Norwegian University
of Life Sciences

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

Computer Vision for Fish Monitoring: Challenges and Possibilities

Mikal Breiteig
MSc Data Science

Preface

This thesis finalizes my master's degree at the Norwegian University of Life Science (NMBU). The work on this thesis started in January 2023 and the entire experience has been enriching, educational, and memorable.

First and foremost, I would like to express my heartfelt appreciation to my supervisor, Prof. Kristian Liland, and my co-supervisors, PhD. Lars Erik Solberg and PhD. Santhosh Kumaran. Their guidance, expertise, and unwavering support throughout the entire process have been invaluable in shaping the outcome of this thesis. Additionally, I would also like to express my deep appreciation to Jens Kristian Røed Holmboe for the engaging discussions that have been a great source of motivation.

Furthermore, I would like to express my sincere gratitude to Christopher Noble for his generous provision of the dataset from the "FASTWELL" and "INSIGHT" research projects financially supported by the Research Council of Norway. The invaluable contribution of this dataset has played a significant role in the successful execution of this thesis.

Finally, I would like to thank my family and friends for their unwavering support and encouragement. These years at NMBU have been an unforgettable experience, and I am grateful for everyone who has contributed to making it so.

Mikal Breiteig
Ås, May 15, 2023

Abstract

This master's thesis focuses on the evaluation and exploration of detection and tracking algorithms for fish in a dense underwater environment. The primary objectives were to achieve precise and accurate fish detection and to track fish over an extended period. The thesis explores the performance of two object detection algorithms, YOLOv4 and YOLOv8, as well as their integration with the DeepSORT tracking algorithm. The algorithms were trained and evaluated using a dataset collected from a densely populated underwater fish tank. The dataset was manually annotated using bounding box annotation techniques to accurately label the objects of interest.

The results demonstrated the effectiveness of both YOLOv4 and YOLOv8 in detecting fish in densely populated environments. However, YOLOv8 achieved a significantly higher mAP50-95 score, indicating better localization and detection accuracy. It proved more adept at precisely locating the position of detected fish, leading to improved overall detection performance.

In terms of fish tracking the combination of DeepSORT and YOLOv8 showed the best overall performance, as evidenced by higher MOTA and IDF1 scores, and lower MOTP scores. However, tracking individual fish over extended periods presented challenges due to occlusions and rapid trajectory changes, leading to a high number of identity switches.

By evaluating and exploring the effectiveness of detection and tracking algorithms, this thesis contributes to the advancement of fish monitoring techniques in aquaculture. The findings provide valuable insights into the performance of YOLOv4 and YOLOv8 and the potential of DeepSORT for accurate and reliable fish detection and tracking. The results and methodologies presented in this study lay the groundwork for further research and development in the field, aiming to enhance fish welfare, optimize resource management, and improve efficiency in aquaculture practices.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Ethical and humane treatment of fish has become increasingly important to customers in recent years, as noted by Conte [1] and Aas-Hansen et al. [2]. Customers are gradually looking for seafood produced in a sustainable and responsible way, which has increased focus on making sure that fish farming methods adhere to these values. Consumers are calling for more industry transparency as they become more conscious of the ethical and environmental effects of seafood production [1] [3]. By placing a higher priority on fish welfare and ethical practices the aquaculture industry can meet the needs of consumers while also advancing sustainability and ethical fish production.

The pursuit of ethical fish production is not solely driven by customers; it is also aligned with the United Nations' sustainability goal of "Life Below Water" The objective of this goal is to preserve and utilize the oceans, seas, and marine resources in a responsible manner [4]. By prioritizing fish welfare in aquaculture, the industry can promote sustainable and responsible fish production that supports this global goal.

Farming fish in an ethical and humane manner is widely recognized as a lucrative business [2]. Thus, enhancing the welfare of fish in aquaculture also has economic implications. Implementing effective and humane farming practices can reduce stress and makes the fish less vulnerable to diseases, resulting in a positive impact on the fish's overall health and well-being [5]. Consequently, they require fewer medications and treatments, exhibit improved growth rates, and yield a better quality product [5]. Prioritizing animal health can therefore result in increased revenue, expanded market opportunities, and an improved reputation for the industry. As a result, improving fish welfare is also a sound economic investment, and it is a crucial step for achieving sustainable and responsible fish production that supports the UN's sustainability goal.

The success story of fish farming in Norway is recognized as an impressive achievement in the aquaculture industry. The Norwegian government enacted the "Dyrevelferdsloven" Act to govern animal welfare in the country, which includes all types of farmed fish [6]. The Norwegian Food Authority (Mattilsynet) is responsible for overseeing the proper and humane treatment of fish and other aquatic animals in accordance with the act. In their published report "Fiskehelserapporten 2021", the Norwegian Veterinary Institute (Veterinærinstituttet) stated that the "dødfisk" (fish mortality) metric, which measures fish mortality during farming, is the most commonly reported and utilized welfare indicator in the industry [6]. However, measuring and documenting the progress of fish welfare improvement is a complex task, and it is important to realize that the concept of welfare extends beyond simply preventing fish from dying. Hence,

there is a need to explore further indicators that can provide insight into fish welfare during the farming process, thus enabling a more comprehensive understanding of the overall health and well-being of the fish.

In 2018, Noble et al. [7] published a handbook to assist farmers in evaluating fish welfare. The handbook covers various indicators categorized into two groups: animal-based indicators and environment-based indicators. Animal-based indicators include appetite, behavior, growth, and diseases, while environment-based indicators include water quality and lighting, among others[7]. The mortality rate indicator referred to earlier falls under the category of animal-based indicators. While some indicators can be visually observed, others require human intervention. Today, many farmers use underwater cameras to manually observe the behavior of fish through video, but this method can be inefficient and prone to inaccuracies due to limited visibility, fish occlusion, and the time-consuming nature of the process. Additionally, extracting objective information from these observations can be challenging due to the subjective nature of human interpretation.

In recent years, artificial intelligence has made significant advancements in computer vision, a field of study enabling computers to learn from visual data. This progress opens up the possibility of leveraging this technology to increase fish welfare by monitoring individual fish and how they interact with the environment. Utilizing computer vision can facilitate the real-time collection and analysis of numerous welfare indicators, hence, providing valuable insight into fish welfare. As the industry moves towards greater sustainability and efficiency, enhancing the level of autonomy will be crucial. To observe and monitor farmed fish, deep-learning-based computer vision techniques offer significant potential, as evidenced by previous research in the field. As discussed by Yang et al. [8], various computer vision techniques can be used to aid in fish species classification [9], counting [10] [11], and behavioral analysis [12] among others.

1.2 Problem Statement

Nofima, a leading Norwegian research institute for food and aquaculture, is exploring the potential of using computer vision to assess fish welfare without altering the environment. This technology has the potential to provide non-invasive, real-time monitoring of fish behavior and welfare, allowing farmers to make informed decisions on fish health. It is important to note that not all operational welfare indicators can be feasibly assessed using video alone. However, behavioral indicators can be effectively measured using computer vision technology, as fish behavior is closely tied to their overall welfare. For example, swimming patterns, feeding behaviors, and social interactions can all provide valuable insight into fish welfare. By analyzing these behavioral indicators, farmers can detect any abnormal responses to the environment and potential health issues.

By minimizing disturbance to the fish, this technology could improve the accuracy and reliability of welfare assessment. Such insights into fish welfare can be leveraged to develop more effective strategies for achieving optimal living conditions, leading to more efficient and sustainable fish farming practices. While still in its early stages of use in aquaculture, the potential of computer vision technology to enhance fish welfare cannot be overlooked.

Building upon the potential of computer vision technology in aquaculture, the primary aim of this thesis is to identify and track fish in a dense aquatic environment over an extended period. This project may be part of a larger effort to utilize artificial intelligence for real-time event detection in fish tanks, such as attacks, flights, flashes, or bites. However, the specific goals of this project thesis will be focusing on achieving the following objectives sequentially:

1. Detect individual fish in a dense aquatic environment with high precision and accuracy utilizing computer vision.
2. Track individual fish over an extended period based on computer vision in the same dense environment.

From a computer vision perspective, the primary difficulty in detecting and tracking fish in a dense environment lies in the complex and dynamic nature of fish behavior. Fish can exhibit unpredictable movement patterns, sudden changes in direction, and visual obstruction of other fish, which can make it challenging for object detection algorithms to accurately identify and track. Furthermore, the appearance of fish can vary based on factors such as lighting, water quality, and camera placement, which can affect the performance of the algorithms.

In this preliminary attempt to employ computer vision technology for detecting and tracking fish, Nofima does not have a sufficient foundation to select a specific threshold. Therefore, the primary aim of this thesis is to attain precise object detection accuracy and to achieve long-duration tracking, while acknowledging that the requisite level of performance may vary based on the environment. To detect fish, the object detection algorithms YOLOv4 [13] and YOLOv8 [14] have been selected. To track fish over an extended period, these algorithms will be integrated with the DeepSORT [15] algorithm, recognized for its high accuracy in multi-object tracking (MOT). The outcomes of the algorithms will be presented and contrasted in Chapter 4.

The datasets used in this thesis were generously provided by Christopher Noble at Nofima, and consist of between 30 and 40 salmon. These datasets were collected in 2021 and the video frames from the datasets were analyzed using the aforementioned computer vision algorithms, which will be further detailed in subsequent chapters. The dataset is concurrently being used in another master's thesis to explore computer vision applications using Detectron2 and YOLOV8 as detectors and ByteTrack [16] as the tracking algorithm. The choice to use this particular dataset allows for a direct comparison of our findings, enabling a deeper understanding of the challenges and potential solutions in computer vision used in aquatic settings.

Furthermore, the main contribution of this master's thesis lies in the exploration and evaluation of the performance of algorithms in the context of detecting and tracking fish in a dense environment. Due to a lack of open knowledge in this application, this research endeavors to bridge the gap by thoroughly examining the feasibility of employing computer vision for such a task. This thesis serves as a foundation for further research on the unique challenge presented by detecting and tracking fish and contributes to the goal of improving fish welfare.

1.3 Structure of the thesis

The structure of the following chapters is as follows. Chapter 2 provides an overview of the theoretical background relevant to this thesis. Chapter 3 presents the methodology used during object detection and object tracking. Chapter 4 presents and compares the results. Chapter 5 discusses these results and suggests future work. The results are summarized and concluded in Chapter 6.

Chapter 2

Theory

Machine learning, a branch of artificial intelligence, aims to create algorithms that enable computers to learn without being explicitly programmed. Such self-learning algorithms that utilize data to derive knowledge are employed to enhance performance during prediction [17]. Machine learning can be categorized into three main categories: supervised learning, unsupervised learning, and reinforcement learning [17]. Supervised learning is the focus of this thesis and aims to train a model that can make accurate predictions on new and unseen data [17]. To achieve this, labeled training data is utilized through a data annotation process. Accurate data annotation is essential to secure a generalized model on unseen data. The training process of a supervised learning algorithm uses feedback to iteratively adjust the model's parameters to minimize the difference between the predicted output and the ground truth of the training data.

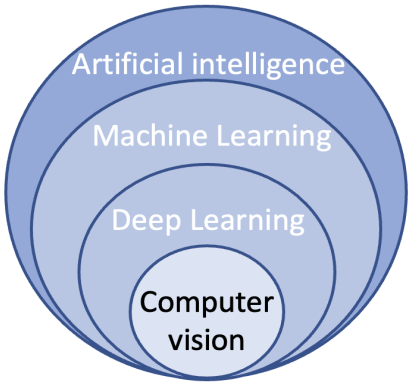


Figure 2.1: Overview of Artificial intelligence hierarchy.

Deep Learning, a branch of machine learning, utilizes artificial neural networks (ANNs) to analyze data and make predictions [17]. The fundamental idea behind ANNs is based on how the human brain functions to solve complex problems [17]. Due to the recent advancements in computing infrastructure and the availability of large amounts of training data, deep learning has become increasingly popular. These advancements have led to significant breakthroughs in various domains, including natural language processing, speech recognition, and computer vision.

In light of these advancements, this chapter aims to establish a theoretical framework that serves as a foundation for this thesis. To achieve this, the chapter delves into the fundamental

concepts related to deep learning. Furthermore, it provides an overview of the inner workings of the YOLOv4 and YOLOv8 object detection algorithms, as well as the DeepSORT tracking algorithm. Finally, the chapter outlines the evaluation metrics that will be utilized to assess the performance of the models in the context of fish detection and tracking.

2.1 Deep Learning Theory

2.1.1 The Perceptron

In 1958, Frank Rosenblatt introduced the concept of the Perceptron, an advancement of the earlier McCulloch-Pitts (MCP) neuron model [18]. The MCP neuron was an initial endeavor to comprehend the biological brain's functionality with the objective of developing artificial intelligence [19]. Building upon the foundation laid by the MCP neuron, the Perceptron is designed to learn from a set of examples and adapt its weights in order to minimize the discrepancy between its predicted output and the actual ground truth output.

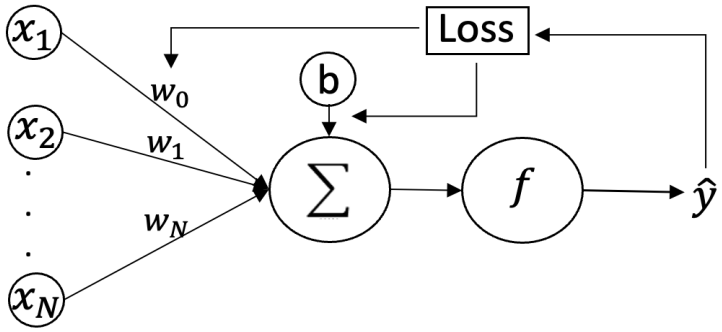


Figure 2.2: The architecture of Rosenlatts Perceptron. Input signals, along with their respective weights, contribute to a weighted sum that is subsequently directed to an activation function (f). Figure inspired by Raschka [17].

As stated in the article by Rosenblatt, the fundamental requirement for the Perceptron is its capacity to recognize patterns that display considerable similarities or are connected through experimental evidence. This mechanism resembles the psychological concepts of "association" and "stimulus generalization" [18]. The Perceptron must be capable of recognizing the "same" object despite changes in orientation, size, color, or transformation, as well as against diverse backgrounds [18]. Today, such neurons hold a vital position in the field of deep learning, as they serve as the foundational building blocks for constructing ANNs that can effectively process and learn from vast amounts of data. ANNs have become a key technology in a wide range of fields, and ongoing research continues to push the boundaries of what is possible with this powerful technology.

2.1.2 Multi-Layer Neural Network

In ANNs, a neuron is referred to as a unit or a node. A single layer in the network consists of several of these units receiving the same input, but are differentiated by their weights. Individually, each unit has limited efficacy, producing only a single numerical value [20]. However, the true power of the system emerges when multiple units are combined into a single layer,

as exemplified by the Multi-Layer Perceptron (MLP) illustrated in Figure 2.3. The MLP is a type of fully-connected ANN that utilizes the concept of aggregating units into layers. It represents the basic architecture of a neural network with an input layer, at least one hidden layer, and an output layer. If the MLP contains multiple hidden layers it is referred to as a deep ANN.

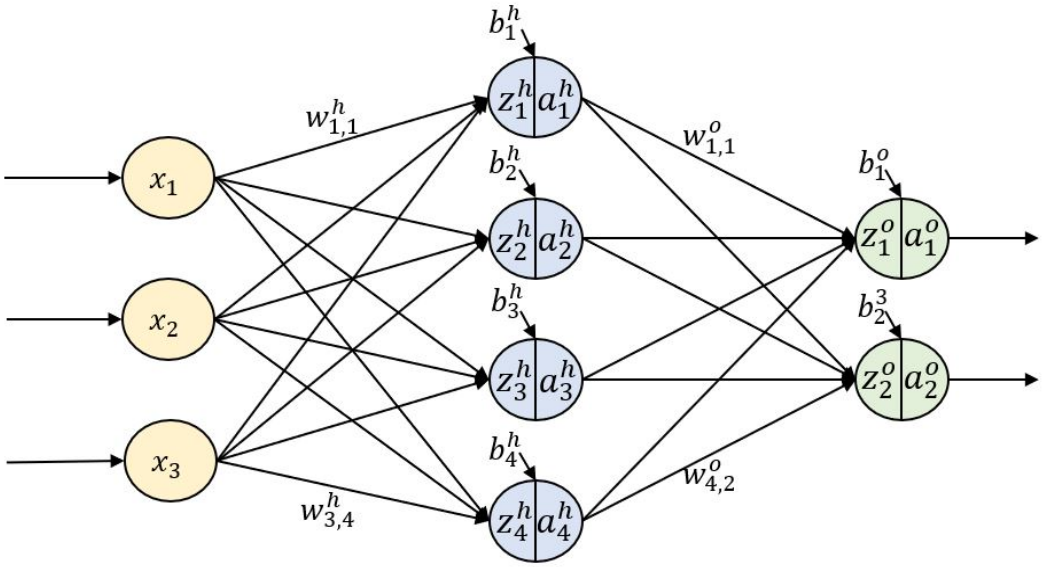


Figure 2.3: The figure illustrates the structure of a Multi-Layer Perceptron (MLP) neural network. The particular MLP consists of an input layer, a hidden layer, and an output layer. Each layer contains a set of units that are connected to units in the previous and following layers. The units in the hidden and output layers calculate the weighted sum that is passed to an activation function, transforming the input data. The weights between the units in each layer are adjusted during the training process to optimize the network's performance on a specific task.

In a neural network, the input signals get processed through the network to form a prediction. The weighted sum for each unit is computed, with the following equation illustrating the calculation for the first unit in the hidden layer, denoted as z_1^h:

$$z_1^h = x_1 \cdot w_{1,1}^h + x_2 \cdot w_{2,1}^h + x_3 \cdot w_{3,1}^h + b_1^h \tag{2.1}$$

The term z_1^h denotes the weighted sum associated with neuron, while b_1^h represents the corresponding bias. Inside the neuron, a non-linear activation function typically controls the calculated weighted sum. The inclusion of the bias term in the activation function allows for horizontal shifting, providing the model with greater flexibility to capture diverse patterns and relationships in the data [21]. The activation function takes the weighted sum as its input and computes its value accordingly:

$$a_1^h = \phi(z_1^h) \tag{2.2}$$

Various activation functions can be employed depending on the specific scenario and the desired properties of the network. Figure 2.4 illustrates the graphical representation of Sigmoid and ReLu activation functions, along with their corresponding derivatives [22].

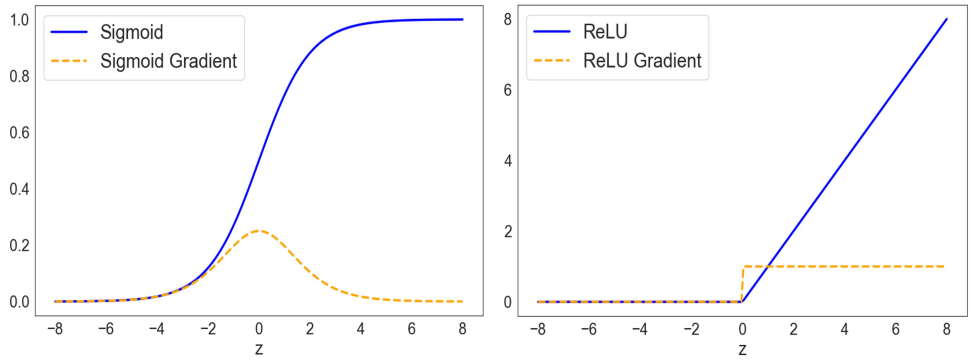


Figure 2.4: Sigmoid and ReLU activation functions and their derivatives. The derivatives are useful during backpropagation.

2.1.3 Optimization: Backpropagation and Gradient Descent

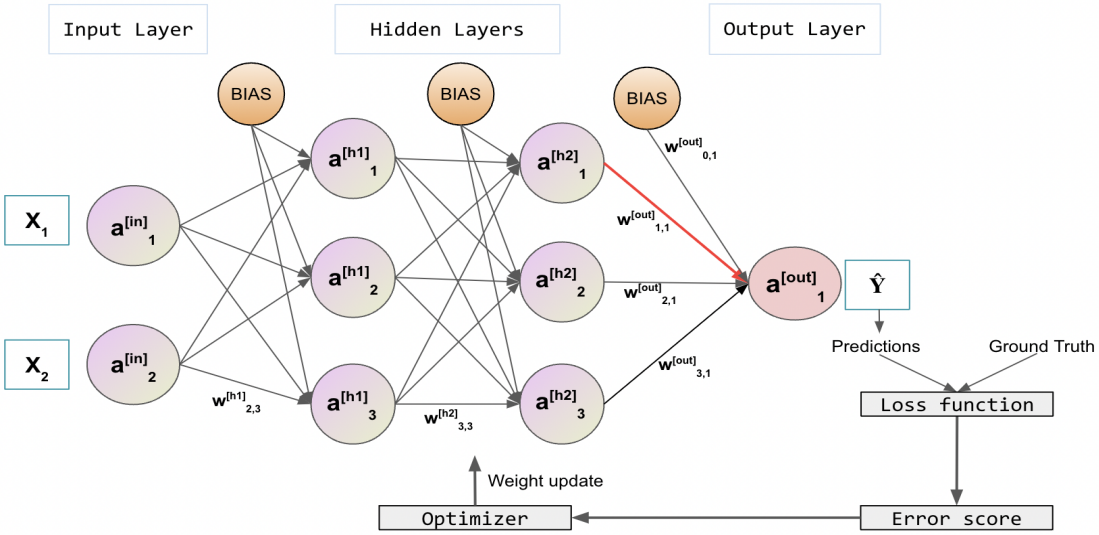


Figure 2.5: Illustration of the backpropagation process, where predictions from the output layer are compared to ground truth labels using a loss function, resulting in error scores that guide the subsequent update of model weights to minimize prediction errors.

An ANN process input signals to generate a prediction, which is then evaluated against the ground truth using a loss function. An optimization strategy is applied to fine-tune the network's parameters, ultimately improving prediction accuracy. The error or loss associated with the output layer can be expressed as:

$$\delta^{[out]} = a^{[out]} - y = \hat{Y} - y \tag{2.3}$$

where $a^{(out)}$ is the activation signal of the output layer and y is the ground truth. Backpropagation is an optimization technique that efficiently adjusts a network's parameters (weights and biases) to minimize loss by leveraging the error vector, $\delta^{[out]}$. It utilizes the widely used gradient descent optimization algorithm to refine the parameters and decrease the loss. Backpropagation is a technique for computing the negative gradient of the loss function, which measures the difference between the predicted and ground truth values [23]. The chain rule displayed in Equation 2.4, a fundamental principle in calculus, is used to calculate the partial derivative of the loss function with respect to each parameter. In backpropagation, this is achieved by

multiplying a series of partial derivatives to propagate errors back through the network to the input layer. This allows for efficient parameter updates and improves the performance of the ANN.

$$\frac{d}{dx}[f(g(x))] = \frac{df}{dg} \cdot \frac{dg}{dx} \tag{2.4}$$

During each iteration of the data, called an epoch, the $\delta^{[out]}$ is used to propagate the error back through the network and compute the gradients concerning each parameter. As seen in Equation 2.5, the magnitude of each parameter's adjustment is scaled by a learning rate (η), which controls the step size of the update at each iteration. The "J" in the equation represents the loss function.

$$w := w + \Delta w, \text{where } \Delta w = -\eta J(w) \tag{2.5}$$

Referring to Figure 2.5, the partial derivative of the weight associated with the $a_1^{[h2]}$ node (red line) can be calculated as follows:

$$\frac{\partial}{\partial w_{1,1}^{[out]}} J(W) = a_1^{[h2]} \delta_1^{[out]} \tag{2.6}$$

where $\delta_1^{[out]}$ is the error term for the output node. The parameters are then iteratively adjusted from the output layer toward the input layer, passing through each hidden layer, to improve the prediction. This process continues until the model converges to an optimal solution.

