Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

Vebjørn Bjørlo-Larsen

Vision based real-time fish counting, inspection and classification using deep learning

Master's thesis in Simulation and Visualisation
Supervisor: Ibrahim A. Hameed
July 2021

NTNU
Norwegian University of Science and Technology

Vebjørn Bjørlo-Larsen

Vision based real-time fish counting, inspection and classification using deep learning

Master's thesis in Simulation and Visualisation
Supervisor: Ibrahim A. Hameed
July 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

Preface

This thesis was written during the spring of 2021 at the Norwegian University of Science and Technology (NTNU), Faculty of Information Technology and Electrical Engineering, Department of ICT and Engineering. The thesis was proposed by and done in collaboration with Stranda Prolog AS.

I would like to thank Stranda Prolog AS for a very interesting and challenging problem for the thesis, and for assisting with gathering of video material. I would also like to thank my supervisor for help during the thesis, and the university for assisting with a powerful computer for training and testing.

Supervisor: Ibrahim A. Hameed

Contact person at Stranda Prolog AS: Kjetil Osland Brekken

Vebjørn Bjørlo-Larsen

Ålesund, July 5, 2021

Abstract

Current fish counters rely on feeding fish through dedicated equipment as a part of the fish transportation. This thesis proposes a vision-based alternative, using cameras mounted above conveyor belts to count, inspect, and classify fish. The proposed solution is based on a multiple object tracking algorithm, using deep learning to detect and track fish from frame to frame in a video. Tracking of fish through a video ensures that each fish is only counted once, and it also enables fish inspection and classification. Thus, in addition to fish counting, this thesis also investigates damage detection approaches and methods for classifying fish as dead or alive. The experiments conducted show that the developed solution performs better in fish counting than existing fish counters, accurately counting above 98% of fish, with a total score of above 99% when including fish counted twice. For fish inspection, the damage detection accuracy is close to 90%, up to 95% with false positives. Classification accuracy is around 70% for alive fish and 90% for dead fish, resulting in a total score of around 100%, when including the false positives from each category. The inspection results are promising, though further work is required to improve the results even more. The datasets used for training the deep learning networks, another contribution of this work, were created specifically for the project, using video footage from a conveyor belt in use.

Sammendrag

Dagens fisketellere er avhengige av å mate fisk gjennom dedikert utstyr som en del av fisketransporten. Denne oppgaven foreslår et alternativ basert på maskinsyn, som bruker videokamera montert over samlebånd for å telle, inspisere og klassifisere fisk. Den foreslåtte løsningen er basert på *mutiple object tracking* algoritmer, som ved bruk av dype kunstige nevrale nettverk detekterer og sporer fisk fra bilde til bilde i videoen. Ved å spore fisk gjennom videoen, er det mulig å kun telle hver fisk en gang, og det muliggjør også fiskeinspeksjon og klassifisering som er avhengig av sporing av fisk. I tillegg til å telle fisk, undersøker denne oppgaven også metoder for påvisning av skader og metoder for å klassifisere fisk som død eller levende.

Eksperimentene som ble utført viser at den utviklede løsningen presterer bedre i fisketelling enn eksisterende fisketellere, med en sann nøyaktighet på over 98 %, opp mot over 99 % når falske positiver er inkludert. For fiskeinspeksjon er nøyaktigheten til deteksjon av skader nær 90 %, opptil 95 % med falske positive skader. Klassifiseringsnøyaktigheten er rundt 70 % for levende fisk og 90 % for død fisk, noe som resulterer i en samlet score på rundt 100 % når falske positiver fra hver kategori er inkludert. Inspeksjonsresultatene er lovende, men det kreves ytterligere arbeid for å forbedre resultatene enda mer.

Datasettene som ble brukt til å trene maskinlæringsnetverkene er et annet bidrag fra dette arbeidet. De ble laget spesielt for prosjektet, og er laget av videomateriale fra et transportbånd i bruk.

