (2020). arXiv: 2004.10934. URL: <https://arxiv.org/abs/2004.10934>.

- 
- [48] Roboflow. *Supervision*. <https://github.com/roboflow/supervision>. Version 0.1.0. 2023-01-19.
- [49] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. “Vision Transformers for Dense Prediction”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 12179–12188.

# Appendix **A**

## Tables of Detection and Tracking Dependencies

### A.1 Detectron2 Dependencies

Package name	Version	Purpose of use
cloudpickle	2.2.1	Extended pickling support
docutils	0.16	Documentation utilities for reStructuredText
future	0.18.3	Compatibility layer between Python 2 and 3
fvcore	0.1.5	Facebook AI Research core library
hydra-core	1.3.2	Configuration management and composition
matplotlib	3.7.1	Plotting library
numpy	1.22.4	Numerical computing and array manipulation
omegaconf	2.3.0	Configuration management
Pillow	8.4.0	Image processing and manipulation
scipy	1.10.1	Scientific computing library
sphinx	3.2.0	Documentation generator
tabulate	0.8.10	Pretty-print tabular data
termcolor	2.2.0	Terminal output coloring
torch	2.0.0	Deep learning framework
torchvision	0.15.1	Pre-trained models for computer vision, Data data augmentation and image preprocessing
tqdm	4.65.0	Progress bar and performance metrics
yacs	0.1.8	Configuration management

Table A.1: Detectron2 dependencies with their respective versions and purpose of use



---

## A.2 YOLOv8 Dependencies

Package name	Version	Purpose of use
ipython	7.34.0	Interactive Python shell
matplotlib	3.7.1	Plotting library
numpy	1.22.4	Numerical computing and array manipulation
opencv-python	4.7.0.72	Image processing and computer vision library
pandas	1.5.3	Data manipulation and analysis
Pillow	8.4.0	Image processing and manipulation
psutil	5.9.5	System monitoring and management
PyYAML	6.0	YAML parsing and serialization
requests	2.27.1	HTTP library for making requests
roboflow	1.0.5	Dataset management and versioning
scikit-learn	1.2.2	CoreML quantization
scipy	1.10.1	Scientific computing library
seaborn	0.12.2	Data visualization
tensorflow	2.12.0	Deep learning framework
tensorboard	2.12.2	Logging
thop	0.1.1	FLOPs and parameter counter for PyTorch
torch	2.0.0	Deep learning framework
torchvision	0.15.1	Pre-trained models for computer vision, Data data augmentation and image preprocessing
tqdm	4.65.0	Progress bar and performance metrics
pycocotools	2.0.6	COCO evaluation

Table A.2: YOLOv8 dependencies with their respective versions and purpose of use

---

### A.3 ByteTrack Dependencies

Python name	package	Version	Purpose of use
	filterpy	1.4.5	Kalman filters and other optimal estimation techniques for object tracking
	h5py	3.8.0	Interface to the HDF5 binary data format
	lap	0.4.0	Linear assignment problem solver for data association in tracking
	loguru	0.7.0	Logging
	motmetrics	1.4.0	Evaluation metrics for object tracking
	ninja	1.11.1	Build system for faster compilation
	NumPy	1.22.4	Numerical computing and array manipulation
	onnx	1.9.0	Open Neural Network Exchange format
	onnxruntime	1.8.0	Inference engine for ONNX models
	onnx-simplifier	0.3.5	Simplify and optimize ONNX models
	opencv-python	4.7.0.72	Image processing
	Pillow	8.4.0	Image processing and manipulation
	scikit-image	0.19.3	Image processing and manipulation
	supervision	0.1.0	Connecting Object detection and Tracking
	tabulate	0.8.10	Pretty-print tabular data
	tensorboard	2.12.2	Visualization and debugging for machine learning
	thop	0.1.1	FLOPs and parameters counter for PyTorch models
	Torch	2.0.0	General-purpose deep learning framework for object tracking
	tqdm	4.65.0	Progress bar and performance metrics
	torchvision	0.15.1	Pre-trained models for computer vision, Data data augmentation and image preprocessing

Table A.3: ByteTrack dependencies with their respective versions and purpose of use

# Appendix **B**

## Parameters

### B.1 Detectron2 parameters

Parameter	Value	Purpose of use
BASE_LR	0.02	Sets the initial learning rate for the training process.
Batch_size	16	Specifies the number of images per batch across all machines.
gamma	0.1	Used as a multiplicative factor to decrease learning rate at specified steps.
Max_Iter	6000	Defines the total number of iterations (or the number of data batches) the model will be trained.
Optimizer	SGD	Specifies the optimization algorithm to be used during the training.
RandomBrightness	0.8	Sets the intensity for random brightness data augmentation.
RandomLighting	0.7	Sets the intensity for random lighting data augmentation.
Regions Per Image	512	Defines the number of region proposals per image.
RndFlip_horizontal	True	Enables horizontal random flipping for data augmentation.
RndFlip_vertical	True	Enables random vertical flipping for data augmentation.
SOLVER.STEPS	(4000, 5400)	Sets the points at which the learning rate is decreased.

Table B.1: Parameters and their respective values used in Detectron2

---

## B.2 YOLOv8 parameters

Parameter	Value	Purpose of use
batch	16	Specifies the number of training samples to work through before the model's internal parameters are updated.
close_mosaic	10	Determines the boundary conditions for the mosaic augmentation.
epochs	600	Defines the number times that the learning algorithm will work through the entire training dataset.
flipud	0.0	Specifies the probability of flipping the image up-down for data augmentation.
fliplr	0.5	Specifies the probability of flipping the image left-right for data augmentation.
hsv_h	0.015	Defines the hue adjustment for HSV (Hue, Saturation, Value) data augmentation.
hsv_s	0.7	Defines the saturation adjustment for HSV data augmentation.
hsv_v	0.4	Defines the value adjustment for HSV data augmentation.
imgsz	800	Specifies the size of the input image.
iou	0.7	Determines the Intersection over Union threshold for deciding true positives and false positives.
lr0	0.01	Specifies the initial learning rate.
lrf	0.01	Specifies the final learning rate.
mixup	0.5	Defines the mixup image augmentation usage probability.
mosaic	1.0	Specifies the mosaic image augmentation usage probability.
optimizer	SGD	Specifies the optimization algorithm used for training.
patience	50	Determines the number of epochs to wait before early stopping if no progress is made on the validation set.

Table B.2: Parameters and their respective values used in YOLOv8

---

### B.3 ByteTrack parameters

Parameter	Value	Purpose of use
track_thresh	0.6	Sets the detection score threshold that divides high score detection from low score detections.
track_buffer	30	Specifies how many frames a lost track should be stored in memory before being deleted.
match_thresh	0.8	Sets the IoU threshold between detection and predictions.

Table B.3: Parameters and their respective values used in Byte Tracker



**Norges miljø- og biovitenskapelige universitet**  
Noregs miljø- og biovitenskapelige universitet  
Norwegian University of Life Sciences

Postboks 5003  
NO-1432 Ås  
Norway