The principles of backpropagation are employed in numerous forms of ANNs. In the following section, we will introduce a particular type of neural network inspired by the human brain, known as Convolutional Neural Networks (CNNs). These networks excel at processing images and videos, making them ideal for computer vision tasks.

2.1.4 Convolutional Neural Network (CNN)

As humans, our perceptual faculties enable us to understand the three-dimensional world in which we exist. This capability stems from the seamless interaction between our eyes and brain, which work collaboratively to understand visual information. In the field of computer vision, this phenomenon is replicated using deep learning and ANNs, which aim to mimic the cooperative behavior of the human visual system. With the advancement of computing technology, computer vision is now capable of handling and analyzing extensive visual data. This technology finds its use in several areas, including autonomous driving [24], security systems [25], and medical imaging [26].

One of the most influential developments in deep learning for computer vision has been the introduction of Convolutional Neural Networks (CNNs). CNNs are a family of models that stems from the way the human visual cortex recognizes objects [17]. It is a type of ANN and is particularly proficient at performing image analysis tasks with high accuracy. The benefits of CNNs include their ability to recognize patterns regardless of their location in the image, due to their translation invariance, and to efficiently learn complex and abstract visual concepts through the use of spatial hierarchies of patterns [21]. Unlike MLP, which relies on fully connected layers where each unit in a layer connects to all the units in the adjacent layers, CNNs

are constructed using a combination of convolutional, pooling, and fully connected layers, as seen in Figure 2.6.

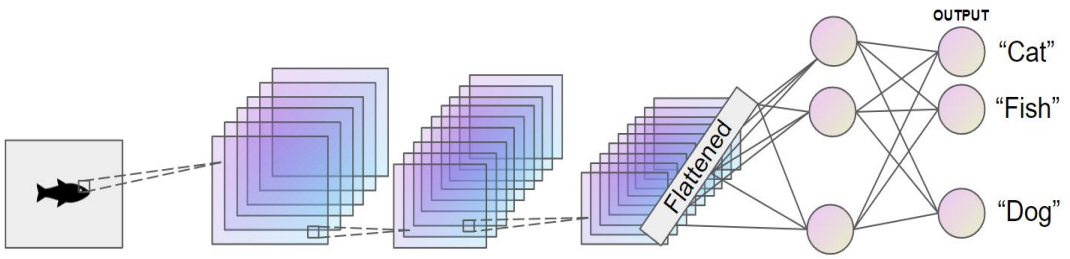


Figure 2.6: The figure illustrates a CNN used on an image. It consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers. Figure inspired by Raschka [17].

Convolutional Layer

A convolutional layer is a fundamental component in CNNs used for processing input signals, such as images or videos. In these networks, the initial layers focus on extracting low-level features, such as edges and blobs, which are subsequently integrated into higher-level features through the successive convolutional layers [17]. In Convolution 2D, a set of filters with dimensions of 3x3 or greater is applied to the input feature map. Activation occurs by moving the adjustable-weight filter across the input, calculating the dot product between the filter and each input's local section, and subsequently applying a non-linear activation function.

In Figure 2.7, the process of performing a 2D convolution between an input matrix X_{3x3} and a filter matrix W_{3x3} is illustrated. The convolution is carried out with a padding of p=(1,1) and a stride s=(2,2). Padding involves appending rows and/or columns of pixels, usually with zero values, to the input feature map to maintain the image's spatial resolution during convolutional operations. The stride refers to the step size by which the filter moves across the input feature map during convolutional operations. A larger stride results in a smaller output feature map and vice versa. Conversely, a stride of 2 causes the filter to skip one position in each step, reducing the output's spatial dimensions and increasing computational efficiency at the cost of potentially losing some local information.

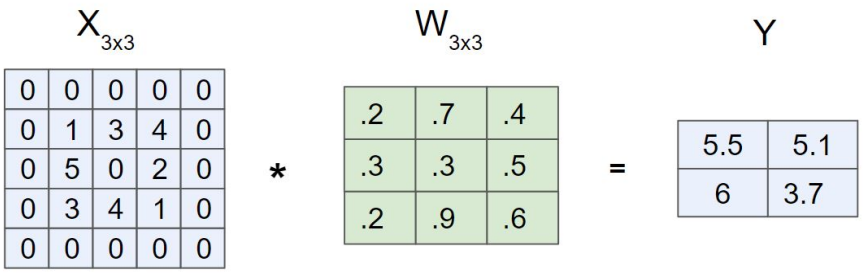


Figure 2.7: A convolutional operation with a 3x3 input feature map with padding. The filter matrix has dimension 3x3 and uses a stride of 2 to return a feature map with dimension 2x2.

The output value for the upper-left cell of the output matrix is calculated by taking the dot product of the corresponding elements of the input matrix and the filter matrix and summing the results. In this case, the calculation is done as follows:

$$5.5 = (0 \cdot 0.2 + 0 \cdot 0.3 + 0 \cdot 0.2) + (0 \cdot 0.7 + 1 \cdot 0.3 + 3 \cdot 0.9) + (0 \cdot 0.4 + 5 \cdot 0.5 + 0 \cdot 0.6)$$

Pooling Layer (Sub-sampling)

The pooling layer in CNNs performs downsampling of the input feature map, similar to the convolutional layer with stride, with the aim of decreasing its spatial dimensionality, increasing computational efficiency, and expanding the receptive field. Max pooling is the most common pooling method in CNNs, where a filter without weights is used to extract the maximum value over the filter area, as seen in Figure 2.8. This method helps retain the most salient feature within each local region of the feature map, in addition to reducing the spatial dimension [17].

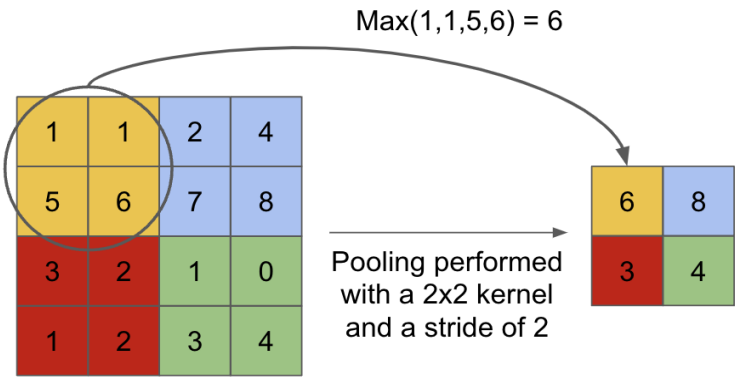


Figure 2.8: Max pooling with a 2x2 filter and stride of 2. Figure inspired by Saravia [27].

Batch Normalization

Batch Normalization facilitates faster and more stable training during the learning process of deep ANNs [17]. In essence, Batch Normalization is an approach designed to enhance the training of ANNs by stabilizing the distributions of inputs across layers [28]. The technique is implemented through the use of four parameters, namely gamma (γ), beta (β), moving mean (μ), and moving variance (σ^2). Gamma scales the normalized value, while beta shifts the normalized value. The moving mean is the mean of the current batch, a subset of the training data, used to normalize the batch while the moving variance is the variance of the current batch used for normalization. The main advantage of batch normalization is the improvement of gradient propagation, which is similar to residual connections explained below, allowing for deeper networks to be created.

Non-sequential Layer

Non-sequential layers have been shown to be effective in improving the performance of deep ANNs, particularly in image recognition tasks [29]. In a sequential network, the output of one layer acts as input to the next layer in a linear fashion. However, in a non-sequential neural network, the connections between various layers can be complex, allowing for more flexible and powerful architecture.

Residual connection, seen in Figure 2.9, is a type of non-sequential layer that is commonly used in CNNs. It is a shortcut connection that bypasses one or more convolutional layers, allowing the network to keep information from earlier layers and avoid the problem of the vanishing gradient [30]. A residual block is another type of non-sequential layer that contains one or more

residual connections, providing a shortcut path for the gradient to flow back to earlier layers during backpropagation.

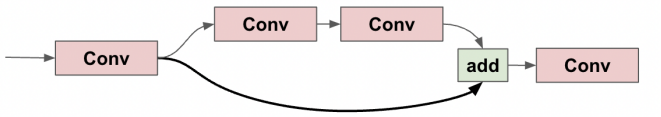


Figure 2.9: The figure shows an example of a non-sequential connection.

2.1.5 Transfer Learning

While CNNs have proven to be highly effective in extracting features and learning complex patterns from images and videos, training these networks from scratch is computationally expensive, time-consuming, and requires a large amount of available data [28]. This challenge has given rise to the concept of transfer learning, a technique that enables the reduction of training time by using the knowledge gained from pre-trained models to accelerate the learning process. This approach is especially beneficial for deep ANNs, which can require significant time and computational resources to train. A pre-trained model has been trained on a large, diverse, and publicly accessible dataset, enabling it to recognize low-level features. Subsequently, the network can be fine-tuned on a custom dataset to learn more complex features, thereby enhancing its overall performance. The use of transfer learning can significantly reduce the amount of training time required for deep networks used in object detection tasks, while simultaneously enhancing the model's performance [31].

2.2 Object Detection and YOLOv4 and YOLOv8 Detectors

In this section, we will explore object detection as a method within the domain of computer vision, and present two prominent object detectors: YOLOv4 and YOLOv8. A thorough examination of their architectural design and components will be provided to enhance comprehension of their efficacy in detecting fish within densely populated fish environments.

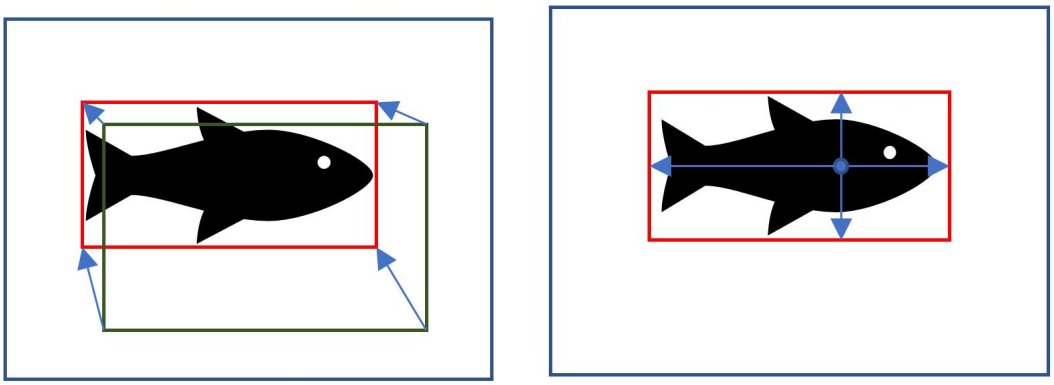
2.2.1 Object Detection: An Overview

Object detection is a technique that aims to recognize and locate different objects within an image or video and assign them to specific classes or categories, such as people, buildings, or vehicles [23]. Object detection has gained significant popularity in recent years due to the breakthroughs achieved by ANNs. Specifically, methods leveraging CNNs have gained substantial traction and have demonstrated their effectiveness in object detection [32] [33] [34].

Region-based and single-shot methods represent the two primary approaches to object detection. Region-based methods, such as the R-CNN family [32], involves a two-stage process, where the first stage generates object proposals, and the second stage classifies these proposals using CNNs. Although these methods have demonstrated high accuracy in object detection tasks [33], they often suffer from computational inefficiency and slower processing times [35]. Unlike regional-based, single-shot methods such as YOLO aim to predict both class probabilities and bounding box coordinates in one go as data passes through the CNN. This makes them faster and better adapted for applications that require real-time processing [35]. As a result, the

single-shot method has been chosen as the focus of this thesis.

Single-shot detection algorithms can be divided into two subclasses: anchor-based and anchor-free. Both subclasses aim to predict class probabilities and bounding box coordinates in a single pass through the CNN. The difference between the two methods lies in how they handle the selection and placement of bounding boxes.



(a) Anchor-based detection predicting the offset based on a predefined box.

(b) Anchor-free detection where the method estimates the offset of a point to its boundaries.

Figure 2.10: The provided illustration depicts the contrast between two approaches for object detection, namely anchor-based and anchor-free. The true boundary of the object is represented with a red rectangle, while the green rectangle represents the pre-defined anchor. Moreover, the blue lines indicate the deviations. Figure inspired by [36].

Anchor-based object detection, as depicted in Figure 2.10a, is a widely used approach that improves upon the traditional sliding window technique [37]. In anchor-based detection, the network predicts a fixed set of bounding box anchors with predefined aspect ratios and scales, called anchor boxes [36]. These anchors act as reference boxes, which the network then adjusts to fit the object's shape and location. One disadvantage of anchor-based detection is that it requires a large number of anchors to be generated, which can be computationally expensive and time-consuming.

In contrast, anchor-free detection, as depicted in Figure 2.10b, eliminates the need for predefined anchors by directly regressing to the object's location and size [36]. This approach is more computationally efficient than anchor-based detection because it does not require the generation and selection of anchors. Anchor-free detection has gained significant popularity in recent years, with methods like CenterNet [38] achieving state-of-the-art performance in object detection tasks. The recently introduced YOLOv8 model also incorporates an anchor-free approach to object detection.

Despite the significant advancement in object detection techniques in recent years, challenges identified by Shakik et al. [39] in 2014 still exist today. Some of the primary challenges include handling scale variation, where objects may appear in different sizes; aspect ratio variation, where objects may have diverse shapes and orientations; occlusion, where parts of the objects may be hidden or overlapped by other objects; and background clutter, where objects may blend in with or be confused by other elements in the scene.

In the following sections, we will explore the object detectors YOLOv4 and YOLOv8, and delve

into their architecture and components to understand their performance in detecting fish in a high-density fish environment.

2.2.2 YOLOv4: Architecture and Features

YOLOv4 [13], released in April 2020, is the fourth iteration of the popular You Only Look Once (YOLO) series of object detection models. YOLOv4 introduces notable enhancements compared to its earlier versions, including increased accuracy, speed, and overall efficiency, positioning it as a strong option for tasks requiring real-time object detection. The architecture of the anchor-based YOLOv4 algorithm consists of three parts: Backbone, Neck, and Head.

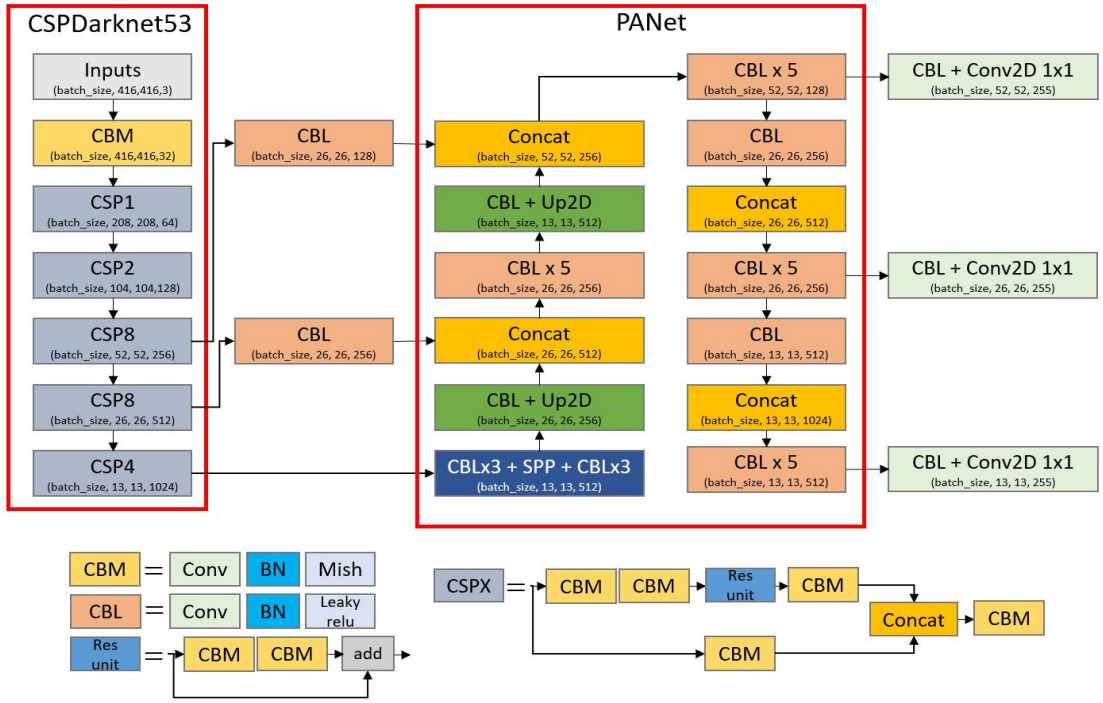


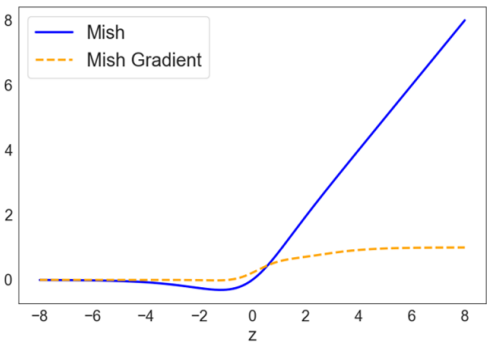
Figure 2.11: The figure illustrates YOLOv4 architecture with CSPDarknet53 as Backbone, PANet as the Neck, and three detection heads.

The Backbone

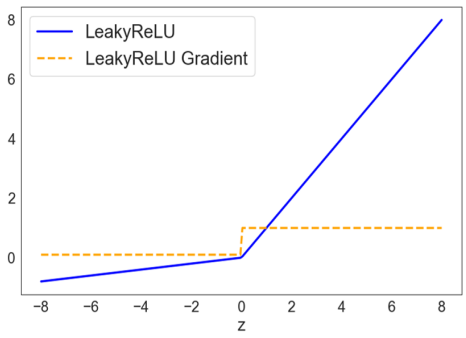
The Backbone is used for feature extraction and YOLOv4 utilizes CSPDarknet53 [40] which is an adaptation of the Darknet53 [41] backbone employed in YOLOv3. Darknet53 is based on DenseNet design [42], a design to alleviate the vanishing gradient problem. CSPDarknet53 integrates the advanced learning capabilities of the Cross-Stage Partial Network (CSPNet) [40] into the Darknet53 architecture. The CSP connections, a type of non-sequential connection (Section 2.1.4), are fused into the Backbone of the architecture to improve information and gradient flow during both forward and backward passes. This integration enables the network to learn and utilize more complex features and correlations between them.

CSRDarknet53 comprises five residual blocks, called CSP blocks, with a total of 53 convolutional layers, each accompanied by a batch normalization layer (Section 2.1.4). The model is pre-trained (Section 2.1.5) on the ImageNet dataset [43]. The backbone leverages the Mish activation function [44], which resembles ReLU but offers a smoother transition near zero and

a non-zero derivative at zero, as seen in Figure 2.12a.



(a) Mish activation and its gradient.



(b) Leaky ReLu activation and its gradient.

Figure 2.12: The figures display a comparison between the frame before and after annotation. The annotated bounding boxes are represented by solid lines, while the dotted lines indicate occluded objects.

The Neck

The Neck of YOLOv4 is comprised of Spatial Pyramid Pooling (SPP) [45] and Path Aggregation Network (PANet) [46]. In the SPP process, the feature layer undergoes three convolutions initially following a max pooling (Section 2.1.4) using various filter sizes (5x5, 9x9, 13x13). The pooling results are then concatenated and further convolved three times. The objective of SPP is to enable the network to manage input images of varying sizes and extract features that are invariant to scale and aspect ratio. After the SPP block, the PANet processes the feature layers by convolving and up-sampling them, effectively doubling their height and width. It then concatenates the convoluted and up-sampled feature layers with those obtained from CSPDarknet53 to achieve feature fusion. Following this, the network performs downsampling, which compresses the height and width of the layers. Lastly, it stacks these layers with previous ones to accomplish further feature fusion. PANet enables the network to utilize information from both lower-level and higher-level features, enhancing its accuracy in object detection tasks. Whilst the Backbone of YOLOV4 uses Mish as an activation function, the Neck uses the Leaky ReLU function seen in Figure 2.12b [47].

The Head

The Head of YOLOv4 can make detections by using the features extracted earlier in the network. As seen in Figure 2.11, YOLOv4 uses three detection heads operating at different scales - large, medium, and small - to capture objects of various sizes. The detection heads in the model have the task of making predictions about various attributes of the objects, including their class, bounding box coordinates, and objectness score. To aid in this process, each detection head relies on a set of anchor boxes. These anchor boxes supply crucial insights that enable the model to better understand the form and dimensions of the objects it aims to detect.

The input image is first divided into a grid of cells, with each cell corresponding to a specific region of the image. Within each cell, there are nine pre-defined anchor boxes that have specific aspect ratios and scales. These anchor boxes are used to predict the object's class, location, and size within that cell. The prediction process involves calculating the confidence score for each anchor box, which represents the likelihood of an object being present in that location [13]. The highest-scoring anchor box is selected as the one responsible for detecting the object within

that cell. This approach allows the YOLOv4 model to efficiently detect objects of various sizes and shapes.

During the inference process in YOLOv4, an input image is resized to a fixed size (e.g., 416x416 pixels) while maintaining its aspect ratio. The image gets passed through the Backbone where it extracts hierarchical feature maps. These feature maps are passed to the Neck, generating three feature maps of different scales to facilitate the detection of objects of varying sizes. Each of these feature maps is fed into a corresponding detection head, which are convolutional layers that predict class probabilities, bounding box coordinates, and objectness scores for each grid cell in the feature map. To obtain the final bounding box coordinates, the predicted offsets and scales are decoded with respect to predefined anchor boxes.