Contents

Abbreviations

API Application Programming Interface

CNN Convolutional Neural Network

CPU Central Processing Unit

DNN Deep Neural Network

GPU Graphics Processing Unit

IoU Intersection over Union

LSTM Long Short-Term Memory

MOT Multiple object tracking

MSE Mean Squared Error

PTZ Pan-Tilt-Zoom (Camera)

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

List of Figures

List of Tables

Chapter 1

Introduction

This chapter aims to give a brief introduction to this thesis, starting with background concepts and motivational aspects associated with the project. Next, the problem outline and the goals and research questions this thesis explores are described. Finally, the research approach chosen to achieve these goals is presented, before a quick overview of the structure of the thesis.

1.1 Background and Motivation

The aquaculture industry is a growing industry both nationally in Norway [15] and on a global scale [6], with aquaculture productions covering the majority of the rise in fish consumption since the early 1990s [6]. The Norwegian Seafood Federation (Sjømat Norge) aims to double the value creation of the Norwegian seafood industry by 2030 and quintuple it by 2050, in a sustainable way, and one of the areas of focus is on the use of new technologies and automation in all parts of the industry [16].

One area where new technology can be deployed is for fish counting and inspection, and for this thesis specifically during transportation of fish in fish processing facilities. Existing solutions for fish counting use dedicated equipment installed as a step in the fish transportation setup, usually as a counting module inserted between pipes , or as a separate counting table that the fish are fed through.[1][2] This means that planning and dedicated space need to be used for

[1] AquaScan: https://www.aquascan.com/ (As of June 2021).
[2] Calitri Technology: https://www.calitri-technology.com/en/fish-counters/ (As of June 2021).

the fish counters. This is especially challenging when it comes to upgrading or adding counters to a system, as it would require changes to the layout and at least partial halts to the production as parts are changed.

The motivation behind this thesis is to explore the use of cameras mounted above conveyor belts as an alternative to existing technologies (see figure 1.1). The potential benefits of using cameras are manifold. Cameras are small and can be easily mounted to both existing and new equipment without the need for change in the layout. Because of this, they can also be used in cramped areas where larger equipment can not fit. Cameras are non-intrusive as they do not directly interfere with the fish, which eliminates risk of damaging them. Existing solutions are also made to have as little impact on the fish as possible, but they do add an additional step in the transportation, which can increase the chance of damaging fish.

Another benefit of cameras are their flexibility and versatility. They can be mounted throughout a facility without affecting the flow of fish, which allows for both monitoring and regulation of the flow of fish where desired. In addition to counting, cameras also have a huge potential to be used for inspection of fish. While the main focus of this thesis is on counting, it also touches on areas where cameras can be used for inspection, such as damage detection and classification if a fish is dead or alive.

For a camera to be competitive for counting, it needs to offer similar or better results than the existing technologies. While actual accuracy will depend on the implementation conditions, current solutions[3] [4] are rated at 97% or above accuracy in optimal conditions, which means there is a high burden of accuracy to be met.

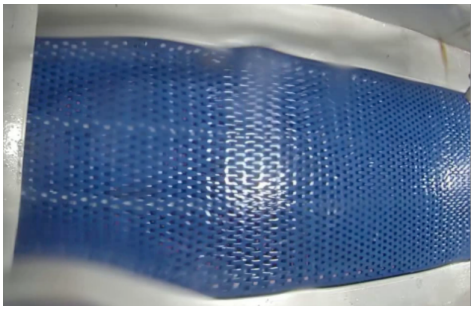


Figure 1.1: Example view from a camera mounted above a conveyor belt.

[3] AquaScan: https://www.aquascan.com/ (As of June 2021).
[4] Calitri Technology: https://www.calitri-technology.com/en/fish-counters/ (As of June 2021).

This thesis builds upon knowledge gained through a specialisation project on the same topic, with the report from the project attached in Appendix A. The specialisation project acts as background knowledge for this thesis, giving a direction for where to focus during the work. However, for the sake of proper comparisons using new video material, all relevant parts from the specialisation project have been redeveloped and new results and conclusions have been drawn, as described in the main body of this document.