Predictions with low confidence are eliminated using a confidence threshold based on Intersection over Union (IOU), measuring the overlap between the predicted bounding box and the round truth bounding box (Section 2.4). The model retains only those bounding boxes with an IOU exceeding the specified threshold. Subsequently, the remaining bounding boxes undergo Non-Maximum Suppression (NMS) [48], an algorithm that effectively removes overlapping boxes. It involves sorting the bounding boxes based on their confidence scores and iteratively removing boxes with high overlap until only the most confident prediction is retained [23]. The NMS process comprises the following steps:

1. Eliminate all bounding boxes that have predictions with probability less than a particular threshold. This threshold, called the confidence score, can be customized, and a bounding box will only be preserved if the prediction probability is above this threshold.
2. Analyze the remaining boxes, and pick the box with the highest probability.
3. Calculate the IOU for the remaining boxes that predict the same class, which measures the degree of overlap between them. If two boxes have high overlap and predict the same class, they are averaged together. Averaging these boxes means that their coordinates (i.e., x, y, width, and height) are combined into a single bounding box that represents the object more accurately.
4. Eliminate any box with an IOU value lower than a specific threshold (NMS threshold). Typically, the NMS threshold is 0.5, but it can be adjusted to produce fewer or more bounding boxes.

The final output of the YOLOv4 algorithm consists of bounding boxes, class labels, and confidence scores, which together represent the detected objects, their locations, and their respective classes within the input image. YOLOv4 leverages multi-scale feature representation using a combination of its backbone CSPDarknet53, PANet, and SPP module, enabling the detection of objects with varying sizes and shapes. The application of thresholding and NMS ensures that the resulting detections have high confidence and minimal overlapping boxes.

2.2.3 YOLOv8: Architecture and Features

The YOLOv8 model is a recent addition to the family of YOLO models and was introduced in 2023 by Ultralytics, a computer vision company responsible for YOLOv5. At present time, YOLOv8 is yet to be fully documented or described in the literature, hence, it remains relatively unexplored in academic circles, with no official publications or detailed descriptions of

the model's inner workings currently available by Ultralytics. However, the team has expressed an interest in releasing such a paper in the near future, but at the current time, they focus on incorporating new features into the YOLOv8 framework. Notwithstanding the lack of an official publication, the development team has received a significant level of interest from the community. Indeed, the community has been actively engaging with the Ultralytics team to better understand the model, underscoring the potential utility of YOLOv8. A visual representation of the YOLOv8 architecture, confirmed by one of the YOLOv8 developers on GitHub issues [49], is presented in Figure 2.13.

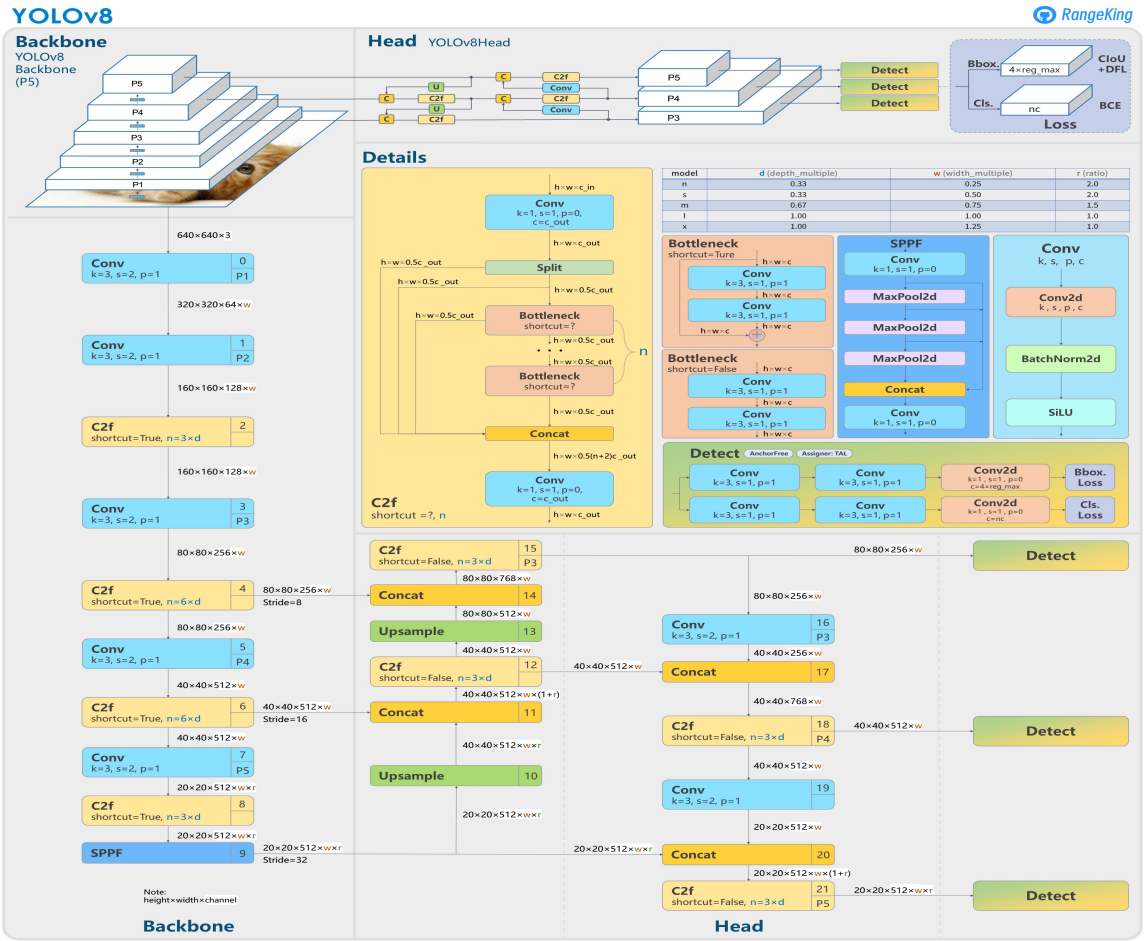


Figure 2.13: The architecture of YOLOv8 object detector created by the GitHub user RangeKing.

Given the absence of an official publication, a comprehensive investigation has been conducted into the YOLOv8 model's repository and the available information to document its novel features. This analysis aims to provide a thorough understanding of the model's unique characteristics, despite the lack of an official publication or detailed description of its inner workings. Consistent with the architecture of YOLOv4, YOLOv8 comprises three primary sections: the Backbone, Neck, and Head.

The Backbone

The Backbone of YOLOv8, similar to YOLOv4, is responsible for extracting relevant features from the input data. YOLOv8 uses a modified version of the CSPDarknet53 network presented in Section 2.2.2 used in both YOLOv4 and YOLOv5. In YOLOv8, all convolutional operations in the network utilize the Sigmoid Linear Unit (SiLu) activation function as seen in Figure 2.14,

in contrast to YOLOv4 using Mish and LeakyReLU activation functions. In the backbone of YOLOv8, the four CSP layer (also known as "C2f") is associated with a skip connection, a type of non-sequential connection, in the figure. The use of non-sequential connections is a fundamental design choice in the CSPDarknet53 backbone. It helps preserve important features and gradients as they pass through the network, ultimately contributing to improved performance in object detection.

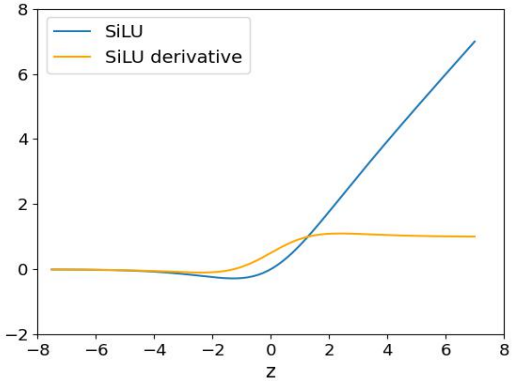


Figure 2.14: The figure illustrates the SiLu activation function and its derivative.

The Neck

The Neck section of YOLOv8 is responsible for merging the features extracted by the Backbone and increasing their resolution. The YOLOv8 uses an SPPF (SPP-Fast) module, which produces results comparable to the SPP introduced in Section 2.2.2. However, it requires fewer floating-point operations (FLOPs), which is why it is labeled "Fast" or "F". This reduction in FLOPs allows the algorithm to consume fewer computational resources.

Furthermore, YOLOv8 uses PANet which includes several upsampling layers to increase the spatial resolution of the feature maps and concatenation operations to combine feature maps generated by the Backbone and the Neck. It also includes additional convolutional layers to refine the features and extract object-specific information. In YOLOv8, the PANet is modified by incorporating the CSPNet strategy, as depicted in the figure. By introducing CSP into the neck the YOLOv8 can obtain more abundant gradient flow information while ensuring a lighter architecture [40].

The Head

Finally, the Head section of YOLOv8 is dedicated to predicting bounding boxes and object classes within an image. The YOLOv8 head employs an array of convolutional layers and decoupling structures to efficiently and accurately recognize objects in the image. It carries out the detection task by processing feature maps generated by the Backbone and the Neck. The Head is characterized by a decoupled design, in which classification and regression tasks are processed independently. This arrangement enables each branch to concentrate on its specific task, thereby enhancing the model's overall performance [50]. The head processes feature maps using a sequence of convolutional layers, followed by a linear layer for predicting bounding boxes and class probabilities.

It is important to note that YOLOv8 is an anchor-free model that employs a novel approach to object detection. Contrasting with YOLOv4, which predicts an object's center coordinates and offsets from anchor boxes, YOLOv8 directly predicts the object's center coordinates without the use of anchor boxes. This eliminates the need to calculate the offset with respect to the anchor box, thereby reducing the number of box predictions. As a result, the new method used in YOLOv8 significantly speeds up the complex inference step of NMS.

2.3 Object Tracking and DeepSORT

2.3.1 Object Tracking: An Overview

A tracking-by-detection tracking algorithm, illustrated in Figure 2.15, comprises a dual-process approach. Firstly, an object detection algorithm (e.g., YOLOv4) generates bounding boxes enclosing the objects of interest within individual frames. Secondly, the tracking algorithm conducts data association across consecutive frames to create trajectories for effectively tracking the identified objects. In recent years, deep learning-based computer vision techniques have demonstrated substantial enhancements in object-tracking performance [51]. These deep learning methods can capture more intricate representations of an object's appearance and motion, resulting in better tracking performance.

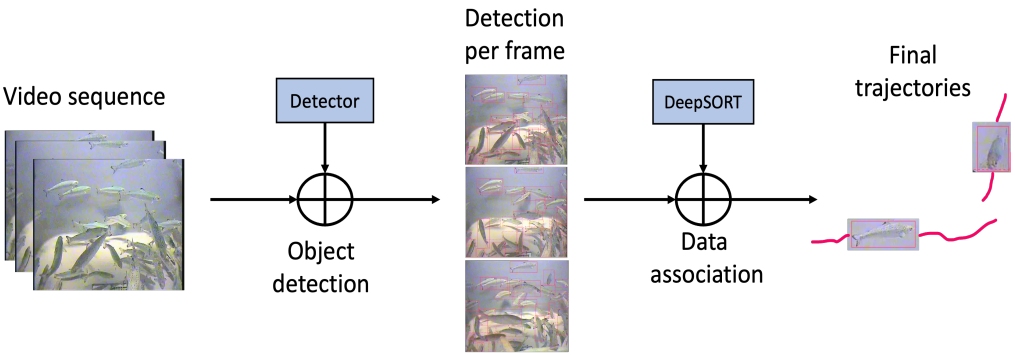


Figure 2.15: Visual representation of tracking-by-detection using DeepSORT as the object tracking algorithm. The detections obtained for each frame are used as input to the tracking part of the tracking algorithm, which performs data association and creates trajectories for the detected objects.

There are several challenges to object tracking, including scale variation, motion blur, occlusion, and object appearance changes. To overcome these challenges, object tracking algorithms often use a combination of motion models, appearance models, and data association methods. Motion models describe how objects move in the scene and can be used to predict the location of the object in the next frame. Appearance models describe the visual appearance of the object and can be used to identify the object in different frames, even if the object's appearance changes due to occlusion or illumination changes. Data association methods are used to link object detections across frames, enabling the creation of object trajectories.

2.3.2 Object Tracking with DeepSORT

DeepSORT (Simple Online Realtime Tracker with a Deep Association Metric) [15] serves as an extension of the SORT (Simple Online Realtime Tracking) [52] tracking algorithm. One of the limitations of the SORT algorithm was the high number of track switches during object tracking in dense scenarios. The DeepSORT algorithm tries to address this limitation by incorporating a Deep Association Metric that uses a learned feature representation of each object to associate the object across frames. The algorithm has been shown to be effective in handling complex tracking scenarios, such as occlusion, and can track multiple objects in real-time [15].

During the inference process, an object detection algorithm (such as YOLOv4 or YOLOv8) is employed to identify objects of interest within video frames, generating bounding boxes, class labels, and confidence scores for each detected object. Subsequently, DeepSORT computes a cost matrix, which is derived from appearance features (obtained via a deep ANN), spatial distance (obtained through the Kalman filter), or a combination thereof. The cost matrix's rows correspond to detected objects, while columns represent predicted tracks. The Hungarian algorithm is then utilized to ascertain the optimal assignment, minimizing the overall cost (distance) between detected objects and predicted tracks. Moreover, the associated tracks are updated with relevant detection information. All unmatched detections are placed in a list of tentative detections and monitored for a predetermined number of frames before the assignment of a new track. In the case of unmatched tracks, they are marked as unmatched and removed from the tracking list following a specified number of consecutive missed detections. Finally, the updated tracking information, encompassing object identities, bounding boxes, and class labels, is returned for visualization or further processing.

Kalman Filter

The Kalman filter estimates the current frame's location and motion of an object based on its position and movement in the preceding frame. This takes the form of an eight-dimensional vector, $(x, y, a, h, vx, vy, va, vh)$ [15]. This vector comprises the bounding box's central coordinates (x, y), the aspect ratio a, height h, and their corresponding velocities in the image coordinates [15]. DeepSORT utilizes the bounding box coordinates (x, y, a, h) as immediate observations of the object state.

Each track in DeepSORT is associated with a counter that increments during Kalman filter prediction and resets to zero upon successful association with a measurement. Tracks surpassing a predefined maximum age, the hyperparameter "max_age", are considered to have exited the footage, leading to their deletion [15]. New track hypotheses are created for detections unassociated with existing tracks. During the initial three frames (which is a hyperparameter), these newly created tracks are categorized as tentative, with the expectation that suitable measurement associations will be made at each frame. Any tracks that do not successfully associate with a measurement within their initial three frames are subsequently deleted.

As illustrated in Figure 2.16, the Kalman filter utilizes the Mahalanobis distance as a metric to evaluate the discrepancy between the predicted Kalman state of an object and the new detections in the current frame, which enables the effective linkage of a track with a corresponding detection.

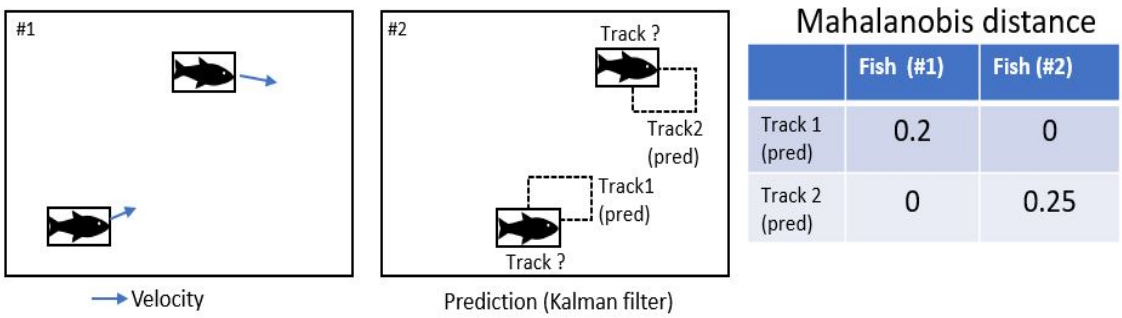


Figure 2.16: The figure illustrates the Kalman filter utilizing the Mahalanobis distance. The Kalman filter is used in DeepSORT to estimate the state of the tracked objects by incorporating both the current detection and the historical motion information.

Deep Appearance Descriptor

Beyond the use of the Kalman filter, DeepSORT incorporates an additional metric into the assignment problem to tackle issues related to occlusion and track switches [15]. In DeepSORT, the Deep Appearance Descriptor computes an appearance descriptor for each bounding box detection, generating distinctive features of the identified objects. It employs a CNN pre-trained on an extensive person re-identification dataset [53], encompassing more than 1,100,000 images featuring 1,261 distinct objects. It quantifies the dissimilarity between two vectors by determining the cosine of the angle between them, assessing their similarity based on the vectors' orientation rather than their magnitude.

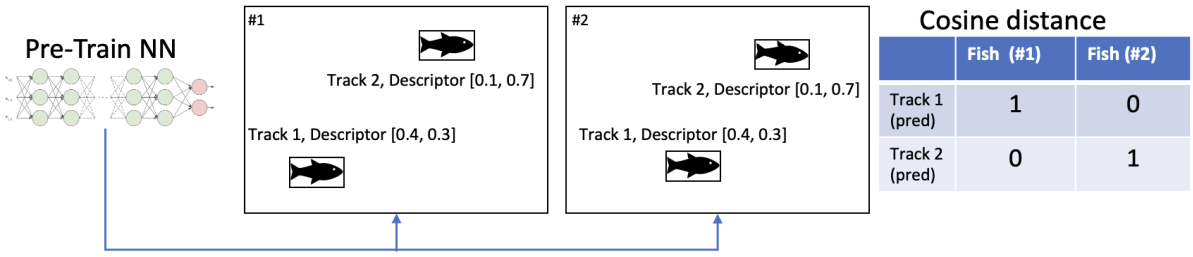


Figure 2.17: The Deep Appearance descriptor utilizes the cosine distance to measure the similarity between the detection and tracks.

The Hungarian Algorithm

To construct the association problem solvable by the Hungarian algorithm [54], the motion predictions from the Kalman filter and the appearance descriptor from the Deep Appearance Descriptor are combined into a cost matrix [15], as illustrated in Figure 2.18.

Mahalanobis distance

	Fish (#1)	Fish (#2)
Track 1 (pred)	0.2	0
Track 2 (pred)	0	0.25

+

Cosine distance

	Fish (#1)	Fish (#2)
Track 1 (pred)	1	0
Track 2 (pred)	0	1

=

Cost Matrix

	Fish (#1)	Fish (#2)
Track 1 (pred)	1.2	0
Track 2 (pred)	0	1.25

Figure 2.18: DeepSORT combines the Kalman filter and the appearance descriptor to construct an assignment problem.

The Hungarian algorithm, by utilizing the cost matrix, effectively tackles the data association challenge. This problem involves associating detected objects in the current frame with the predicted tracks from previous frames, with the constraint that each detection can only be assigned to a single track, and vice versa. The algorithm computes the optimal assignment that minimizes the total cost, considering both motion-based and appearance-based information. This robust assignment process contributes to the overall effectiveness of the DeepSORT algorithm in real-world tracking scenarios, ensuring that the tracker can adapt to changing conditions and maintain a reliable association between detected objects and their corresponding tracks.

2.4 Evaluation Metrics

In this section, an overview of the criteria employed for selecting the object detection models will be provided. Additionally, a comprehensive discussion of the various metrics utilized for assessing the performance of the tracking algorithms will be presented.

2.4.1 Metrics for Object Detection

Intersection over Union (IOU)

The Intersection over Union (IOU) is a widely used metric for classifying predicted bounding boxes. It measures the overlap between the predicted bounding box and the ground-truth bounding box by calculating the ratio of their intersection to their union. The IOU is scale-invariant, which means that it can be used to evaluate detection accuracy regardless of the size of the objects being detected [55].

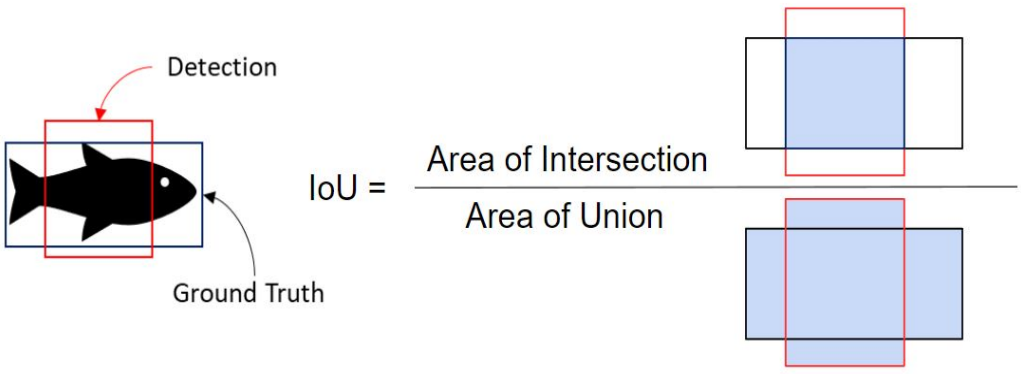


Figure 2.19: Definition of IOU.

Mean Average Precision (mAP)

The mean average precision (mAP) is typically utilized as a metric for assessing the performance of object detection models [56]. The mAP calculates the average precision (AP) for all classes in a multi-class problem. To comprehend the mAP metric, it is essential to understand three key concepts: precision, recall, and the aforementioned IOU.

Table 2.1: Confusion Matrix

	Predicted Positive	**Predicted Negative**
Real Positive	True Positive (TP)	False Negative (FN)
Real Negative	False Positive (FP)	True Negative (TN)

From the confusion matrix displayed in Table 2.1, precision and recall can be derived [57]. The precision (Equation 2.7) measures the accuracy of the model's predictions, that is, in the context of object detection, it indicates the fraction of the predicted bounding boxes that were correctly predicted based on the IOU threshold. The recall (Equation 2.8) measures the accuracy of positive predictions, meaning it quantifies the fraction of accurately predicted bounding boxes relative to all actual bounding boxes.

$$Precision = \frac{TP}{TP + FP} \tag{2.7}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.8}$$

Figure 2.20 illustrates the relationship between the IOU threshold and the true positive (TP), false positive (FP), and false negative (FN) detections with a confidence threshold of 0.5. In object detection, a TP detection occurs when the IOU is above the threshold, indicating a successful detection. Conversely, a detection with an IOU below 0.5 is considered an FP, indicating a false detection. If an object is not detected at all, it is classified as a FN. These thresholds play a crucial role in evaluating the precision and recall of the object detection model. Achieving an appropriate balance between precision and recall is essential for accurate and reliable object detection.

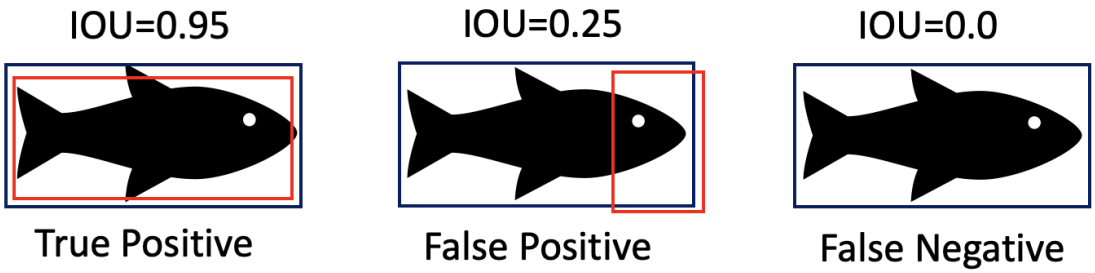


Figure 2.20: The figure illustrates the relationship between the IOU threshold and the true positive (TP), false positive (FP), and false negative (FN) detections. The detections are evaluated using a confidence threshold of 0.5.

To calculate the mAP value, a precision-recall (PR) curve must be generated[58]. This is accomplished by having the model generate a set of predictions for the test data, which includes the class labels, confidence scores, and bounding box coordinates for each detected object. Next,

an IOU threshold is chosen for the confidence score, and the predictions above this threshold are considered TP. By varying the confidence score threshold, a set of precision-recall pairs is generated, which can be plotted on a graph to form a PR curve. The area under the PR curve (AUC) provides a single scalar value that summarizes the overall performance of the model. Higher AUC values indicate better model performance [57]. The formula for AP is provided below:

$$AP = \sum_{k=1}^{n} P(k) \times \Delta r(k) \tag{2.9}$$

The equation calculates the AP by summing the product of precision ($P(k)$) and the difference in recall levels ($\Delta r(k)$) for each data point k, up to the total number of data points n. In this thesis, we will use the term mAP instead of AP for consistency, even though our problem focuses on a single-class problem.

mAP50-95

In addition to the commonly used mAP50 object detection metric, the mAP50-95 is also a noteworthy evaluation metric. It measures the mAP of examples across a range of IOU thresholds, spanning from 0.5 to 0.95 with a stride of 0.05 [59]. This metric provides a more comprehensive assessment of an object detection algorithm's performance, as it takes into account various levels of overlap between the predicted bounding boxes and the ground truth annotations. By evaluating the detection performance at multiple IOU thresholds, the mAP50-95 metric offers a deeper understanding of the algorithm's ability to accurately and precisely localize objects within the scene.

2.4.2 Metrics for Object Tracking

The effectiveness of the DeepSORT tracking algorithm is evaluated using several metrics provided by the MOTMetrics library, as described in Section 3.7. As seen in Equation 2.10, the IOU hyperparameter used in MOTMetrics is inverse to the IOU threshold used for detection. Therefore, an IOU of 0 indicates perfect overlap between bounding boxes, while an IOU of 1 implies no overlap between them.

$$\text{IOU} = 1 - \frac{\text{Area of Intersection}}{\text{Area of union}} \tag{2.10}$$

Multi-Object Tracking Accuracy (MOTA)

When using datasets with a high number of objects, occlusion, and identity switches between objects can occur, making it challenging to accurately track objects. To address these issues, it is important to use a performance metric that considers these factors. In the field of computer vision for multi-object tracking (MOT), Multi-Object Tracking Accuracy (MOTA) is a widely used metric for such tasks [60]. The formula for MOTA is given by Equation 2.11, where T represents the total number of frames in an image stream. Within each frame t in the set of all frames T, the number of FP is denoted as FP_t, the number of FN is denoted as FN_t, the number of identity switches is denoted as $IDSW_t$, and the total number of ground truth detections is denoted as GT_t.

$$\text{MOTA} = \left[1 - \frac{\sum_{t=1}^{T}(FP_t + FN_t + IDSW_t)}{\sum_{t=1}^{T} GT_t}\right] \cdot 100\% \tag{2.11}$$

MOTA values span from -∞ to 1, with 1 signifying high accuracy, while MOTA values near zero or lower indicate poor accuracy. In tracking evaluation, MOTA is usually used in conjunction with other metrics to provide a more comprehensive evaluation, such as Multiple Object Tracking Precision (MOTP).

Multi-Object Tracking Precision (MOTP)

MOTP is a metric used to evaluate the localization accuracy of an algorithm in MOT, seen in Equation 2.12.

$$\text{MOTP} = \left[\frac{\sum_{t,i} d_{t,i}}{\sum_t c_t}\right] \cdot 100\% \tag{2.12}$$

where $d_{t,i}$ is the error of the detected bounding box positions for matches objects averaged over the number of matches, c_t, in frame "t" and match "i". In essence, MOTP calculates the average distance between the predicted and ground truth bounding boxes, where a lower MOTP indicates a better detection accuracy [61]. The MOTP values range from 0 to 1, where a value near zero indicates high precision. To clarify, if the ground truth and predicted bounding box overlap perfectly (IOU=1), the distance (1-IOU) becomes 0, resulting in a MOTP value of zero, indicating accurate bounding box localization.

Identification F1 (IDF1)

Whilst MOTA and MOTP are built on matching at a detection level, the MOTMetrics library also contains identity metrics emphasizing the data association aspect of tracking. One such identity metric is the Identity F1 (IDF1) metric seen in Equation 2.15. This metric evaluates the tracker's ability to maintain correct target identities over time, rather than assessing the frequency of mismatches. The IDF1 metric combines identity-based precision (IDP) shown in Equation 2.13 and identity-based recall (IDR) shown in Equation 2.14 [62].

To calculate IDP, IDR, and IDF1, the library defines a new set of detection matches: Identity True Positives (IDTPs), Identity False Positives (IDFPs), and Identity False Negatives (IDFNs). IDTPs correspond to matches (TPs) that occur in the intersecting portion of matched predicted and ground truth trajectories. IDFPs consist of the leftover predicted detections (FPs) originating from the non-intersecting parts of matched trajectories as well as from the trajectories that remain unmatched. Similarly, IDFNs comprise the remaining ground truth detections (FNs).

Whilst the IDP measures the fraction of ground truth detections that are correctly assigned to a unique identity, the IDR measures the proportion of ground truth detections that are correctly identified by the tracking model [62]. The higher the IDF1 score, the better the tracking algorithm is at maintaining the correct identities of objects throughout the video. The definition of IDP, IDR, and IDF1 are as follows:

$$IDP = \frac{IDTP}{IDTP + IDFP} \tag{2.13}$$

$$IDR = \frac{IDTP}{IDTP + IDFN} \tag{2.14}$$

$$\text{IDF1} = \frac{2 \cdot \text{IDTP}}{2 \cdot \text{IDTP} + \text{IDFP} + \text{IDFN}} \tag{2.15}$$

Other Evaluation Metrics

In addition to the three previously mentioned metrics, several other evaluation measures have been employed to assess the performance of tracking models:

- **Identity switches (IDSW):** Scalar that counts the number of times the DeepSORT tracker switches the identity label of the same object in the ground truth data during the tracking process. This metric is used to evaluate the performance of the DeepSORT tracking algorithm with respect to maintaining consistent object identities over time. A higher number of switches indicates that the algorithm is struggling to maintain accurate object identities.
- **Mostly tracked (MT):** Calculates the number of objects that are tracked more than 80% of its lifespan.
- **Partially tracked (PT):** Calculates the number of objects that are tracked between 20% and 80% of its lifespan.
- **Mostly lost (MS):** Calculates the number of objects that are tracked less than 20% of its lifespan.

Chapter 3

Method

This chapter delves into the datasets utilized in this thesis, providing an overview of the data annotation and pre-processing procedures, including guidelines for various annotation scenarios. Furthermore, the chapter details the implementation of the two object detection algorithms and their integration with DeepSORT. Both detectors were trained on the same training set (Channel 6) and tested on two different test sets (Channel 3 and Channel 6). The models were implemented using the Python programming language, and the implementation was conducted on the Google Colaboratory platform. Furthermore, the chapter addresses the challenges faced during the process and the approaches adopted to overcome them.

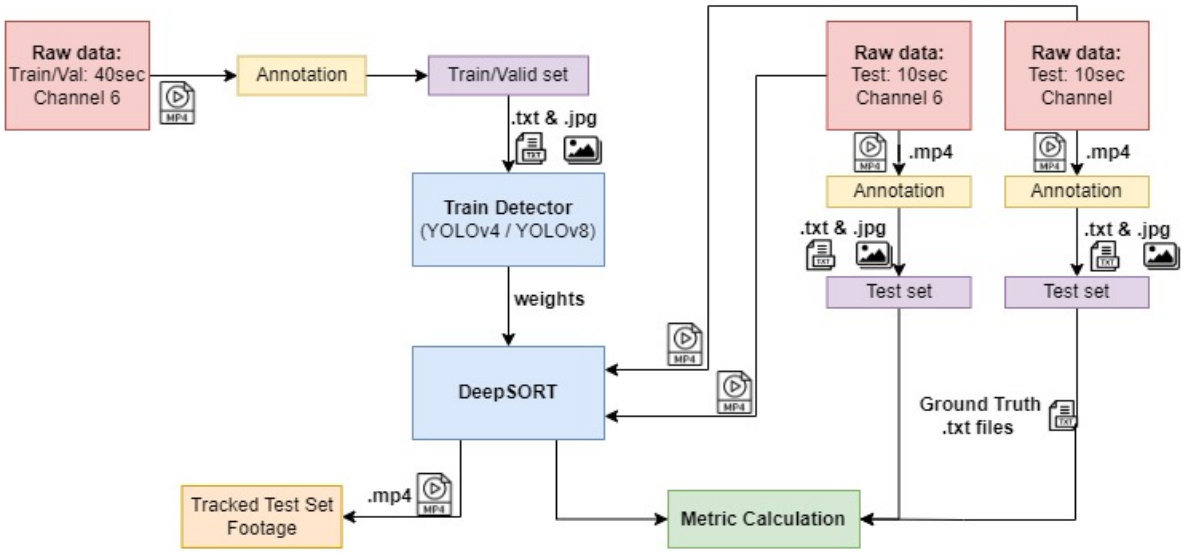


Figure 3.1: Illustration of the technical workflow of this thesis.

Figure 3.1 displays the technical workflow used in this thesis. It began with the acquisition of three distinct sets of data: a 40-second training set from Channel 6, a 10-second test set from Channel 6, and a 10-second test set from Channel 3. Each of these sets was manually annotated by a company named LabelYourData. The 40-second training data was subsequently utilized to train the detectors. The output of the detectors comprised the trained weights, which were then provided as input to the DeepSORT tracker, alongside the test sets in video format (.mp4). The text files generated from DeepSORT, in addition to the ground truth text files, were then utilized to calculate the performance metrics of the tracking algorithm.

3.1 Software and Hardware

The Google Collaboratory Pro environment provided access to powerful computational resources, including GPUs, which significantly accelerated the training and evaluation processes. Google Collaborator is a cloud-based service developed by Google that provides users with a collaborative platform for coding and data analysis. The service is based on an IPython kernel, which allows users to run code and execute commands in a Python environment. The service is widely used in data science, machine learning, and other fields that involve coding and data analysis. The efficiency of training the detectors was further increased by utilizing the Professional version of Collaboratory (Google Colab Pro). This version provides additional features and resources such as increased GPU and RAM allocation, as seen in Table 3.1, which significantly improved the speed during model training of the detector algorithms.

Table 3.1: While training the detectors, the Google Colab Pro version was employed, utilizing the Tesla A100-SXM4-40GB. However, for inference, the standard Google Colab with Tesla T4 was used to avoid exhausting the limits of the Pro version.

Training/Inference	**System**	**GPU name**	**RAM**
Model Training	Google Colab Pro	TESLA A100-SXM4-40GB	40GB
Inference	Google Colab	TESLA T4	16GB

3.2 Dataset

The datasets used in this thesis are provided by Christopher Noble at Nofima. There were six channels or videos provided to us in this thesis. In the context of this study, the term "channel" refers to separate video feeds or sources, not to be mistaken with color channels (Red, Green, Blue) typically associated with digital image processing. The channels were provided in .mp4 format and captured using an underwater camera, each with a duration ranging from three to six minutes. Unfortunately, a significant portion of the videos proved to be challenging to use in this thesis, as some of the fish were positioned directly in front of the lens, obscuring the view of all other fish. Nevertheless, as seen from Table 3.2, a 40-second segment from Channel 6 was selected as a training set, with the initial 35 seconds designated for training and the following 5 seconds for validation. The choice to use continuous 35-second footage for training was driven by the absence of a basis for selecting footage from multiple intervals within the channel. The chosen 40-second video featured fish movement across the frames, ensuring an adequate level of data variability for training purposes.

Moreover, two 10-second videos were extracted as test sets: one from the same channel (Channel 6) and another from Channel 3. The test set from Channel 3 was chosen to assess the models' generalization capabilities since it originated from a different fish tank. This ensured that the models were not overfitting to the training data from Channel 6 and could effectively generalize to previously unseen data. The inclusion of this additional test set allowed for a more comprehensive evaluation of the models' performance, bolstering confidence in their ability to accurately detect and track fish in various environments.

Table 3.2: The table presents a summary of the video channels and their corresponding durations used in this study. The first 35 seconds of Channel 6 were utilized as training (875 frames) and the subsequent 5 seconds were utilized as a validation set (125 frames). The remaining 10-second segment from the same channel was used as a test set, comprising 250 frames. Additionally, a 10-second video from Channel 3 was employed as a second test set, containing another 250 frames.

Video Channel	**Data split**	**Duration**	**Number of Frames**
Channel 6	Train	35s	875
Channel 6	Validation	5s	125
Channel 6	Test	10s	250
Channel 3	Test	10s	250

The dataset used in this thesis was recorded using an underwater camera that captures 25 frames per second with a resolution of 720 pixels in width and 576 pixels in height. Figure 3.2 displays the environment present in Channel 6 and Channel 3, respectively. Both the extracted frames show environments full of fish, which underscores the relevance of the challenges highlighted in Section 2.2 by Shakik et al.

A higher number of fish present in a frame leads to more opportunities for objects to be occluded. Occlusion can affect the performance of the algorithm as it can lead to more identity switches during object tracking. Given that fish exhibit high mobility and possess the ability to move in any direction in their environment, accurately tracking their trajectories presents a considerable challenge. In addition to the high number of fish, several other factors may impact the performance of the detection and tracking algorithms. These factors encompass variations in lighting conditions, water turbidity, and fish behavior.

In both channels, the environments exhibit darker regions where the fish and the background closely resemble each other, posing possible challenges for the algorithms in accurately distinguishing between them. Moreover, it's essential to acknowledge that the data quality is relatively low, which is a significant factor when considering the necessity for precise detection and tracking. These factors collectively contribute to the complexity of the objectives, necessitating the development of robust and adaptable algorithms to ensure precise object detection and tracking in underwater environments.

3.3 Data Annotation

The datasets were initially not intended for computer vision applications. Therefore, to enable the algorithms used in this study to comprehend the raw data, it was necessary to manually annotate all the 1500 frames presented in Table 3.2. Data annotation, also known as labeling, involves the process of identifying and marking relevant objects within the data that are useful for the algorithms to learn. In the context of this thesis, data annotation specifically refers to the identification and labeling of all fish present in the training, validation, and test sets. Various annotation techniques were explored, with bounding box annotation emerging as the most practical option. As illustrated in Figure 3.3b, bounding box annotation involves enclosing the object of interest within a rectangular box to indicate its position and size.

Manually annotating all objects in the datasets used in this thesis would have been a labor-intensive process, demanding considerable effort and diverting focus from object detection and

(a) Raw data from Channel 6. (b) Raw data from Channel 3.

Figure 3.2: The figures depict raw data from Channel 6 and Channel 3, illustrating the differences in the environmental conditions between the two channels.

tracking to data annotation. To tackle this challenge, the annotation task was outsourced with financial support from Nofima and the Faculty of Science and Technology at NMBU. The annotation service, LabelYourData, utilized the Computer Vision Annotation Tool (CVAT) [63] for the annotation process. CVAT is known for its user-friendly, efficient, and intuitive interface, making it a suitable tool for the annotation task.

A key feature of CVAT is its interpolation capability which automatically calculates linear changes in the size and position of bounding boxes between two frames. This functionality removes the need for making minor adjustments for each frame, thus accelerating the annotation process. As LabelYourData utilized interpolation, the allocated funds were sufficient to label all the data presented in Table 3.2. Outsourcing the annotation task proved essential in optimizing the use of time and resources, enabling us to concentrate on the primary objectives of this thesis, namely the detection and tracking of fish.

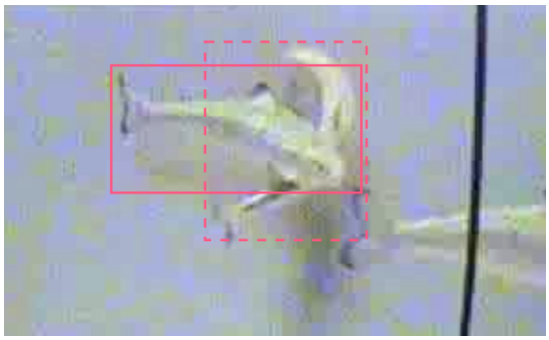
(a) Raw data provided by Christopher Noble. (b) Bounding box annotation.

Figure 3.3: The figures display a comparison between the frame before and after annotation. The annotated bounding boxes are represented by solid lines, while the dotted lines indicate occluded objects.

3.3.1 Guideline for annotation

As previously noted, the datasets are densely populated and include multiple different scenarios during annotation. Moreover, the provided videos only capture a section of the fish tanks, causing fish to continuously enter and leave the frame. Since the quality of annotations influences the performance of detection and tracking algorithms, it is essential to ensure consistent and accurate labeling of the data. Consequently, an annotation guideline was prepared to serve as instructions for LabelYourData during the labeling process, with the aim of minimizing errors and achieving high-quality annotations. The guidelines included the following points:

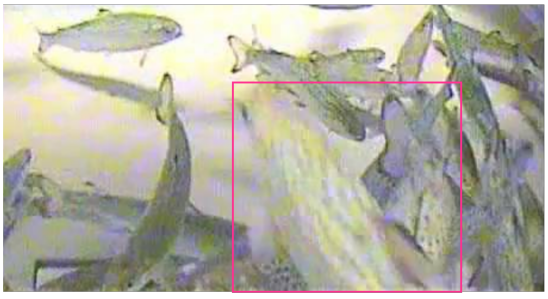
- A new track ID was assigned to every fish that enters or re-enters the frame.
- A fish should be annotated only when it is distinctly visible.
- If a fish is fully occluded, it should not be annotated.
- If a fish is partially occluded, it should be annotated with an occluded bounding box, as shown in Figure 3.5a.
- When more than 50% of the fish is visible, it is advised to annotate the visible part of the fish with a normal bounding box, as seen in Figure 3.4d. For instance, if the largest part of the front section of the fish is in-frame, this part should be annotated with a normal bounding box.



(a) Occluded fish annotated using an occluded bounding box.

(b) The fish is no longer occluded, hence annotated using a normal bounding box.

(c) The fish entering the frame is not visible enough for annotation.



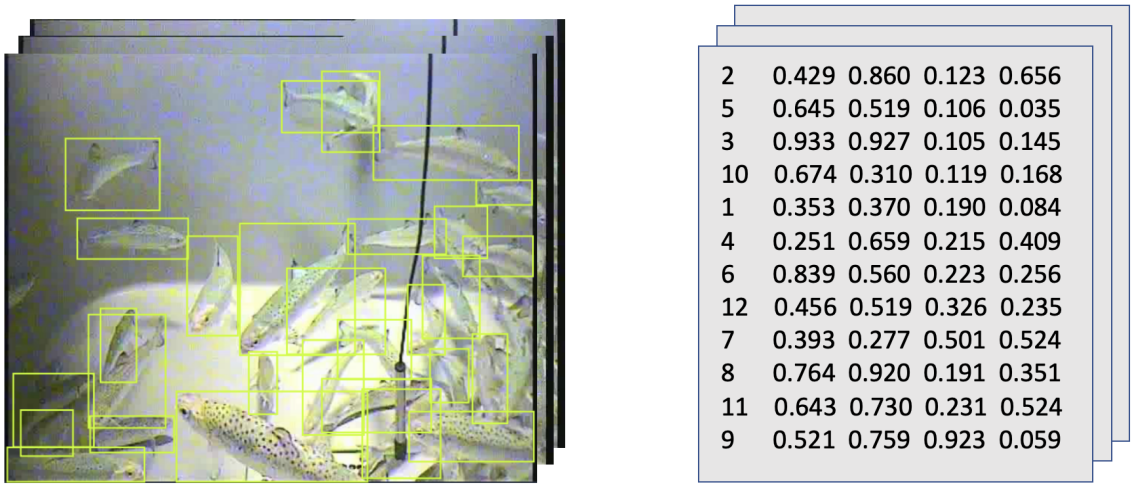
(d) The same fish is visible, permitting a standard bounding box annotation.

Figure 3.4: Images illustrating examples from the annotation guideline.

3.3.2 YOLO data format

Following the completion of the annotation process by LabelYourData, CVAT generated two types of outputs: the extracted video frames and a COCO JSON file. The JSON file contained properties such as "image_id", "class_id", "occluded", and "bbox", which included the x and y coordinates of the bounding box's center, as well as its height and width (x, y, width, and height). However, since YOLO detection algorithms do not support the COCO JSON format, it was necessary to convert the annotated data into the correct format. The Roboflow platform offered a conversion tool to facilitate this process. The resulting YOLO format consisted of separate text files for each image frame, which included the class label and the bounding box coordinates for all objects in that specific frame:

<object_id> <x> <y> <width> <height>



(a) (b)

Figure 3.5: Figure (a) displays the individual frames and Figure (b) shows the corresponding text files on YOLO format.

The YOLO detection algorithm, like most other object detection algorithms, does not provide explicit support for handling occlusion during detection. Figure 3.6 illustrates the output from Roboflow, which consists of four folders corresponding to each dataset. Within each folder, every image file must have a matching text file that shares the same filename.

Figure 3.6: The folder structure output from RoboFlow. To ensure accurate matching of bounding box annotations with their respective images during training and validation, it is important that each image file has a corresponding text file with the same filename.

3.4 YOLOv4

YOLOv4, an anchor-based detector introduced in Section 2.2.2, was chosen as one of the detectors for this thesis due to its efficiency and effectiveness in real-time detection. YOLOv4 is based on the Darknet framework [64] and utilizes the CSPDarknet53 CNN architecture. The open-source framework is written in C that operates on GPUs through CUDA. All previous YOLO versions, including YOLOv4, runs on this framework. The requirements for running YOLOv4 can be found in Appendix A.

3.4.1 YOLOv4 Model Training

In this thesis, transfer learning was employed by utilizing pre-trained weights (yolov4.conv.137) of YOLOv4 on the MS COCO dataset [65]. The MS COCO dataset comprises 80 distinct classes and 123,287 images, equipping the model with a robust basis for object detection tasks [65]. By leveraging the knowledge learned from a large-scale dataset, the YOLOv4 model can effectively initialize its parameters with meaningful weights. This initialization allows the model to start with a good understanding of general object features and improves its ability to detect and track objects in the specific domain of interest, in this case, fish detection. As elaborated in Section 2.1.5, employing transfer learning strategies can lead to a substantial reduction in training time while simultaneously enhancing model performance, requiring relatively fewer annotated training data.