1.2 Goals and Research Questions

Goal 1: *Gather video material and create datasets and testing videos to develop and test solutions proposed in this thesis.*

The main goal of this thesis is to explore methods for vision-based counting of fish on a conveyor belt, using real-time video footage. To perform fish counting with cameras, there are a few challenges and problems that need to be addressed. First of all, it is necessary to gather video material to create datasets and testing videos for the project. The availability of such data is limited, thus the first goal of this thesis is to gather the required data in collaboration with Stranda Prolog AS, and use this data to create datasets and testing videos.

Goal 2: *Develop solutions for accurate real-time fish counting from video of fish on a conveyor belt.*

The next task addressed is to perform the actual fish counting with the video material gathered, which presents its own set of challenges. Counting objects in a static image is well understood and can be done in a variety of ways, from using simple image processing methods [9], to more advanced image segmentation and deep learning methods [10]. However, simply detecting and counting objects in individual frames are not sufficient when a video is considered. Each fish will appear in many successive frames, thus a way to track each individual from frame to frame is required to perform accurate counting. This thesis will explore the use of the multiple object tracking algorithm [5] and deep learning methods for fish detection and tracking from frame to frame, and how this can be used for counting, inspection, and classification.

Research question 1: *How to use multiple object tracking algorithm to perform counting of*

fish on video?

Multiple object tracking consists of two main parts, object detection and object tracking. Object detection deals with detection of objects in each individual frame, while object tracking attempts to track each object from frame to frame [5]. There are many different ways to achieve this, thus one of the research goals is to find suitable methods for this implementation, primarily focusing on deep learning methods.

Research question 2: *Which methods, algorithms, and techniques are most suited for a real-time implementation of multiple object tracking for fish counting?*

Goal 3: *Expand the fish counting solution to include fish inspection and classification*

In addition to fish counting, this thesis will also explore fish inspection and classification based on the same system of tracking fish from frame to frame in a video. This is a very broad topic and there are a lot of different possibilities, such as estimating fish size and weight, orientation on conveyor belt, fish species, damage detection, classifying if fish are dead or alive, and so on. This thesis will focus on damage detection and classifying if fish are dead or alive, while laying the groundwork for future expansion.

Research question 3: *How to extend the same system used for fish counting to also perform fish inspection?*

Research question 4: *Specifically, how to extend the system so that it can be used for damage detection, and to classify fish as dead or alive?*

1.3 Research Approach

1. **Literature Review:** The first phase of the project was to perform a literature review of existing solutions within fish counting, as well as for other technologies, methods, and techniques that can be applied to fish counting. After an initial research on the topic, the main area of research was narrowed to be within multiple object tracking. This includes research into the best ways to perform object detection, as well as different methods for tracking objects frame to frame.

2. **Data Gathering:** The next step of the project was to gather data for development and testing. This was done in collaboration with Stranda Prolog, who set up a PTZ camera above a conveyor belt in use at a fish processing facility. Through remote control of the camera the necessary video material could be gathered, from which datasets could be created.

3. **Development and Testing:** The main part of this project consisted of development and testing of sections of the solution and finally the solution as a whole. The development was done following an agile methodology, with a focus on rapid iteration and development based on a combination of the original goals and results [1]. Throughout the project, the work was focused on achieving the goals and answering the research questions set out in Section 1.2, while also being responsive to results and basing further work on them. For example, after data gathering, it became clear that there were enough examples of damaged fish to explore damage detection, which made that a focus for exploring fish inspection. Through this process, the final solution presented in this thesis was derived.

1.4 Thesis Structure

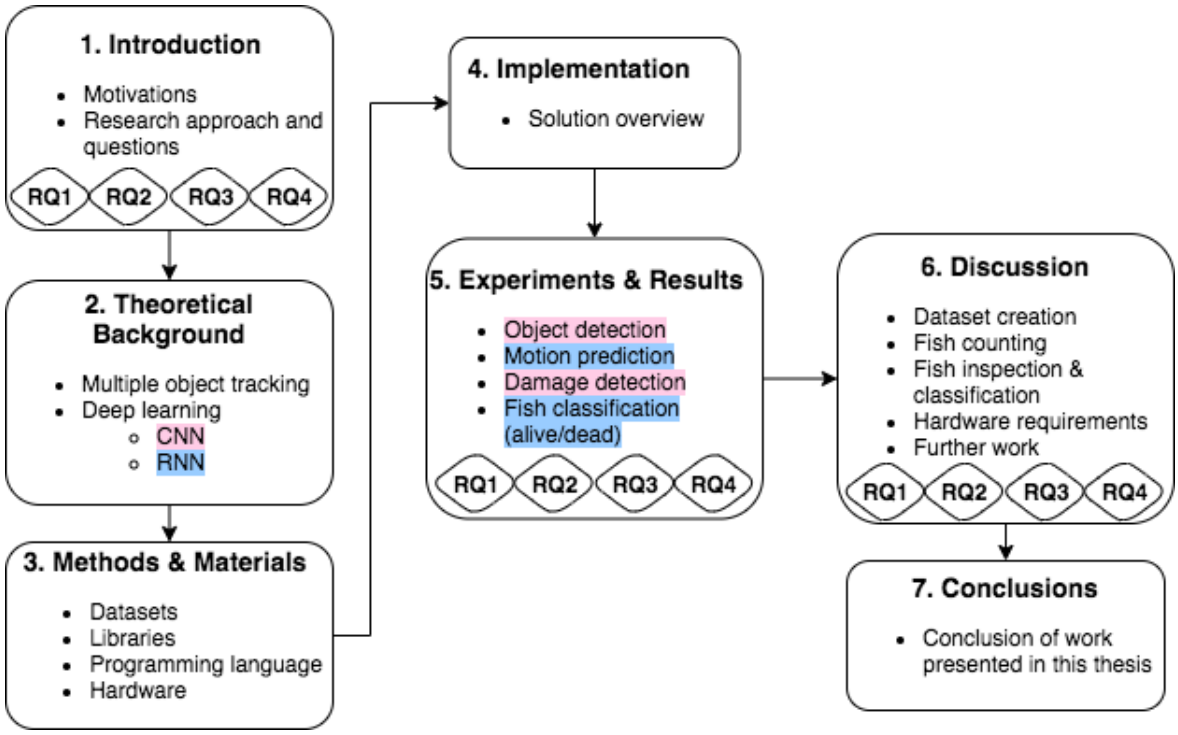


Figure 1.2: Overview diagram of thesis organization

- **Chapter 1 - Introduction** gives an introduction to the project presented in this thesis, including background and motivational aspects, goals and research questions, and the research approach chosen for the project.
- **Chapter 2 - Theoretical basis** provides an introduction to the theoretical background that forms the basis for the solutions presented in this thesis.
- **Chapter 3 - Methods** contains a description of the methodology and materials that were considered throughout the project.
- **Chapter 4 - Implementation** provides a detailed overview of the developed solution.
- **Chapter 5 - Experiments and Results** goes into details about the experiments used to validate the solution and their results
- **Chapter 6 - Discussion** presents a discussion of the results, advantages, disadvantages, and plans for further development.

- **Chapter 7 - Conclusions** presents an overall conclusion and final results of the whole thesis.

Chapter 2

Theoretical basis

This chapter contains an overview of the theory behind the methods and algorithms used in this thesis. The central focus of this thesis is on the use of multiple object tracking (MOT) and deep learning for visual fish counting and inspection, thus this chapter will first give an overview of MOT and the steps that make up the algorithm. This is followed by an overview of the specific theory behind the methods used for the specific MOT steps, such as the deep learning methods used. The following chapters assume that the reader is familiar with the theory from this chapter.

2.1 Multiple Object Tracking

Multiple object tracking (MOT) is a computer vision problem that attempts to identify and track multiple objects in a video sequence, keeping track of their positions and trajectories. Each object is tracked in a *track*, which contains a unique ID and information of the object from previous frames. The most commonly used strategy, and the one used in this thesis, is detection-based tracking (or "tracking-by-detection") [13]. This method consists of four main parts or stages for each frame of the video. First is the detection stage, where all the objects in the frame are identified. This is followed by a motion prediction or feature extraction stage, where the goal is to either predict the position of existing tracks, or to extract features such as appearance features of the objects. This is followed by an affinity / cost stage, where all the objects in the new frame are compared to all the existing tracks, and given an affinity or cost score based on a

chosen metric, such as distance or appearance similarity. Lastly, a matching algorithm is used in an association stage to match detected objects to existing tracks, and handle the birth/death of tracked objects [5].