Hyperparameters in YOLOv4

YOLOv4 comes with a set of default hyperparameters that can influence the learning process and model performance. Fine-tuning these parameters may enhance the model's accuracy. However, optimizing hyperparameters in object detection projects can be a challenging and resource-intensive task. In YOLOv4, hyperparameter tuning involves modifying one parameter at a time, executing the algorithm, and evaluating the performance. To validate the improvements, it is necessary to repeat this process multiple times. However, due to time and financial constraints, primarily the need for Google Colab Pro, comprehensive hyperparameter tuning was not conducted in this thesis. Appendix B provides a list of the default hyperparameters for YOLOv4, along with brief explanations for each.

Data Augmentation in YOLOv4:

Apart from the selection of hyperparameters, data augmentation also plays a crucial role in the model's performance. Overfitting in computer vision is caused by having too few samples to learn from, leading to a model that does not generalize well to new and unseen data [21]. Data augmentation mitigates this issue by artificially increasing the size of the training dataset through techniques such as flipping, cropping, and rotating the existing images, allowing the model to learn from a larger and more diverse set of data. Two important data augmentation techniques used in YOLOv4 are listed below.

- **Mosaic**: Involves randomly selecting four pictures from the training images and merging them into a single mosaic image used for training. The corresponding bounding boxes are adjusted to reflect the object's new position in the image [66].
- **Mixup**: A technique that blends two images by multiplying and superimposing them with different ratios. The resulting image is then used to train the model, with the label adjusted accordingly based on the blending ratios [13].

Upon selection of data augmentation techniques and the specification of hyperparameters, the subsequent step was to train the YOLOv4 model on the training data. Despite the use of transfer learning, the process remains quite resource-intensive and demands a substantial amount of time and computational power. During the training process, the model continually evaluates its own performance on the validation set, which serves as a basis for selecting the optimal model when the training is terminated. During the training process, the model weights were saved every 1000 iterations, along with the best weights obtained based on the highest validation mean Average Precision (mAP) score.

3.5 YOLOv8

The second detection algorithm utilized in this thesis, YOLOv8, is the latest addition to the YOLO family, and its architecture was introduced in Section 2.2.3. In contrast to YOLOv4, YOLOv8 uses a decoupled head with an anchor-free structure. Utilizing an anchor-free method, the algorithm is able to directly estimate the object class and location information based on the output generated by the network. In this thesis, the Ultralytics YOLOv8.0.20 version is used and it is installed via pip in Google Colab.

3.5.1 YOLOv8 Model Training

During the training process, YOLOv8 provides five scaled versions of the YOLO model with varying speeds and accuracies, pre-trained on the MS COCO dataset [67]. The smallest and fastest version is YOLOv8 Nano (YOLOv8n), while the largest and most accurate version is YOLOv8 Extra Large (YOLOv8x). The other three versions are YOLOv8 Small (YOLOv8s), YOLOv8 Medium (YOLOv8m), and YOLOv8 Large (YOLOv8l), each with a unique balance between speed and accuracy. The availability of these scaled versions allows users to choose the most appropriate version for their specific use case, depending on the trade-off between speed and accuracy that they require. In this thesis, the YOLOv8-small was utilized due to the balance between computational efficiency and detection performance. The default hyperparameters used during YOLOv8 training are listed in Appendix C with an accompanying explanation. Lastly, YOLOv8 uses the same data augmentation techniques as YOLOV4.

3.6 DeepSORT

After the completion of the training process for each of the two detectors, the obtained weights are separately utilized during the detection phase of DeepSORT, providing two distinct models for object tracking. As outlined in Section 2.2, the DeepSORT algorithm encompasses a dual structure with a detection part and a tracking part, commonly denoted as a tracking-by-detection algorithm. Initially, DeepSORT employs its integrated detection algorithm to identify and locate objects as seen in Figure 3.7. Subsequently, the tracking phase of DeepSORT is executed using the Kalman filter, Deep Association Matrix, and the Hungarian algorithm, ensuring the accurate tracking and preservation of each object's identity over time through the assignment of unique identifiers.

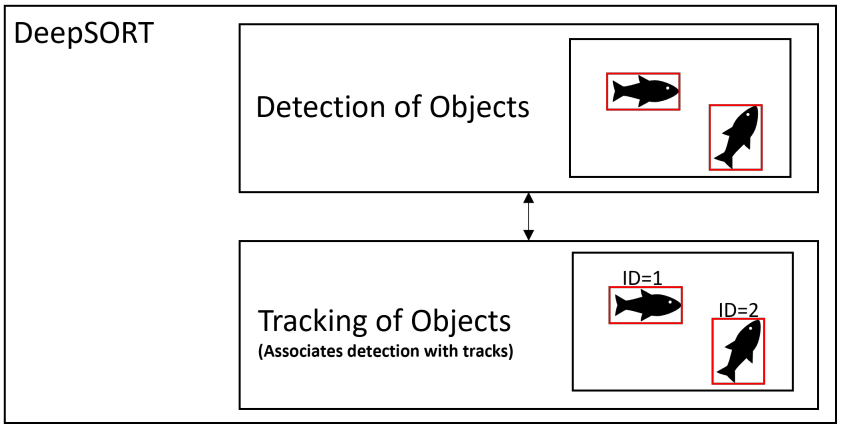


Figure 3.7: For a tracking-by-detection algorithm, a detection algorithm is required to detect objects before the DeepSORT tracking stage assigns unique identities to each detected object.

3.6.1 Hyperparameters in DeepSORT

Similar to the previously mentioned detectors, DeepSORT also contains a set of hyperparameters that can be adjusted to enhance the tracking performance. The default hyperparameters and a brief explanation of them are listed in Appendix D.

Unlike detection models, DeepSORT doesn't require any training process, and adjusting its hyperparameters during the inference stage is considerably faster than the training phase of the detectors. The trained weights obtained from the detectors, along with a test set containing a .mp4 video footage file, are used as inputs to the DeepSORT tracking algorithm. The output of this algorithm is a video file containing the input video with tracking properties.

Since the original DeepSORT model did not provide the functionality to export text files containing detection bounding box information (tracking id, x, y, w, h), it was necessary to modify the DeepSORT implementation to facilitate the evaluation of the two test sets (Channel 3 and Channel 6) against the ground truth. These modifications allowed the algorithm to produce prediction bounding boxes in the test sets utilizing the YOLO file format. The added code can be accessed in Appendix E.

3.7 Evaluation using MOTMetrics

To evaluate and compare the two tracking models, we implemented a series of evaluation metrics as described in Section 2.4. For metric calculations, we utilized the MOTMetrics library (version 1.4.0), installed via pip. This library necessitates ground truth text files for each frame, in addition to the detection text files, which were made feasible due to the aforementioned DeepSORT code modifications.

To employ MOTMetrics, we organized two folders containing text files. The first folder holds ground truth text files, while the second folder stores tracking text files extracted from DeepSORT. Both folders adhere to the YOLO format, as demonstrated in the two corresponding text files in Figure 3.8.

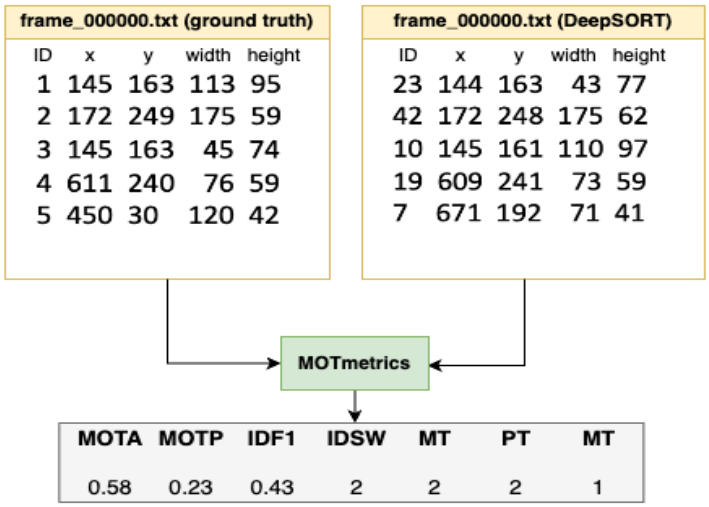


Figure 3.8: The figure illustrates an evaluation between a ground truth file from the ground truth folder and a corresponding frame from the output DeepSORT folder.

To systematically calculate the tracking metrics, the function provided in Appendix F processes every corresponding text file from the two folders using MOTMetrics. It uses the IOU metric to match the objects from the DeepSORT output folder to the ground truth folder for each frame. The object being tracked is deemed a match when it exhibits an IoU score that exceeds a specified threshold value with a corresponding ground truth object. In this thesis an IOU threshold of 0.5 is used, signifying a 50% overlap is required for a detection to be paired with a ground truth object. After the objects are matched, MOTmetrics proceeds to calculate the tracking metrics introduced in Section 2.3. The evaluation is used to determine which model combination is the most effective at tracking fish in an underwater environment, specifically comparing the performance of DeepSORT when utilizing either YOLOv4 or YOLOv8 as the detector.

Chapter 4

Results

This chapter provides an in-depth analysis and discussion of the results obtained from the models presented in this thesis, in line with the objectives outlined in Chapter 1. Section 4.1 presents and examines the quantitative outcomes of the YOLOv4 and YOLOv8 object detectors, followed by a comparison and discussion of the detectors combined with the DeepSORT tracking algorithm. The similarities and differences are emphasized, offering valuable insights into each method and the impact of the detectors on object tracking in the test videos. Section 4.2 supplement the quantitative analysis by delving into the qualitative distinctions between the object detectors and the object tracking models. This analysis explores the visual differences, performance variations, and overall characteristics exhibited by each model, leading to a deeper understanding of their capabilities. Through the extensive analysis and discussion presented in this chapter, informed conclusions can be made regarding the effectiveness of the object detection and object tracking models in fulfilling the thesis objectives.

4.1 Quantitative Analysis

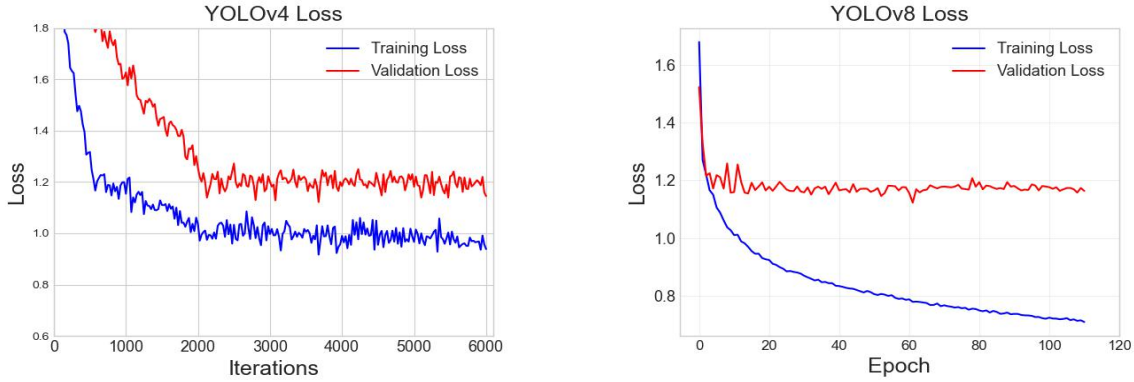
4.1.1 Quantitative Analysis of Detection Performance

The YOLOv4 object detection model was trained for 6000 iterations (375 epochs) with default hyperparameters in this thesis, as shown in Appendix B. The training process for the YOLOv4 model on the Google Colab servers, using the Pro subscription and Tesla A100 GPU, took slightly over five hours to complete.

On the other hand, the YOLOv8 object detection algorithm was trained for 100 epochs using the default hyperparameters specified in Appendix C. As the model did not converge within the specified number of epochs, the number was increased to 200 during the training process. To prevent overfitting and optimize the training process, an early stopping hyperparameter called "patience" was set to 50. This means that if the model did not show any improvement in the last 50 epochs, training would be stopped. In this particular case, the model stopped at 111 epochs since it had not demonstrated any improvement in performance during the last 50 epochs. The best-performing epoch was therefore found to be number 61. The training process was conducted on a Tesla A100 GPU provided by Google Colab Pro, and it took approximately two hours to reach completion.

Figure 4.1a presents the progression of the training and validation loss over the course of the YOLOv4 model's training process, as detailed in Section 2.2.2. It visually depicts how the model's capability to accurately identify fish within bounding boxes improves over time, as

evidenced by the reduction in training loss. Notably, the validation loss also exhibits a similar trend, decreasing over the course of training, which is an indicator of the model's improving generalization to unseen data. The validation loss reaches a minimum at around 3000 iterations, suggesting that the model's performance on the validation set is optimal at this point.

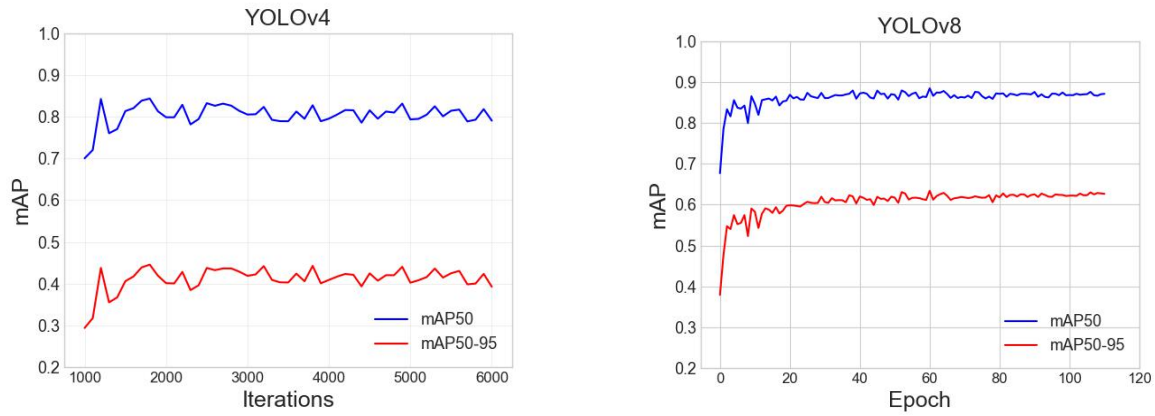


(a) (b)

Figure 4.1: The training loss and validation loss curves for YOLOv4 and YOLOv8 are depicted in Figure (a) and Figure (b), respectively.

Figure 4.1b illustrates the loss for YOLOv8 throughout the training process. Both the training loss and validation loss show a rapid decline in the initial epochs. However, although the validation loss stabilizes at approximately 1.2, the training loss keeps decreasing gradually. As previously stated, the optimal performance was attained at epoch 61. Figures showing the data augmentation during training is presented in Appendix G.

Furthermore, as explained in Section 2.4, mAP50 and mAP50-95 are among the metrics used for evaluating detection algorithms. For YOLOv4, as depicted in Figure 4.2a, the mAP50 value remains relatively stable at around 0.81, indicating that YOLOv4 is capable of detecting objects in approximately 81% of the validation images with a threshold (IOU) of 50%. However, when considering the more stringent mAP50-95 metric, YOLOv4's average value drops to 0.39, implying lower accuracy in localizing objects when higher overlaps are considered. The validation for both mAP50 and mAP50-95 in YOLOv4 begins after the first 1000 iterations, by default. This is primarily because the computation of mAP validation scores is resource-intensive and requires significantly more computational power compared to the standard training process.



(a) (b)

Figure 4.2: The mAP50 and mAP50-95 scores for YOLOv4 and YOLOv8 are shown in Figure (a) and Figure (b), respectively.

In contrast, based on the mAP50 metric shown in Figure 4.2b, YOLOv8 demonstrates a stronger validation performance than YOLOv4. The figure illustrates that the model achieves a maximum mAP50 score exceeding 0.88, indicating high precision in object detection. Additionally, with a mAP50-95 score of approximately 0.64, YOLOv8 exhibits the ability to more precisely identify and locate objects with higher IOU thresholds. This observation implies that YOLOv8 could be better suited to handle diverse detection situations in the underwater environments featured in the thesis datasets.

Table 4.1 provides a summary of the quantitative performance of YOLOv4 and YOLOv8 on the validation set, as well as two unseen test sets (Channel 3 and Channel 6). The table highlights consistent results for each detector across the datasets, while also uncovering distinctions between the two detectors. When comparing the performance on the validation set to the unseen data, it is evident that YOLOv8 surpasses YOLOv4, particularly in the mAP50-95 metric. This indicates that YOLOv8 exhibits greater precision in localizing the ground truth bounding boxes as previously discussed. Moreover, the higher precision scores of YOLOv8 indicate a reduced number of false positive predictions, demonstrating a lower tendency to detect nonexistent fish. Additionally, YOLOv8 demonstrates better recall, showcasing its ability to detect and localize actual fish in the frame.

Table 4.1: The table presents the detection metrics for the validation set during training, and the two previously unseen test sets during inference.

	YOLOv4			**YOLOv8**		
Metric	**Validation**	**Test Ch6**	**Test Ch3**	**Validation**	**Test Ch6**	**Test Ch3**
mAP50	84.3%	84.0%	83.3%	88.4%	88.2%	87.6%
mAP50-95	39.8%	39.2%	38.7%	64.4%	64.3%	63.6%
Precision	0.80	0.79	0.77	0.87	0.87	0.84
Recall	0.81	0.80	0.77	0.83	0.84	0.81

4.1.2 Quantitative Analysis of Tracking Performance

The detection models, YOLOv4 and YOLOv8, were integrated with the DeepSORT tracking algorithm, as explained in Section 3.6. The tracking performance was evaluated on the video test sets from Channel 3 and Channel 6, using the default hyperparameters of DeepSORT, listed

in Appendix D. The tracking results for each combination are summarized in Table 4.2.

Table 4.2: Tracking performance using DeepSORT combined with YOLOv4 and YOLOv8 on video data from Channel 3 and Channel 6 using default parameters.

Detector	**Dataset**	**MOTA**	**MOTP**	**IDF1**	**IDSW**	**MT**	**PT**	**ML**
YOLOv4	Channel 3	0.563	0.264	0.401	222	24	31	6
	Channel 6	0.533	0.267	0.391	236	23	31	8
YOLOv8	Channel 3	0.598	0.192	0.455	188	27	28	6
	Channel 6	0.556	0.185	0.466	213	30	27	5

Comparing the two tracking models, it is evident that the combination of DeepSORT with YOLOv8 outperforms DeepSORT with YOLOv4. The results show that tracking with YOLOv8 yields higher MOTA scores for both Channel 6 (0.556) and Channel 3 (0.598), indicating better overall tracking accuracy. Additionally, the lower MOTP values observed for tracking with YOLOv8 suggest higher location accuracy of predicted bounding boxes. The IDF1 values are also higher for tracking with YOLOv8, indicating better data association and object identity maintenance, which ultimately leads to enhanced tracking performance. It is worth noting that tracking with YOLOv8 results in an increased count of mostly tracked (MT) objects across both datasets and a significantly reduced number of mostly lost (ML) objects in the Channel 6 test set.

Furthermore, both tracking models were tested in which direction a parameter change would impact the models' performance, starting with YOLOv4. Table 4.3 presents values above and below the default settings for the "MaxAge" ("max_age"), "MaxIOUDist" ("max_iou_distance") and "IOU" parameters. Modifying these hyperparameters resulted in the most noticeable differences in tracking performance, unlike the other hyperparameters tested.

The most significant improvement was to reduce "MaxAge" to 30 frames from its default value of 60, resulting in higher MOTA and fewer identity switches. One second of the video consists of 25 frames, which means that 30 frames make up slightly more than one second of the video. Furthermore, the IDF1 score showed a significant increase, indicating improved data association and tracking performance. On the other hand, if the "MaxAge" value is increased, the tracker may retain tracks for a longer duration without matching to new detections before they are deleted, resulting in an increase in IDSW (identity switches). This leads to incorrect associations with new detections and increases the number of identity switches.

A slight modification to the "IOU" threshold showed a difference in the performance of the model. A decrease in "IOU" allows for more bounding boxes to be present, resulting in marginally higher MOTA and IDF1. On the other hand, increasing the IOU threshold resulted in a decrease in performance. These findings highlight the sensitivity of the model to the IOU threshold and the importance of selecting an appropriate value to optimize the tracking performance. Additionally, increasing the "MaxIOUDist" parameter improved the MOTA, MOTP, and IDF1 scores marginally. However, when combined with a reduced "MaxAge" parameter, it did not yield better performance compared to solely decreasing the "MaxAge" parameter.

Table 4.3: Tuning the hyperparameters to optimize the DeepSORT tracking algorithm utilizing YOLOV4 on the Channel 3 test set was carried out with the default hyperparameter performance as the reference point. The percentage indicates the change in performance relative to the reference point.

Metric	MOTA	MOTP	IDF1	IDSW	MT	PT	ML
$Channel3_{default}$	0.563	0.264	0.401	222	24	31	6
$MaxAge_{90}$	-0.6%	-0.23%	-1.8%	+119	0	0	0
$MaxAge_{30}$	+9.2%	-0.31%	+18.6%	-80	0	0	0
$IOU_{0.5}$	-0.17%	+0.17%	-0.5%	+20	-1	+1	0
$IOU_{0.4}$	+1.12%	-0.05%	+0.5%	-7	+2	-1	-1
$MaxIOUDist_{0.8}$	+0.12%	-0.03%	+1.3%	0	+1	-1	0
$MaxIOUDist_{0.6}$	-0.42%	-0.12%	-3.3%	+42	+1	0	-1
$MaxAge_{30}$ + $MaxIOUDist_{0.8}$	+7.7%	+0.3%	+15.2%	+38	+1	-1	0
$MaxAge_{30}$ + $MaxIOUDist_{0.6}$	+3.2%	+0.2%	+11.3%	-62	0	0	0
$MaxAge_{30}$ + $IOU_{0.4}$	+6.5%	-0.03%	+11.1%	-69	0	0	0
$MaxAge_{30}$ + $IOU_{0.5}$	+8.4%	+0.07%	+16.8%	-72	0	0	0

In Table 4.4, the performance of the DeepSORT tracking algorithm with YOLOv8 as the detector was evaluated by exploring the influence the hyperparameters have on the performance. The most substantial enhancement in tracking performance for DeepSORT with YOLOv8 was achieved by reducing the "MaxAge" parameter. This adjustment resulted in improved tracking accuracy, evidenced by higher MOTA scores and a significantly increased IDF1 score, aligning with the observations in DeepSORT with YOLOv4. Consequently, the primary focus for optimizing the DeepSORT algorithm with YOLOv8 should concentrate on reducing the "MaxAge" parameter while maintaining the "IOU" threshold at its default value of 0.45 and "MaxIOUDist" at 0.7.

Table 4.4: Tuning the hyperparameters to optimize the DeepSORT algorithm utilizing YOLOv8 on the Channel 3 test set was conducted with the default hyperparameter performance as the reference point.