Within detection-based tracking there are different models that can be used, such as an appearance model or a motion model. The appearance model uses the visual representation of an object to calculate similarity between objects, while the motion model uses the dynamic behaviour to estimate positions of known objects and compare them to detected objects [13]. This thesis uses the motion model, which will be described in more detail in the following sections.

2.1.1 Object Detection

The object detection stage deals with detecting and identifying objects within a frame. Usually the output from this step is a set of bounding boxes, and the corresponding types of objects if there are multiple object types (human, dog, car, etc). There are a range of different methods for object detection, but most state-of-the-art MOT algorithms use deep learning algorithms such as Faster R-CNN, SSD or Yolo [5].

2.1.2 Motion Prediction

In the motion prediction stage, the aim is to estimate the new positions of the tracked objects from the previous frames. The predictions usually take the form of the object centroids or bounding boxes, and are used in the following steps to match and assign new detections to tracked objects.

One common algorithm for this is the kalman filter [5]. The kalman filter uses a linear dynamical system to model the motion of the objects [12], which is used to estimate the centroids of all the tracked object. The kalman filter is described further in 2.2.

An alternative method for motion prediction is through the use of deep learning models, particularly recurrent neural networks (RNNs) such as the long short-term memory (LSTM) network [14]. RNN models take as input the detections (bounding boxes) of an object from the previous frames, and outputs a bounding box prediction for the next frame based on a learned prediction model. RNNs and LSTMs are described further in 2.4.3.

2.1.3 Affinity

In the affinity stage a cost or affinity matrix is created by calculating a score between each of the predicted and detected object positions. This score indicates how similar or how low the distance is between each pair of prediction and detections. The metric used to calculate the scores will depend on the specific implementation, for this thesis the metrics used are distance and IoU [5].

Distance indicates how far away the centers of two objects are, and is calculated as the euclidean distance between the two object centroids:

$$Distance = \sqrt{(X2 - X1)^2 + (Y2 - Y1)^2} \tag{2.1}$$

Where X1, Y1 and X2, Y2 are the coordinates of the object centroids.

IoU, or intersection over union, is a metric for how big the overlap between two rectangles is and is used to determine how closely two bounding boxes match each other. The cost calculation is a ratio between the overlap and the union of the two bounding boxes:

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union} \tag{2.2}$$

2.1.4 Association

In the association stage the aim is to assign or match detected objects from the current frame to the tracked objects (tracks), and if necessary create new tracks for new objects, or remove tracks from objects no longer in the frame. The assignment is done using the cost matrix from the affinity stage, with the goal of matching detection/track pairs with the lowest costs. This can be solved using assignment problem algorithms such as the hungarian algorithm, which is an efficient algorithm for minimising the total cost of all pairs. The hungarian algorithm is explained further in 2.3. To handle the birth and death of tracks, a detection is classified as a new track if it is not paired with an existing track, or if the cost of a pair is deemed too high. If

a track is not paired with a detected object for a set amount of frames, it is deemed to have left the scene, and the track is removed.

2.2 Kalman Filter

The kalman filter uses a linear dynamical system to model the motion of objects. This is an iterative method to predict the next centroid of an object and update the dynamic model based on how accurate the estimates are. The algorithm consists of two main stages, prediction and update. In the prediction stage, the predicted state estimate and predicted error covariance are calculated using formula 2.3 and 2.4, respectively [12].

Predicted state estimate

$$\hat{\mathbf{x}}_k^- = \mathbf{F}_k \hat{\mathbf{x}}_{k-1}^+ + \mathbf{B}\mathbf{u}_{k-1} \tag{2.3}$$

Predicted error covariance

$$\mathbf{P}_k^- = \mathbf{F}\mathbf{P}_{k-1}^+ \mathbf{F}^\top + \mathbf{Q} \tag{2.4}$$

Where $\mathbf{x}$ is the state vector, $\mathbf{F}$ the state transition matrix, $\mathbf{B}$ the control-input matrix, $\mathbf{u}$ the control vector, $\mathbf{P}$ the state error covariance, and $\mathbf{Q}$ the covariance of process noise. The hat operator $\hat{}$ is the estimate value, and $-$ and $+$ signifies if the estimate is the predicted or updated estimate. Superscript ^{T} denotes the transpose of the matrix [12].

During the update stage the formulas 2.5 to 2.8 are used to update the state estimate and error covariance.

Measurement residual

$$\hat{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k^- \tag{2.5}$$

Kalman gain

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}^T (\mathbf{R} + \mathbf{H}\mathbf{P}_k^- \mathbf{H}^T)^{-1} \tag{2.6}$$

Update state estimate

$$\hat{\mathbf{x}}_k^+ + \mathbf{K}_k \hat{\mathbf{y}} \tag{2.7}$$

Update error covariance

$$\mathbf{P}_k^+ = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k^- \tag{2.8}$$

Where $\mathbf{z}$ is the measurement vector, $\mathbf{H}$ the measurement matrix, $\mathbf{R}$ the covariance of observation noise, and the other variables and symbols the same as described for the prediction stage [12].

The algorithm is computationally relatively simple and requires small computational power, which means it can be used for real-time applications. Each of the tracked objects has a corresponding filter, which allows the filters to capture the dynamic model of each individual object.

2.3 Hungarian Algorithm

The hungarian algorithm is an efficient algorithm for solving the assignment problem. The assignment problem consists of finding an optimal assignment of n resources to m tasks, such that the total cost of the assignments is minimized. Each resource and task pair has a cost, which is collected in a cost matrix with the cost of all the pairs [3]. The hungarian algorithm for solving the assignment problem consists of 4 steps, using the cost matrix of the assignments [3]:

Step 1. Find the lowest cost in each row and subtract it from all elements in the row.

Step 2. Find the lowest cost in each column and subtract it from all elements in the column.

Step 3. Draw lines such that all the zeroes in the resulting matrix are covered with the minimum amount of lines. If the number of lines is equal to the highest out of number of tasks or resources, an optimal solution can be found. If not, move on to step 4.

Step 4. Find the lowest cost not covered by a line, subtract it from all costs not covered by a line, and add it to all elements covered by a line twice. Repeat step 3 until an optimal solution is found.

2.4 Deep Learning

One of the big challenges within artificial intelligence is solving problems without set rules, problems that are often trivial to humans, but that are difficult for computers to solve. Machines are great at solving well-defined problems with known rules, but can struggle with more nuanced problems without fixed rules, such as computer vision problems, timeseries forecasting, image classification, speech recognition, and so on. Machine learning is a paradigm within artificial intelligence that attempts to solve such problems. The main difference from classical programming is that instead of getting *answers* based on *rules* and *data*, machine learning attempts to learn the *rules* through *data* and *answers*. A machine learning system is trained by learning from known data, without the need to explicitly program the rules [4].

Deep learning is a subfield within machine learning, with an emphasis on learning based on successive layers of representations. The layers attempt to extract meaningful representations from the data, and through consecutive layers the aim is to learn a representative model of the problem that can be used to predict results from new data. This layered approach in deep learning almost always refer to neural networks [4].

2.4.1 Neural Networks

Neural networks are built up by layers of interconnected processing nodes, usually in the form of "feed-forward" networks where data goes through the network in one direction. Each layers receives input data from the previous layer, and transforms the input based on *trainable parameters* (weights w and biases b) within the layer and the layer *activation function*, which calculates an output based on inputs, weights and biases. The purpose of training a neural network is to "adjust" the network parameters by exposing the network to training data, such that they make a generalized model of the problem that can be used to predict results on unknown data. [4]