Metric	MOTA	MOTP	IDF1	IDSW	MT	PT	ML
$Channel3_{default}$	0.598	0.192	0.455	188	27	28	6
$MaxAge_{90}$	-2.9%	+0.06%	-9.2%	+92	0	0	0
$MaxAge_{30}$	+10.6%	-0.23%	+19.1%	-78	-1	+1	0
$IOU_{0.5}$	-0.15%	-0.11%	-0.6%	+42	-1	+1	0
$IOU_{0.4}$	+1.55%	+0.23%	+1.3%	+28	+1	-1	0
$MaxIOUDist_{0.8}$	+0.35%	-0.04%	+2.1%	+92	0	0	0
$MaxIOUDist_{0.6}$	-0.22%	-0.32%	-2.5%	+56	0	+1	-1
$MaxAge_{30}$ + $MaxIOUDist_{0.8}$	+8.1%	+0.17%	+15.3%	-62	0	0	0
$MaxAge_{30}$ + $MaxIOUDist_{0.6}$	+2.9%	-0.03%	+11.2%	-16	-1	+1	0
$MaxAge_{30}$ + $IOU_{0.4}$	+3.5%	-0.42%	+12.1%	-53	-2	+2	0
$MaxAge_{30}$ + $IOU_{0.5}$	+8.9%	+0.16%	+16.4%	-68	-1	+1	0

The quantitative analysis highlights that DeepSORT with YOLOv8 outperforms DeepSORT with YOLOv4 in terms of MOTA, IDF1, and MOTP. These findings are consistent with the performance of the standalone object detectors, where YOLOv8 exhibits higher precision in localizing objects. Furthermore, the tuning of the trackers as seen in Table 4.3 and Table 4.4, suggest that changing the "MaxAge" hyperparameter to a lower value than the default (60

frames) can result in improved tracking performance, as indicated by higher MOTA and IDF1 scores. However, reducing the "MaxAge" parameter requires a trade-off between minimizing identity switches and improving occlusion handling capability, as a lower "MaxAge" value can affect object re-identification after occlusion if the occlusion period exceeds the "MaxAge" frames.

Although the number of identity switches is high for both tracking models, this information alone is insufficient to provide a comprehensive understanding of the underlying causes of these switches. Further investigation is necessary to determine the specific scenarios in which identity switches occur and the extent to which different factors, such as occlusion or object appearance changes, contribute to these switches.

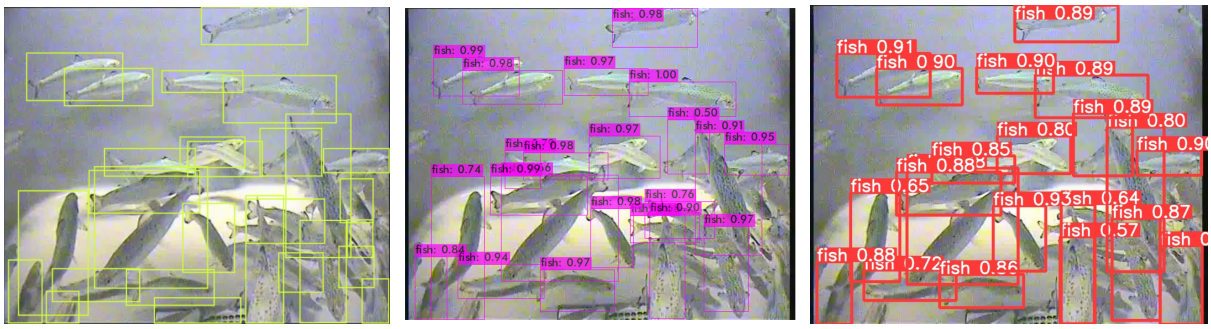
4.2 Qualitative Analysis

To gain a better understanding of the evaluation scores presented in Section 4.1, a qualitative analysis is conducted in this section. Specifically, we focus on the two detectors used in the thesis and investigate any differences they may have in detecting and localizing objects. Furthermore, we examine the tracking results to identify any potential causes of identity switches and assess the robustness of the tracking algorithm when dealing with challenging scenarios such as occlusions, object appearance changes, and crowded scenes. The insights gained from this analysis will help to inform further improvements to the tracking algorithm and guide the development of future work in this field.

4.2.1 Qualitative Analysis of Detection Performance

Figure 4.3a presents the ground truth frame, whereas Figure 4.3b and Figure 4.3c depict the same frame after being processed by YOLOv4 and YOLOv8, respectively. In Figure 4.3b, it can be observed that YOLOv4 identifies fish with high confidence in the dense environment; however, the precision of the bounding box localization has room for improvement. The detector exhibits some false negatives in the bottom right area of the frame, indicating its struggle to detect fish in this region. This leads to an increased false negative rate, which negatively affects the recall score. It is worth noting that the difficulty in detecting fish within this specific section of the frame could be attributed to the lack of high-quality input data.

In contrast, YOLOv8 generates detection outcomes that show a lower confidence score for the identified fish, as depicted in Figure 4.3c. However, YOLOv8 successfully detects a fish in the bottom right corner of the frame that YOLOv4 fails to identify. Although the bounding box localization for this fish is not perfect, YOLOv8's more conservative approach leads to a higher level of localization accuracy in capturing the overall locations of fish. Moreover, Appendix H provides a detailed analysis of the differences in confidence scores and bounding box localization between YOLOv4 and YOLOv8, contributing to the substantial difference in the observed mAP50-95 scores discussed in Section 4.1.1.



(a) Ground truth (frame 3) (b) YOLOv4 detection (frame 3) (c) YOLOv8 detection (frame 3)

Figure 4.3: Figure (a) shows the ground truth annotation for frame 3 in Channel 3, while Figures (b) and (c) display the output from DeepSORT using YOLOv4 and YOLOv8, respectively.

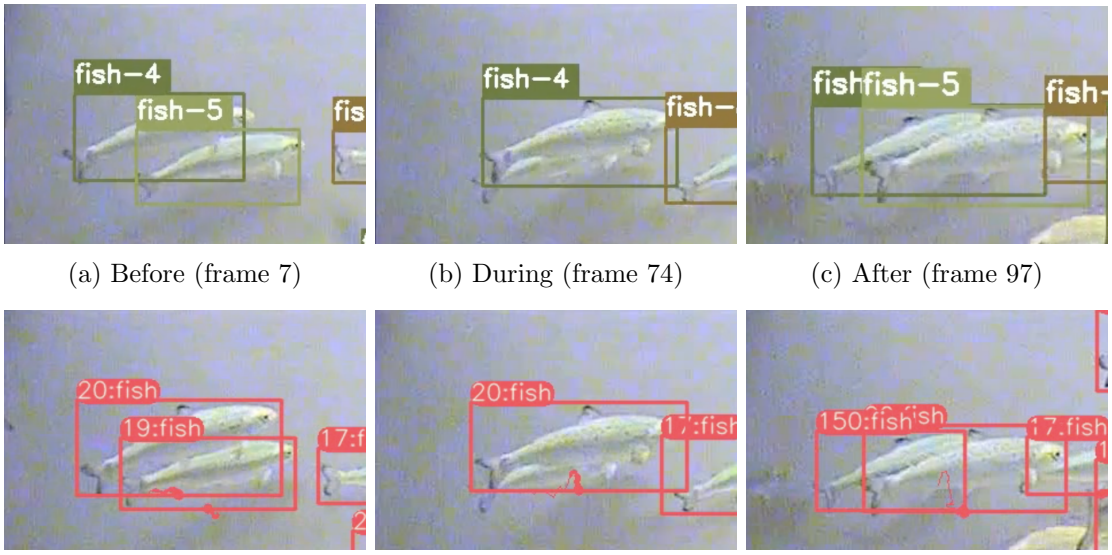
After conducting a qualitative analysis of the performance of the detection models emphasizing the differences in detection and localization, our investigation will now delve deeper into the output videos obtained from tracking with these models. By closely examining the tracked videos, we aim to gain a better understanding of the underlying patterns and occurrences that contribute to identity switches. This qualitative analysis will provide valuable insights into the dynamics of the tracking process and shed light on the specific situations and contexts in which identity switches occur.

4.2.2 Qualitative Analysis of Tracking Performance

The quantitative analysis and qualitative comparison of the detection algorithms revealed that YOLOv8 achieved better performance in detecting fish in terms of object detection accuracy and bounding box localization. Moreover, DeepSORT utilizing YOLOv8 led to significantly fewer identity switches compared to DeepSORT with YOLOv4, although the number of identity switches remained high. In this section, we shift our focus to investigate the underlying factors and circumstances that contribute to identity switches in both tracking models, starting with occlusion handling.

Occlusion Handling in Tracking

The ability to handle occlusions is a critical aspect of the tracking process, especially in the densely populated environment depicted in the datasets used for this study. Occlusions pose a challenge to tracking algorithms as they need to accurately maintain track continuity and associate detections during occlusion events. In such complex scenarios, the ability to handle occlusions effectively becomes essential in achieving reliable and accurate tracking results. Thus, by analyzing the performance of the models before, during, and after occlusion, we can gain qualitative insights into how well they handle occlusions.



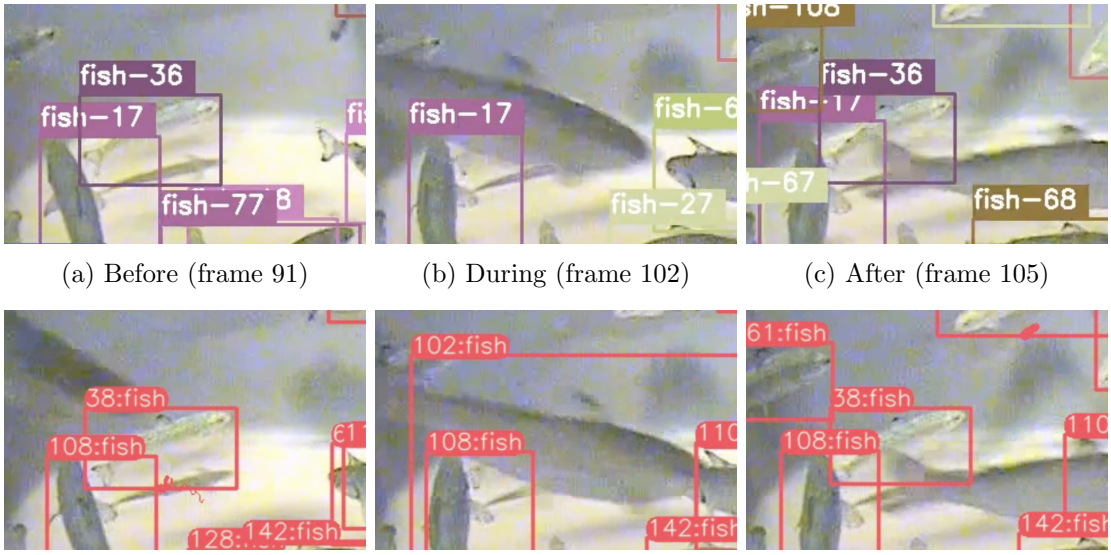
(a) Before (frame 7) (b) During (frame 74) (c) After (frame 97)

(d) Before (frame 7) (e) During (frame 74) (f) After (frame 97)

Figure 4.4: Figure (a-c) depicts instances where occlusion handling in tracking with YOLOv4 failed, resulting in inaccurate track associations. In Figure (d-f), similar challenges are observed in occlusion handling with YOLOv8.

Figure 4.4 illustrates the similarity in occlusion handling between the two tracking models, DeepSORT integrated with YOLOv4 (Figure 4.4a - 4.4c) and YOLOv8 (Figure 4.4d - 4.4f). In both cases, the models fail to accurately maintain the identities of occluded fish. Specifically, in the case of YOLOv4the identities of two fish, namely fish with identity 4 and fish with identity 5, were switched with each other. Similarly, YOLOv8 assigns a completely new identity to the fish after occlusion, resulting in identity switches. This occurrence represents a typical object-object occlusion scenario, which contributes to the high number of identity switches observed in the quantitative analysis presented in Section 4.1.2. The occlusion in question persisted for a duration of more than 40 frames, which corresponds to nearly two seconds of the video. This particular occlusion serves as an example of the limitations associated with reducing the "MaxAge" parameter, as discussed in Section 4.2.2. However, it is noteworthy that the failure to retain the identity of the occluded object was observed across all three tested "MaxAge" parameter values, including the default value of 60 frames, as well as 30 and 90 frames. This finding suggests that the tracking algorithm encounters difficulties in effectively managing such situations, and adjusting the hyperparameters does not appear to have any impact on the outcome.

In contrast to the prolonged occlusion situation presented in Figure 4.4, the subsequent scenario involves a fish being fully occluded for a shorter duration of only four frames. The tracking results obtained using YOLOv4 (depicted in Figure 4.5a - 4.5c) and YOLOv8 (Figure 4.5d - 4.5f) both accurately re-identified the occluded fish. This scenario indicates that both tracking models are capable of handling swift occlusions occurring over a few frames.



(a) Before (frame 91) (b) During (frame 102) (c) After (frame 105)

(d) Before (frame 96) (e) During (frame 102) (f) After (frame 105)

Figure 4.5: In (a-c) the tracking with YOLOv4 demonstrates the correct re-assignment of the occluded fish to the identity 36. Likewise, in (d-f), the tracking with YOLOv8 successfully maintains the identity 38 for the fish after it undergoes occlusion.

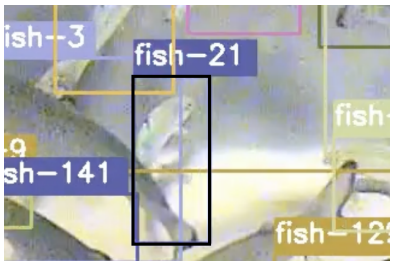
The difference in performance between swift and prolonged occlusions can be attributed to several factors. Firstly, the duration of the occlusion itself plays a crucial role. Shorter occlusions tend to preserve the visual appearance and motion characteristics of the occluded fish to a greater extent. This preservation allows the tracking models, particularly DeepSORT which relies on appearance-based features, to leverage the available visual cues and successfully re-identify the fish after the occlusion. Additionally, the motion behavior of the occluded fish may also influence the models' performance. In scenarios where the occluded fish remains in the same spatial location before and after occlusion, the models can utilize consistent spatial cues for re-identification. Conversely, a change in motion or velocity adds complexity and may hinder the models' ability to accurately track and re-identify occluded objects.

Hence, it can be inferred that the tracking models' proficiency in handling occlusions and accurately re-identifying objects is influenced by several factors, including the duration of occlusion, the preservation of visual features, and the motion of the occluded fish. Swift occlusions can be managed more effectively since they result in a shorter interruption of visual and motion cues, thereby allowing the models to maintain a more precise representation of the tracked objects.

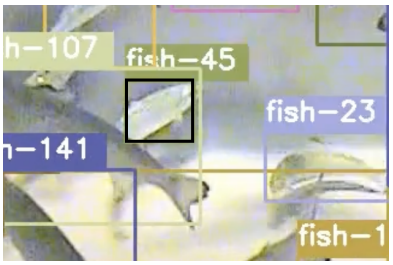
Handling Identity Switching during Tracking

The occurrence of identity switches in the tracking models cannot be solely attributed to occlusion. Figure 4.6 illustrates the performance of DeepSORT integrated with YOLOv8 (Figure 4.6d - 4.6f) and with YOLOv4 (Figure 4.6a - 4.6c) in maintaining the identity of an unobscured fish. The results show that tracking with YOLOv8 consistently preserves the identity of the fish (30), while the tracking model using YOLOv4 exhibits a higher frequency of identity switches. Initially assigned with identity 21, the fish tracked with YOLOv4 switches to identity 45 and subsequently changes to identity 3. As shown in Figure 4.6a, the identity label "3" was previously assigned to a different fish. The results of the quantitative analysis corroborate this observation, as the combination of YOLOv4 and DeepSORT exhibited a higher count of identity switches and a lower IDF1 score compared to the YOLOv8 tracker. The higher rate

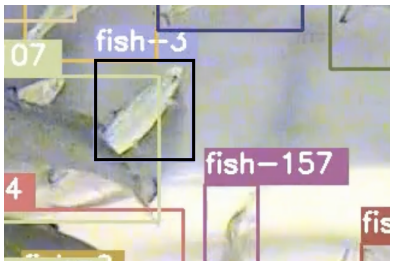
of identity switches underscores the difficulties in preserving stable identities throughout the tracking process.



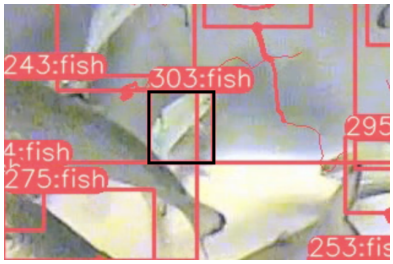
(a) DeepSORT with YOLOv4 (frame 201)



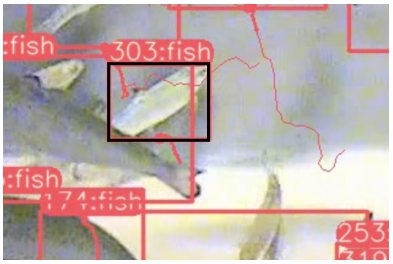
(b) DeepSORT with YOLOv4 (frame 207)



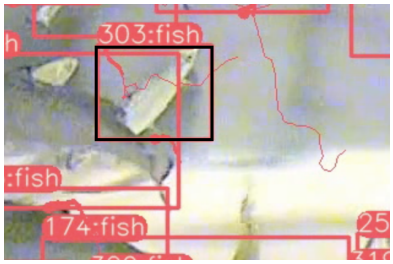
(c) DeepSORT with YOLOv4 (frame 222)



(d) DeepSORT with YOLOv8 (frame 201)



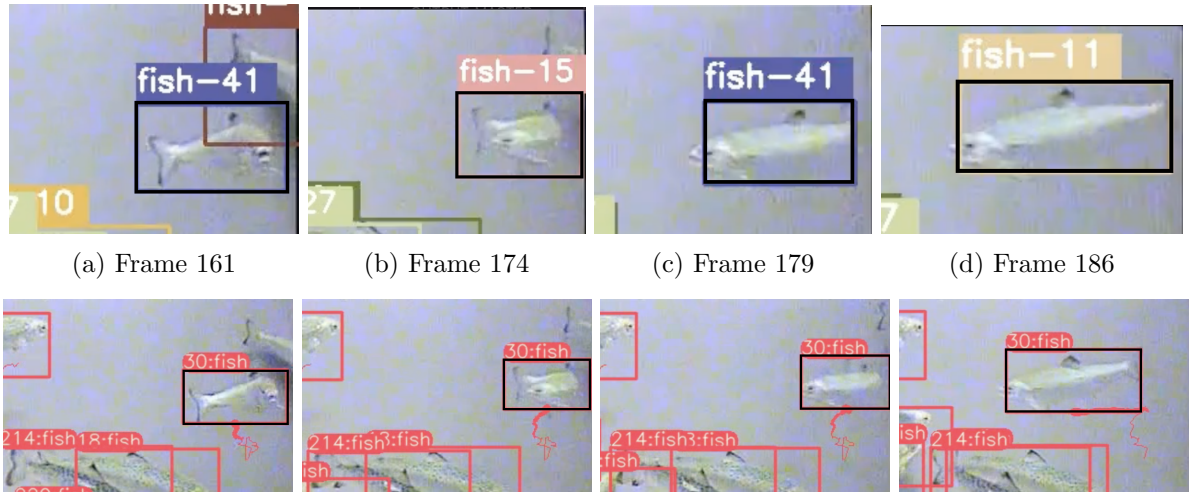
(e) DeepSORT with YOLOv8 (frame 207)



(f) DeepSORT with YOLOv8 (frame 222)

Figure 4.6: (a-c) highlights the multiple identity switches experienced by YOLOv4, indicating a less consistent performance in maintaining object identities, while Figure (d-f) illustrates YOLOv8's successful preservation of the fish's identity (identity 303) during the occlusion period

Furthermore, Figure 4.7 depict a comparison of identity retention between tracking with YOLOv4 and YOLOv8 in the same scenario in the Channel 3 test data. For tracking with YOLOv4, the fish in the scene is initially assigned identity 41 but undergoes identity switches to 15 and then back to 41, ending with identity 11. In contrast, DeepSORT with YOLOv8 successfully maintains the fish's identity throughout the sequence. Also, as stated previously, the figure clearly demonstrates that YOLOv8 exhibits a narrower aspect ratio on the bounding box compared to YOLOv4. This narrower aspect ratio indicates that YOLOv8 is more effective at precisely localizing the fish in the scene. By accurately capturing the spatial extent of the fish, YOLOv8 enhances the tracking performance by providing a better representation of the object's location

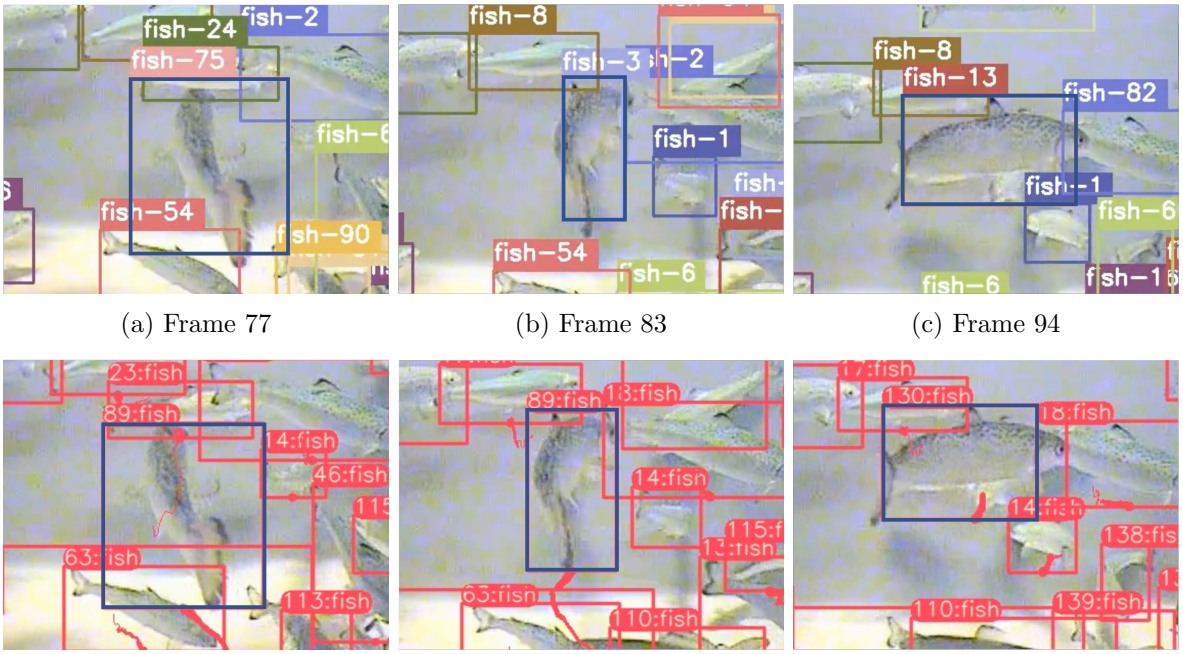


(a) Frame 161 (b) Frame 174 (c) Frame 179 (d) Frame 186

(e) Frame 161 (f) 174 (g) Frame 179 (h) Frame 186

Figure 4.7: Figure (a-c) illustrates several identity switches for the fish with an initial identity of 41 when using DeepSORT with YOLOv4. In contrast, Figure (d-f) shows that the fish successfully retains its identity as 30 when DeepSORT is combined with YOLOv8.

Despite YOLOv8 generally outperforming YOLOv4 in retaining identity for unobscured fish, it should be noted that it still faced occasional challenges as seen in Figure 4.8. The tracking results with YOLOv8 in Figure 4.8d - 4.8f illustrate a tracked fish with identity 89 undergoing an identity switch to identity 130 during the trajectory change. Similarly, YOLOv4 exhibits the same issue in Figure 4.8a - 4.8c, resulting in a total of three identity switches, including one prior to the change in trajectory. These results indicate that while YOLOv8 may encounter occasional difficulties in maintaining consistent identities during tracking, YOLOv4 exhibits greater challenges in achieving consistent and reliable tracking performance. The difficulties observed in maintaining consistent identities across both tracking models could potentially stem from a lack of sufficient or high-quality training data. The models may not have effectively learned to handle such scenarios, resulting in object identity switches, especially when a fish undergoes unexpected changes in direction or movement.



(a) Frame 77 (b) Frame 83 (c) Frame 94

(d) Frame 77 (e) Frame 83 (f) Frame 94

Figure 4.8: Figures (a-c) depict three instances of identity switches for the fish initially assigned identity 75 during tracking with DeepSORT and YOLOv4. Figures (d-f) illustrate a shift in identity during a trajectory change for fish 89, using DeepSORT integrated with YOLOv8.

Chapter 5

Discussion

The first objective of the thesis, as outlined in Chapter 1, was to achieve precise and accurate detection of individual fish in densely populated environments. The results presented in this thesis demonstrate that both YOLOv8 and YOLOv4 perform effectively as standalone object detectors for fish detection in such environments. However, when considering detection accuracy with higher IOU thresholds (mAP50-95), YOLOv8 achieves significantly better results than YOLOv4. This finding aligns with the qualitative analysis, which indicates that YOLOv8 is more adept at precisely localizing the positions of detected fish. Nonetheless, both algorithms have difficulties detecting all the fish in the frames and based on the qualitative analysis this could potentially be attributed to the low quality of the data used in this thesis or the quantity of the training data.

The performance difference in detection between YOLOv4 and YOLOv8 can be attributed to several other key factors. Firstly, YOLOv8 utilizes anchor-free detection with decoupled heads, which might improve localization accuracy and enhances the precision of bounding box predictions. By removing the reliance on predefined anchor boxes, YOLOv8 can adapt more effectively to objects of varying sizes and aspect ratios, resulting in more precise and accurate detections. Additionally, YOLOv8 incorporates a modified backbone architecture that enhances its ability to capture discriminative features. These differences in architecture can contribute to enhancing the model's understanding of visual features, leading to improved performance in object detection.

The second objective of the thesis was to track fish over an extended period. The findings indicate that both tracking models faced challenges in maintaining identity during prolonged occlusion, rapid movement, and variations in swimming patterns. As a result, the tracking of fish was inconsistent, and it can be stated that the objective of achieving tracking over an extended period was not fully realized. These challenges highlight the complexity of tracking fish over extended periods and the need for further research and development to overcome these limitations and enhance the tracking capabilities in dynamic aquatic environments.

DeepSORT, as an appearance-based tracker, heavily relies on the visual characteristics of the tracked objects for re-identification. This characteristic could potentially explain the encountered challenges in maintaining identity during occlusion. The shape of the bounding box can change significantly depending on the direction of the fish's movement in relation to the camera. Fish that swim across horizontally and vertically, or towards and away from the camera, will have extremely different appearances and bounding box shapes, making it challenging for DeepSORT to retain identity during occlusion. In contrast, a human walking behind another human is likely to have a similar appearance and the aspect ratio of the bounding box before

and after occlusion. As a result, maintaining identities during occlusion becomes comparatively easier in such scenarios.

A similar explanation can be applied to the challenges faced in retaining identity during rapid trajectory changes. As observed, when fish undergo swift trajectory changes, often spanning just a few frames, it becomes increasingly difficult for the tracking algorithm to accurately maintain their identities. These rapid switches result in significant transformations in the fish's shape and appearance, further complicating the task of identity retention for DeepSORT.

Despite the challenges faced by the tracking algorithms in handling occlusions and abrupt changes in object trajectory, the study revealed a notable difference between the two tracking models in their ability to maintain tracks for unobscured objects. Tracking with YOLOv8 exhibited a higher degree of success in preserving the identity of unobscured fish, leading to a lower number of identity switches compared to utilizing YOLOv4. A reason for this could be that YOLOv8 predicts bounding boxes with a more conservative aspect ratio, which fits the detected objects better. This better localization provided by YOLOv8 results in less variation in aspect ratio between frames as the object moves, making the tracking process easier. Contrarily, YOLOv4's bounding box prediction may not always be as precise, which can cause rapid alterations in the bounding box's shape and size during tracking. This inconsistency can challenge the tracker's ability to preserve the identity of the tracked object, leading to a higher frequency of identity switches compared to YOLOv8. The ability to maintain consistent identities for unobstructed objects is a critical factor in achieving accurate and dependable tracking results.

The results indicate that the combination of DeepSORT utilizing YOLOv8 presents the highest overall performance in tracking fish over an extended period within our test sets. This combination ensures precise bounding box localization and decent tracking of unobscured fish. However, this combination still experienced a significant number of identity switches. Hence, while this combination of models shows potential, it does not offer a sufficiently high level of accuracy for consistently tracking fish under challenging circumstances. Even though detector tuning was not performed as part of this study, the notable differences observed between the detectors during both detection and tracking suggest that further tuning could enhance the overall tracking performance. Nevertheless, considering the high mAP50-95 demonstrated by YOLOv8, it seems the detectors are not the primary cause for the identity switches during tracking.

A concurrent master's thesis also utilizes the same dataset and incorporates the ByteTrack algorithm [16] in combination with YOLOv8, as described in Chapter 1. The version of the detector employed in that thesis (Ultralytics YOLOv8.0.20) is identical to the one used in this thesis, and the same default hyperparameters have been applied. A key point of divergence between DeepSORT and ByteTrack lies in their treatment of bounding boxes. DeepSORT applies a filtration process that disregards bounding boxes with low detection confidence scores. In contrast, ByteTrack utilizes a different approach where it considers all detection boxes for the association, irrespective of their confidence scores, instead of exclusively prioritizing those with high scores.

Table 5.1: The table illustrates the performance differences between ByteTrack and DeepSORT utilize the same YOLOv8 object detector.

Tracker	Dataset	MOTA	MOTP	IDF1	IDSW	MT	PT	ML
ByteTrack	Channel 6	**0.652**	**0.158**	**0.624**	**73**	**33**	**24**	5
	Channel 3	**0.657**	**0.168**	**0.597**	**93**	**25**	**30**	6
DeepSort	Channel 6	0.556	0.185	0.466	213	30	24	5
	Channel 3	0.598	0.192	0.455	188	25	30	6

Table 5.1 illustrates the better performance of ByteTrack compared to DeepSORT when both are paired with the same YOLOv8 object detection algorithm. With default parameters, ByteTrack achieves a notably higher IDF1 score and a significantly lower number of identity switches. Given that both tracking algorithms employ the same object detection algorithm, the higher performance of the ByteTrack model is largely attributed to its unique method of handling detections, which differentiates it from DeepSORT.

ByteTrack's ability to consider all detection boxes during its association process enables it to recover objects that could be occluded or have low detection confidence scores. In contrast, DeepSORT filters out such detections, potentially treating them as background rather than true objects. As a result of its approach, ByteTrack exhibits significantly higher accuracy and more consistent tracking performance than DeepSORT.

DeepSORT's approach of filtering out detection boxes with low confidence scores can lead to the exclusion of potentially valid object detections, resulting in fragmented trajectories and reduced tracking accuracy, especially in scenarios involving occlusions or partial object visibility. To enhance the accuracy and reliability of fish tracking in dynamic aquatic environments, further research and development are needed to overcome these limitations. It is worth noting that the findings of this study deviate from the claims made in the published paper on DeepSORT, which suggests its ability to track objects through extended periods of occlusions. The observed high number of identity switches during occlusions highlights the necessity for advancements to achieve reliable tracking performance under such conditions. Overall, these findings emphasize the need for further research and development to address the challenges of fish tracking and improve the accuracy and reliability of computer vision-based monitoring systems in aquaculture settings.

5.1 Future Work

This thesis has identified several challenges that need to be addressed to enhance the performance of tracking fish for an extended period in dense environments. These challenges were identified based on the results and discussions presented in the previous chapters.

As part of future work, exploring the integration of a tracker that leverages the strengths of both ByteTrack and DeepSORT is a promising avenue. By combining ByteTrack's ability to utilize low-confidence detection boxes and DeepSORT's robustness in handling motion and appearance, a hybrid tracker might offer improved accuracy, identity retention, and consistency in diverse and challenging scenarios. Further research and development in this direction could pave the way for more effective and efficient multi-object tracking techniques in various applications.

Additionally, transformers such as TrackFormer [68] could also be a promising area for further research, as it adopts a distinct paradigm, known as tracking-by-attention. This technique

utilizes attention mechanisms for data association and performs detection and tracking jointly [68]. TrackFormer, in contrast to DeepSORT, captures the dependencies between objects and frame-level features without the need for explicit modeling of motion and appearance. This is achieved through the utilization of attention mechanisms, which allow the model to dynamically weigh the relevance of different objects and features during the tracking process. [68].

Moreover, in order to achieve accurate object-tracking outcomes, it is crucial to carefully select an appropriate object detection algorithm. Future research could explore the use of alternative detectors to further reduce the occurrence of false negatives and increase localization precision. As computer vision technology continues to advance, object detection algorithms could be designed to handle both normal and occluded bounding boxes, thereby improving the accuracy and effectiveness of object tracking algorithms. The utilization of such detectors could enhance the tracking algorithm's ability to handle re-identification effectively, especially in difficult scenarios such as prolonged occlusions.

The challenges in accurately detecting and tracking fish in this study are not solely due to the detection and tracking algorithms used. The dataset utilized in this thesis was not originally designed for computer vision tasks, so future work should prioritize acquiring higher-quality data with enhanced visual clarity. This can be achieved by employing a camera that has a higher frame rate to capture more frames per second, along with a higher resolution to enhance the overall video quality. In addition, using multiple cameras in the fish tank can help to reduce occlusion challenges by providing multiple views of the same scene, allowing for more complete tracking of fish movement. This approach can enhance the accuracy of the tracking algorithm by providing more comprehensive coverage of the environment, thereby enabling the identification of fish from multiple angles. Alternatively, using a 360-degree camera could be a viable option for capturing a complete view of the tank, but would not provide any additional help in occlusion handling.

Furthermore, the development of a new metric to supplement the existing tracking metrics for identifying occlusion handling would be beneficial. A suitable metric for evaluating object tracking performance during occlusion scenarios could consider factors such as the number of successfully tracked objects during occlusion scenarios, the duration of occlusions, and the accuracy of predicted bounding boxes during occlusion scenarios. The creation of a dedicated metric would be valuable in assessing and comparing the performance of various tracking models when it comes to handling occlusions. This metric would provide insights into how well the models can handle occluded objects and help identify the strengths and weaknesses of each approach.

A consistent tracking model with minimal identity switches has the potential to significantly impact the aquaculture industry in assessing fish welfare. Such a computer vision system could provide valuable insights into appetite, behavior, growth, and disease progression over the lifespan of the fish. The traditional manual labor currently employed in monitoring fish tanks, as described in Chapter 1, could be replaced by an automated system, which would likely be less susceptible to inaccuracies due to enhanced visibility and more time- and cost-efficient. This advantage could be further amplified if multiple cameras are utilized in tandem to track fish movements and minimize occlusion scenarios, thereby preserving the identity of each fish throughout its life cycle.

The successful implementation of a robust computer vision system for monitoring fish has the potential to extend beyond fish tanks and be applied in various environments, including sea-based fish farming. By adapting the system to address challenges related to water conditions, lighting variations, and environmental factors, it can become a versatile tool for monitoring and

managing fish populations in diverse settings. The computer vision system initially developed for tracking salmon can be further customized and expanded to monitor other fish species. By fine-tuning the detection and tracking algorithms to accommodate variations in appearance, behavior, and movement patterns across different species, the automated monitoring system can offer broad applicability within the aquaculture industry.

In addition to its primary tracking capabilities, the computer vision system can be trained to recognize specific events such as attacks, flights, and bites, providing valuable insights into fish behavior and interactions. By detecting and analyzing these events, aquaculture operators can gain a deeper understanding of social dynamics, aggression patterns, and feeding behavior within fish populations. This advancement holds significant implications for the aquaculture industry, enabling researchers to gain valuable insights into fish behavior and make informed decisions regarding fish welfare, feeding patterns, and environmental conditions.

Chapter 6

Conclusion

The two main objectives of this thesis were focused on achieving the following:

1. Detect individual fish in a dense aquatic environment with high precision and accuracy utilizing computer vision.
2. Track individual fish over an extended period based on computer vision in the same dense environment.

The findings of this study show that both YOLOv4 and YOLOv8 object detection models can detect and localize fish in densely populated environments. Among the two, YOLOv8 outperformed YOLOv4, achieving 88.4% mAP50, 64.4% mAP50-95, 0.87 precision, and 0.83 recall on the validation set, as indicated in Table 4.1. For tracking purposes, the combination of DeepSORT and YOLOv8 demonstrated superior overall performance compared to the pairing of DeepSORT and YOLOv4, as indicated by higher MOTA and IDF1 scores and a lower MOTP score for both test sets. Specifically, for the Channel 3 test set, the tracker achieved a MOTA score of 0.598, MOTP score of 0.192, and IDSW score of 0.455, as shown in Table 4.2. However, the tracking models faced difficulties in handling longer-duration occlusions and rapid trajectory changes, which led to a significant number of identity switches.

Based on the results, our findings indicate that YOLOv8 demonstrates accurate fish detection capabilities in high-density environments. However, when combined with DeepSORT for tracking, the overall performance was not consistently reliable due to the challenges posed by occlusion scenarios and rapid changes in fish trajectory, resulting in a high number of identity switches. Therefore, to ultimately achieve the goal of utilizing computer vision in fish welfare monitoring, it is essential to continue conducting further research and development to address the challenges and enhance fish tracking performance in dense environments. Overcoming the limitations observed in this study will contribute to the advancement of fish monitoring techniques and support the goal of promoting fish welfare in aquaculture practices.

Bibliography

[1] Francesca Conte et al. "Consumers' Attitude Towards Fish Meat". In: *Italian Journal of Food Safety* (2014). URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5076726/.

[2] Øyvind Aas-Hansen et al. "Ny teknologi for overvåking av oppdrettsmiljø og fiskevelferd i oppdrettsmerder. Sluttrapport for FHF prosjekt 900085". In: *Nofima* (2010). URL: https://nofima.no/publikasjon/1170717/.

[3] Isaac Ankamah-Yeboah et al. "The Impact of Animal Welfare and Environmental Information on the Choice of Organic Fish: An Empirical Investigation of German Trout Consumers". In: (2019). ISSN: 0738-1360, 2334-5985. DOI: 10.1086/705235. URL: https://www.journals.uchicago.edu/doi/10.1086/705235.

[4] Alicia Bárcena et al. "The 2030 Agenda and the Sustainable Development Goals". In: *United Nations publication* (2018). URL: https://www.vetinst.no/rapporter-og-publikasjoner/rapporter/2022/fiskehelserapporten-2021.

[5] Helmut Segner et al. "Welfare of fishes in aquaculture". In: *FAO Fisheries and Aquaculture Circular* 1189 (2019). ISSN: 2070-6065. URL: https://www.fao.org/3/ca5621en/ca5621en.pdf.

[6] Kristine Gismervik et al. "Fiskehelserapporten 2021". In: *Veterinærinstituttet* (2022), pp. 35–60. ISSN: 1890-3290. URL: https://www.vetinst.no/rapporter-og-publikasjoner/rapporter/2022/fiskehelserapporten-2021.

[7] Christopher Noble et al. "Welfare Indicators for farmed Atlantic salmon: tools for assessing fish welfare". In: *Nofima* (2018). URL: https://nofima.no/wp-content/uploads/2021/05/FISHWELL-Welfare-indicators-for-farmed-Atlantic-salmon-November-2018.pdf.

[8] Ling Yang et al. "Computer vision models in intelligent aquaculture with emphasis on fish detection and behavior analysis: a review". In: *Archives of Computational Methods in Engineering* 28 (2021), pp. 2785–2816.

[9] Ahmad Salman et al. "Fish species classification in unconstrained underwater environments based on deep learning". In: *Limnology and Oceanography: Methods* 14.9 (2016), pp. 570–585.

[10] Geoffrey French et al. "Convolutional Neural Networks for Counting Fish in Fisheries Surveillance Video". In: *Proceedings of the Machine Vision of Animals and their Behaviour (MVAB)*. Sept. 2015, pp. 7.1–7.10. ISBN: 1-901725-57-X. DOI: 10.5244/C.29.MVAB.7. URL: https://dx.doi.org/10.5244/C.29.MVAB.7.

[11] Liang Liu et al. "Counting fish in sonar images". In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE. 2018, pp. 3189–3193.

[12] Cigdem Beyan, Vasiliki-Maria Katsageorgiou, and Robert B Fisher. "Extracting statistically significant behaviour from fish tracking data with and without large dataset cleaning". In: *IET Computer Vision* 12.2 (2018), pp. 162–170.

[13] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "Yolov4: Optimal speed and accuracy of object detection". In: *arXiv preprint arXiv:2004.10934* (2020).

[14] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *YOLO by Ultralytics*. Version 8.0.0. Jan. 2023. URL: https://github.com/ultralytics/ultralytics.

[15] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. "Simple online and realtime tracking with a deep association metric". In: *2017 IEEE international conference on image processing (ICIP)*. IEEE. 2017, pp. 3645–3649.

[16] Yifu Zhang et al. "Bytetrack: Multi-object tracking by associating every detection box". In: *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXII*. Springer. 2022, pp. 1–21.

[17] Sebastian Raschka and Vahid Mirjalili. *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2*. 2019.

[18] Frank Rosenblatt. "The perceptron, a perceiving and recognizing automation". In: *Cornell Aeronautical Laboratory* (1957). URL: https://blogs.umass.edu/brain-wars/files/2016/03/rosenblatt-1957.pdf.

[19] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: 5 (1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259.

[20] AD Dongare, RR Kharde, Amit D Kachare, et al. "Introduction to artificial neural network". In: *International Journal of Engineering and Innovative Technology (IJEIT)* 2.1 (2012), pp. 189–194.

[21] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2021.

[22] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. "Activation functions in deep learning: A comprehensive survey and benchmark". In: *Neurocomputing* (2022).

[23] Mohamed Elgendy. *Deep learning for vision systems*. 2020.

[24] Peiliang Li, Xiaozhi Chen, and Shaojie Shen. "Stereo r-cnn based 3d object detection for autonomous driving". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 7644–7652.

[25] Abdul Hanan Ashraf et al. "Weapons detection for security and video surveillance using cnn and YOLO-v5s". In: *CMC-Comput. Mater. Contin* 70 (2022), pp. 2761–2775.

[26] Andre Esteva et al. "Deep learning-enabled medical computer vision". In: *npj Digital Medicine* (2021). ISSN: 2398-6352. DOI: https://doi.org/10.1038/s41746-020-00376-2.

[27] Elvis Saravia. "ML Visuals". In: *https://github.com/dair-ai/ml-visuals* (Dec. 2021).

[28] Shibani Santurkar et al. "How does batch normalization help optimization?" In: *Advances in neural information processing systems* 31 (2018).

[29] Luyang Wang et al. "Skip-connection convolutional neural network for still image crowd counting". In: *Applied intelligence* 48 (2018), pp. 3360–3371.

[30] Laith Alzubaidi et al. "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions". In: *Journal of big Data* 8 (2021), pp. 1–74.

[31] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. "A survey of transfer learning". In: *Journal of Big data* 3.1 (2016), pp. 1–40.

[32] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: (2014), pp. 580–587.

[33] Ross Girshick et al. “Region-based convolutional networks for accurate object detection and segmentation”. In: *IEEE transactions on pattern analysis and machine intelligence* 38.1 (2015), pp. 142–158.

[34] Pierre Sermanet et al. “Overfeat: Integrated recognition, localization and detection using convolutional networks”. In: *arXiv preprint arXiv:1312.6229* (2013).

[35] Jimin Yu and Wei Zhang. “Face mask wearing detection algorithm based on improved YOLO-v4”. In: *Sensors* 21.9 (2021), p. 3263.

[36] Chengji Wang et al. “Anchor free network for multi-scale face detection”. In: *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE. 2018, pp. 1554–1559.

[37] Taoshan Zhang et al. “A fully convolutional anchor-free object detector”. In: *The Visual Computer* 39.2 (2023), pp. 569–580.

[38] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. “Objects as points”. In: *arXiv preprint arXiv:1904.07850* (2019).

[39] Soharab Hossain Shaikh et al. “Moving object detection approaches, challenges and object tracking”. In: *Moving object detection using background subtraction* (2014), pp. 5–14.

[40] Chien-Yao Wang et al. “CSPNet: A new backbone that can enhance learning capability of CNN”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 2020, pp. 390–391.

[41] Joseph Redmon and Ali Farhadi. “Yolov3: An incremental improvement”. In: *arXiv preprint arXiv:1804.02767* (2018).

[42] Yi Zhu and Shawn Newsam. “Densenet for dense flow”. In: *2017 IEEE international conference on image processing (ICIP)*. IEEE. 2017, pp. 790–794.

[43] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[44] Diganta Misra. “Mish: A self regularized non-monotonic activation function”. In: *arXiv preprint arXiv:1908.08681* (2019).

[45] 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.

[46] Shu Liu et al. “Path aggregation network for instance segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8759–8768.

[47] Bing Xu et al. “Empirical evaluation of rectified activations in convolutional network”. In: *arXiv preprint arXiv:1505.00853* (2015).

[48] Alexander Neubeck and Luc Van Gool. “Efficient non-maximum suppression”. In: *18th international conference on pattern recognition (ICPR’06)*. Vol. 3. IEEE. 2006, pp. 850–855.

[49] RangeKing. *Brief summary of YOLOv8 model structure 189*. Accessed: April, 2023. 2023. URL: `https://github.com/ultralytics/ultralytics/issues/189`.

[50] Zheng Ge et al. “Yolox: Exceeding yolo series in 2021”. In: *arXiv preprint arXiv:2107.08430* (2021).

[51] Junyi Chai et al. “Deep learning in computer vision: A critical review of emerging techniques and application scenarios”. In: *Machine Learning with Applications* 6 (2021), p. 100134.

[52] Alex Bewley et al. “Simple online and realtime tracking”. In: *2016 IEEE international conference on image processing (ICIP)*. IEEE. 2016, pp. 3464–3468.

[53] Liang Zheng et al. “Mars: A video benchmark for large-scale person re-identification”. In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI 14*. Springer. 2016, pp. 868–884.

[54] James Munkres. “Algorithms for the assignment and transportation problems”. In: *SIAM* 5.1 (1957), pp. 32–38.

[55] Hamid Rezatofighi et al. “Generalized intersection over union: A metric and a loss for bounding box regression”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 658–666.

[56] Paul Henderson and Vittorio Ferrari. “End-to-end training of object class detectors for mean average precision”. In: *Computer Vision–ACCV 2016: 13th Asian Conference on Computer Vision, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers, Part V 13*. 2017, pp. 198–213.

[57] Jesse Davis and Mark Goadrich. “The relationship between Precision-Recall and ROC curves”. In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 233–240.

[58] Mark Everingham et al. “The pascal visual object classes challenge: A retrospective”. In: *Springer* 111 (2015), pp. 98–136.

[59] Jiabo He et al. “α-IoU: A Family of Power Intersection over Union Losses for Bounding Box Regression”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 20230–20242.

[60] Ohay Angah and Albert Y Chen. “Tracking multiple construction workers through deep learning and the gradient based method with re-matching based on multi-object tracking accuracy”. In: *Automation in Construction* 119 (2020), p. 103308.

[61] Anton Milan et al. “MOT16: A benchmark for multi-object tracking”. In: *arXiv preprint arXiv:1603.00831* (2016).

[62] Ergys Ristani et al. “Performance measures and a data set for multi-target, multi-camera tracking”. In: *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part II*. 2016, pp. 17–35.

[63] Boris Sekachev et al. *opencv/cvat: v1.1.0*. Version v1.1.0. Aug. 2020. DOI: 10.5281/zenodo.4009388. URL: https://doi.org/10.5281/zenodo.4009388.

[64] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/. 2013–2016.

[65] 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.

[66] Wang Hao and Song Zhili. *Improved Mosaic: Algorithms for more Complex Images*. 2020. DOI: 10.1088/1742-6596/1684/1/012094.

[67] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *YOLO by Ultralytics*. Version 8.0.0. Jan. 2023. URL: https://github.com/ultralytics/ultralytics.

[68] Tim Meinhardt et al. “Trackformer: Multi-object tracking with transformers”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 8844–8854.

Appendix A

Requirements

A.1 YOLOv4 Requirements

R-package name	Version	Purpose of use
"Python"	$\geq$ 3.9	YOLOv4 Requirement
"CUDA"	12.0	Requirement for Darknet framework for running on GPU
"cuDNN"	8.7.0	Requirement for Darknet framework for running on GPU
"OpenCV"	4.2.0	Requirement for Darknet framework
"CMake"	$\geq$ 3.18	Requirement for YOLOv4
"Darknet"		Requirement for YOLOv4 Darknet framework

Table A.1: YOLOv4 requirements to run training process.

A.2 YOLOv8 requirement

R-package name	Version	Purpose of use
"Python"	3.9	YOLOv4 Requirement
"CUDA"	12.0	Necessary for using GPU
"cuDNN"	8.7.0	Necessary for using GPU
"OpenCV"	4.2.0	YOLO requirement

Table A.2: YOLOv8-packages used during model training.

A.3 DeepSORT Requirements

DeepSORT with YOLOv4 requirements	Version	Purpose of use
"Python"	$\geq$ 3.8	Programming language used for implementing YOLOv4 and DeepSORT.
"Tensorflow"	2.3.0	Deep learning framework
"Probuf"	3.19.6	Serializing structured data.
"certifi"	2022.12.7	A collection of root certificates for secure connections
"idna"	2.10	A library for handling Internationalized Domain Names in Applications (IDNA)
"tensorboard"	2.9.1	A visualization tool for monitoring and analyzing TensorFlow runs
"requests"	2.25.1	A library for making HTTP requests
"opencv-python"		A computer vision library used for image and video processing tasks
"Pillow"		A library for image processing tasks, including image resizing, cropping, and format conversion
"Matplotlib"	3.5	A plotting library used for creating visualizations
"numpy"	1.18.5	A fundamental package for scientific computing with Python

Table A.3: Requirements for running inference with DeepSORT.

Appendix B

Default Hyperparameters for YOLOv4

Hyperparameter	Default Value	Purpose of Hyperparameter
"Batch"	64	The number of training examples used in one batch.
"Subdivision"	16	Number of mini-batches per batch.
"Max_batches"	6000	The maximum batch size that can be used during training. Set to 6000 for datasets with three or fewer classes.
"Learning Rate"	0.001	Initial learning rate
"Steps"	4800, 5400	The number of iterations at which to change the learning rate during training. The default values are typically 80% and 90% of "Max_batches".
"Width"	416	Image width resized during training and detection.
"Height"	416	Image height resized during training and detection.
"Momentum"	0.9	Controls the influence of the previous gradient direction on the current weight update during optimization.
"Saturation"	1.5	Augmentation technique. 50% increase.
"Exposure"	1.5	Augmentation technique. 50% increase.
"Hue"	0.1	Augmentation technique. scale: $0-1$.
"Mosaic"	1 (on)	Data augmentation technique: merging four images into one.
"Mixup"	1 (on)	Data augmentation technique: blends two images by multiplying and superimposing them with different ratios.

Table B.1: YOLOv4 and its default hyperparameters during training.

Appendix C

Default Hyperparameters for YOLOv8

Hyperparameter	Default Value	Purpose of Hyperparameter
"Epochs"	100	Number of times the entire training dataset is passed through the model during training.
"Patience"	50	Number of epochs with no improvement after which training will be stopped.
"Batch"	16	Number of samples per gradient update.
"IOU"	0.7	IOU threshold used for NMS to filter out redundant detections.
"size"	640	Input size of images during training and inference.
"close_mosaic"	10	Maximum distance in pixels for two images to be merged together in mosaic augmentation.
"lr0"	0.01	Initial learning rate for the optimizer.
"mosaic"	1.0	Mosaic augmentation during training.
"mixup"	0.5	Mixup augmentation during training.
"fliplr"	0.5	Probability of using flip left-right augmentation during training.
"hsv_h"	0.015	Controls the amount of hue shift applied to the image.
"hsv_s"	0.7	Controls the amount of saturation adjustment applied to the image.
"hsv_v"	0.4	Controls the amount of brightness adjustment applied to the image.

Table C.1: YOLOv8 and its default hyperparameters during training.

Appendix D

Default Hyperparameters for DeepSORT

Hyperparameter	Default Value	Purpose of Hyperparameter
"max_cosine_distance"	0.4	The maximum cosine distance between two features for them to be considered a match.
"nn_budget"	100	The maximum number of previous detections to keep for each track.
"nms_max_overlap"	1.0	The maximum allowed overlap between two bounding boxes for NMS to be applied.
"iou"	0.45	A threshold specifying the IoU determining if detection and a track should be considered the same object. (NMS IOU threshold)
"score"	0.5	A threshold specifying the confidence score.
"max_iou_distance"	0.7	The threshold value that determines how much the bounding boxes should overlap to determine the identity of the unassigned track.
"max_age"	60	Maximum number of frames a track can remain unmatched with a detection before being deleted.
"n_init"	3	Number of consecutive detections before the track is confirmed and gets assigned an id.

Table D.1: DeepSORT and its default hyperparameters.

Appendix E

Modify DeepSORT code

Add the python code to the following file and line:

1. DeepSORT utilizing YOLOv4: Add to objecttracker.py line 220.
2. DeepSORT utilixing YOLOv8: Add this code to ultralytics>yolo>v8>detect>predict.py line 242.

In object_tracker.py line 220, add:

```
file_path = os.path.join('path/to/directory/', "frame_{:06d}.txt".
    format(frame_num))
     w = bbox[2] - bbox[0]
     h = bbox[3] - bbox[1]
     x = (bbox[0] + bbox[2])/2 - w/2
     y = ((bbox[1] + bbox[3])/2) - h/2

     if os.path.isfile(file_path):
         with open(file_path, 'a') as f:  #append
             f.write("{} {} {} {} {} {} \n".format(str(frame_num), str(
                 track.track_id), x, y, w, h))
     else:
         with open('directory/frame_{:06d}.txt'.format(frame_num), 'w')
             as f:  #write
             f.write('{} {} {} {} {} {} \n'.format(str(frame_num), str(
                 track.track_id), x, y, w, h))
```

In addition, you need to create a folder where the predicted bounding box files will be stored. The path to this folder should be added to the first line of the code. The predicted files should be saved using a specific naming convention, such as ”frame000001.txt” for example. These extracted files will be compared to the ground truth files during evaluation using MOTmetrics, hence, the names have to be the same for easy comparison.

Appendix F

MOTMetrics code

The extracted frame from Appendix E are used in this code in combination with the ground truth text files. This functions takes in two folders, the DeepSORT text files and the ground truth files.

```
import motmetrics as mm
import os
import numpy as np

def motmetrics(gt_path, tracking_path, num_frames, metrics_chosen):
    """
    This is a fuction that takes in .txt files from ground truth the
       data and from the output .txt files from DeepSORT, both stored
       in two separate folders. The function further uses a distance
       matrix in the MOTMetrics libarary to map the tracking object to
       the ground truth object. It outputs different evaluation metrics
        given in 'metrics chosen'.

    Args:
        gt_path: path to ground truth folder with .txt files
        tracking_path: path to output txt files
        num_frames: number of frames to use as metric
        metrics_chosen: List of metrics selected
    """

    # Creates an accumulator object to accumulate the counts
    # It stores the evaluation results
    acc = mm.MOTAccumulator(auto_id=True)

    # Looping through each frame in both gt and tracking
    # Starting from frame 3, because DeepSORT needs 2 detections before
        starting the tracks.
    for frame_num in range(3, num_frames):
        # Load the annotation and detection data for the current frame
        # given the .txt files are named ex. frame_000003.txt
        ground_truth = os.path.join(gt_path, f'frame_{frame_num:06d}.
           txt')
        deepsort = os.path.join(tracking_path, f'frame_{frame_num:06d}.
           txt')

        # Load data from a text file into numpy array
        ground_truth_data = np.loadtxt(ground_truth)
```

```
30          deepsort_data = np.loadtxt(deepsort)
31
32          # Extract gt ids and tracking ids for this specific frame
33          ground_truth_id = ground_truth_data[ground_truth_data[:,0] ==
                0, 1] # array of gt ids
34          deepsort_id = deepsort_data[:,1] # array of tracking ids
35
36          # Save all bounding box coordinates in arrays
37          # Format: x, y, width, height of the bounding boxes
38          ground_truth_frame = ground_truth_id[ground_truth_id[:, 0] ==
                0, 2:6]
39          deepsort_frame = deepsort_data[:, 2:6]
40
41
42          #Compute 'intersection over union (IoU)' distance matrix
                between object and hypothesis rectangles.
43          # IoU=0.5: 50% overlapping between tracking BB and gt BB.
44          distances = mm.distances.iou_matrix(ground_truth_frame,
45                                              deepsort_frame,
46                                              max_iou=0.5) # inverse IoU
47
48          # Update the accumulator
49          acc.update(ground_truth_id, deepsort_id, distances)
50
51      mh = mm.metrics.create()
52      summary = mh.compute(acc, metrics=metrics_chosen, name='metrics')
53      return summary
```

Appendix G

YOLOv8 Data Augmentation

Figure G.1 and Figure G.2 shows various augmentation techniques during the training process of YOLOv8.

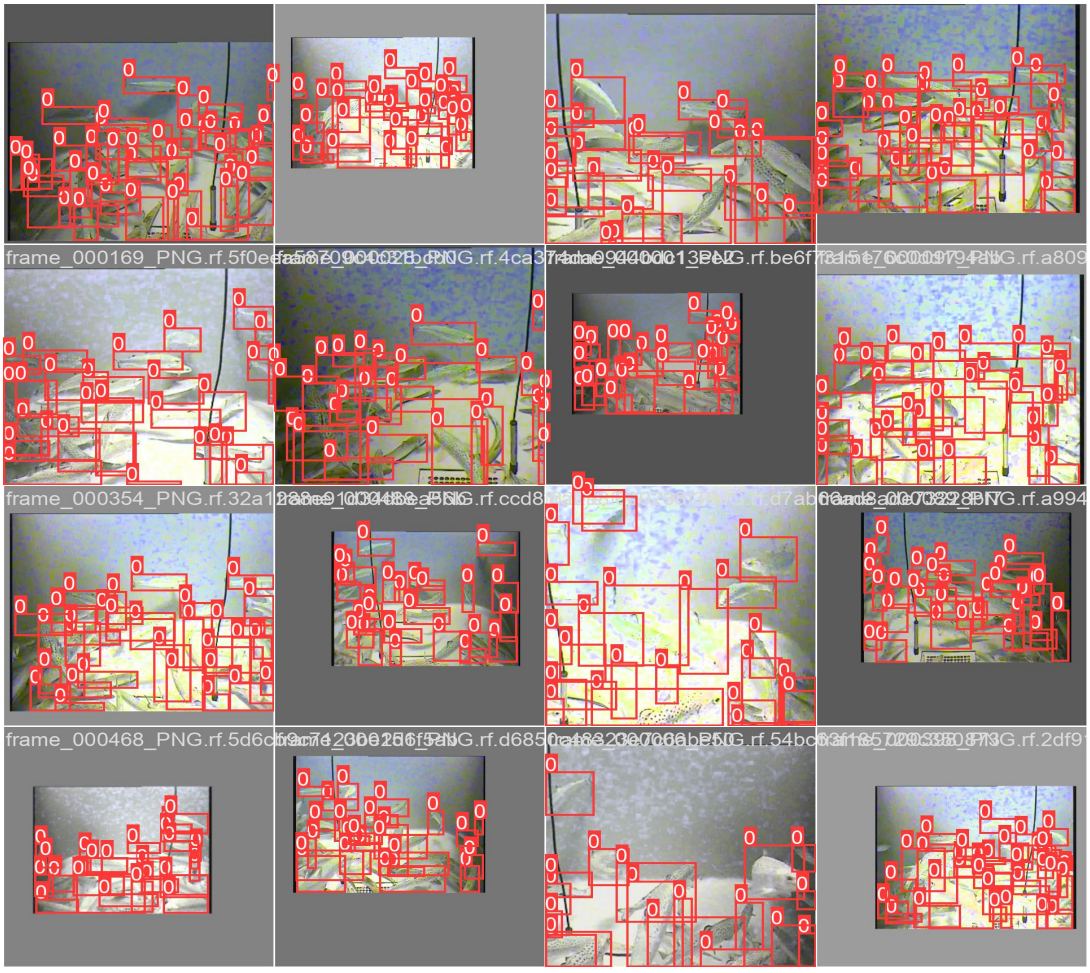


Figure G.1: Illustration of data augmentation in the training process.

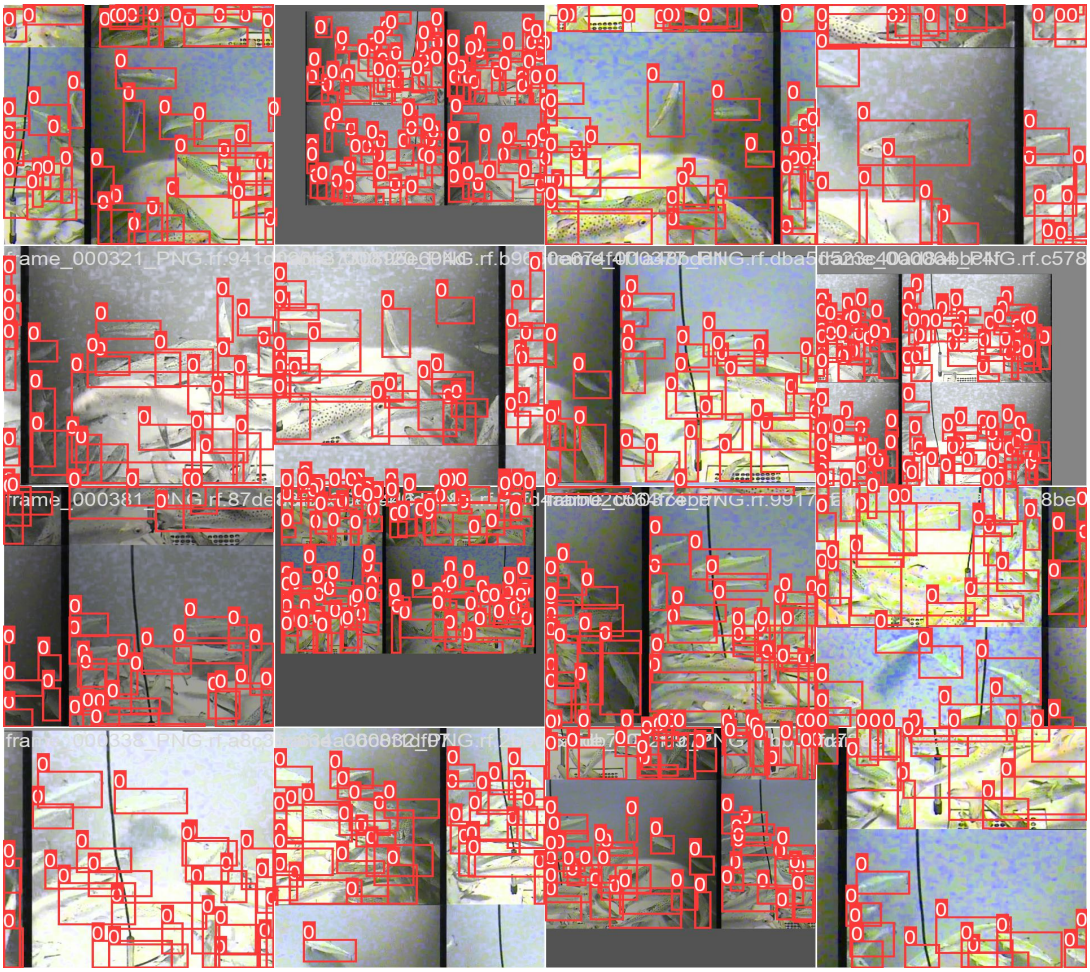


Figure G.2: Illustration of data augmentation in the training process.

Appendix H

Detections: Differences in Localization

Figure H.1 illustrates the disparity in precise localization between the YOLOv4 and YOLOv8 object detectors. Notably, YOLOv8 demonstrates accurate detection of the entire fish, exhibiting high confidence in its prediction. On the other hand, YOLOv4 fails to detect the fish's head and exhibits a lower confidence score for its detection.

Figure H.1: Ground truth vs. YOLOv4 vs YOLOv8. The bounding boxes are highlighted for more visibility.

In Figure H.2, both detectors encounter difficulties in accurately localizing the object. However, there are notable differences in their approach. YOLOv4, despite having a higher confidence score, generates a larger bounding box that encompasses a significant portion of the background. In contrast, YOLOv8, with a lower confidence score, tightly encompasses the fish but falls short in extending the bounding box to include the tail region.

Figure H.2: Ground truth vs. YOLOv4 vs YOLOv8.

Fig. H.3 demonstrates that YOLOv4 exhibits a higher confidence score, yet its bounding box localization is less accurate compared to YOLOv8, which closely resembles the ground truth.

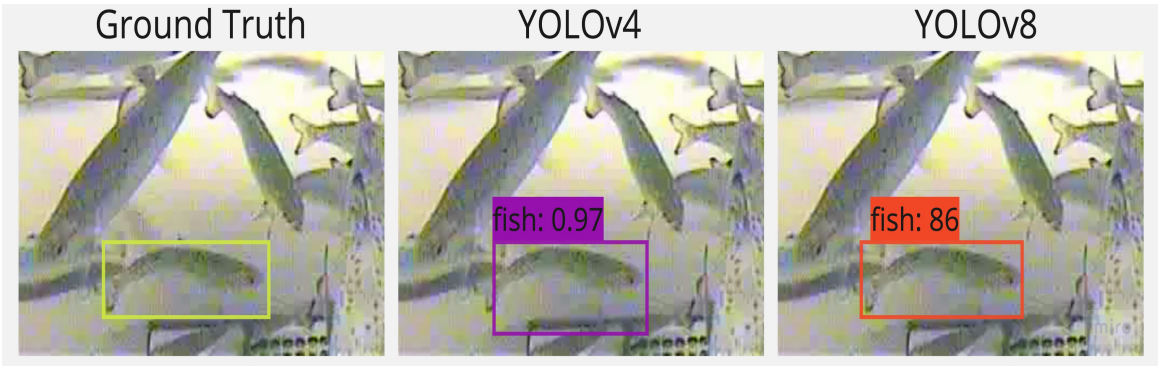


Figure H.3: Ground truth vs. YOLOv4 vs YOLOv8.

Fig. H.4 reveals that YOLOv4 has a notably high confidence score, yet it struggles to accurately fit the bounding box. On the other hand, YOLOv8 successfully replicates the ground truth in this case.

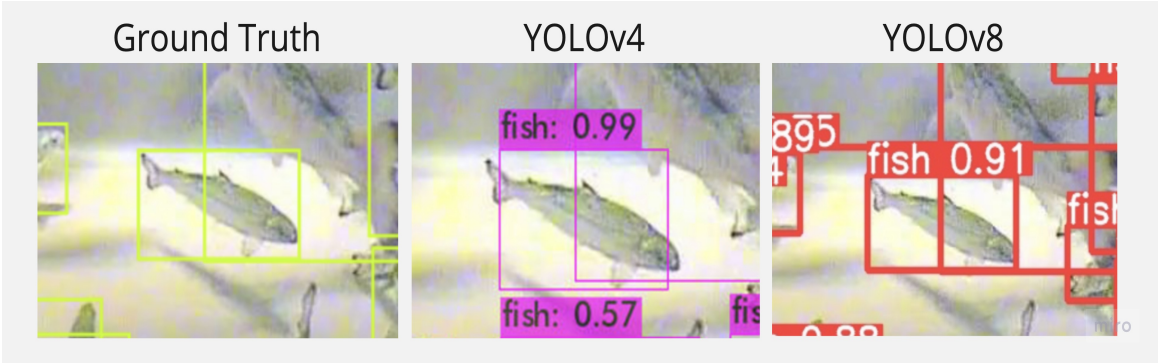


Figure H.4: Ground truth vs. YOLOv4 vs YOLOv8.

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

Postboks 5003
NO-1432 Ås
Norway