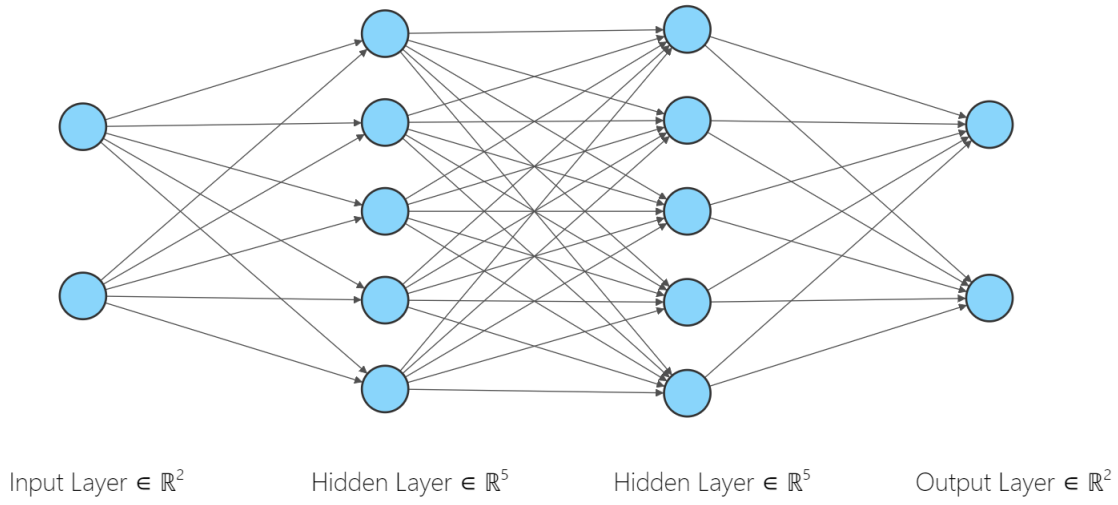


Figure 2.1: Example of a feed-forward neural network with an input layer, two hidden layers and an output layer. Figure made through http://alexlenail.me/NN-SVG/index.html

Training Overview

Training a neural network is usually done using *gradient-based optimization*. The network weights are initialized with small random values, which in itself will not lead to any meaningful representation, however it functions as a starting point. Through a training loop, the weights are then gradually adjusted based on feedback from the training. The gradient-based optimization loop works by first sending a batch of training samples through the network, generating at first random outputs. The results are then compared to the expected outputs and a *loss* is calculated using a *loss function*, which gives an estimate of how accurate the predictions are. This loss is then *backpropagated* through the network, updating the weights based on the *gradient* of the loss over the network. This process is then repeated for as long as desired [4].

Backpropagation

Backpropagation is the algorithm used in gradient-based optimization to update the weights for each step. In backpropagation the chain rule is applied to compute the gradient values of the loss across the neural network. Starting with the final loss value from the output layer, the algorithm works backwards through the hidden layers to the input layer. For each layer, the chain rule is used to calculate how big of a contribution each node has in the loss value, and

based on this each node can be adjusted accordingly [4].

Activation Functions

The activation function is used to calculate the output from a node based on the node inputs. There are many different activation functions for different applications, though perhaps the most commonly used function currently is the **ReLU** or rectified linear unit function. ReLU makes all negative outputs 0, while keeping the positive outputs unchanged.

$$\mathbf{ReLU}(\mathbf{x}) = \mathbf{max}(\mathbf{0}, \mathbf{x}) \tag{2.9}$$

ReLU is a linear activation function that is computationally simple to implement, only requiring a max() function. Additionally, linear models are easier to optimize [8]. Vanishing gradients are less of a problem in a linear model, as the gradient is proportional to the node activations [7].

Softmax is an activation function that transforms the input values into values between 0 and 1 that sum up to 1, representing a probability distribution. Softmax is often used for classification of mutually exclusive classes in the final layer of a network, such as classifying the number in an image. The softmax function is given by:

$$\mathbf{Softmax}(\vec{\mathbf{z}})_i = \frac{\mathbf{e}^{z_i}}{\sum_{\mathbf{j=1}}^{\mathbf{K}} \mathbf{e}^{z_i}} \tag{2.10}$$

Where $\vec{z}$ is the input venctor, z_i the elements of the input vector, K the number of elements in the input vector, and e the exponential function. The function applies the exponential function to each element of the input vector, and normalizer them by dividing by the sum of exponentials [8].

Sigmoid is an activation function that maps all inputs to be between 0 and 1, using an s-shaped curve. Negative values are mapped between 0 and 0.5, and positive values between 0 and 1. The sigmoid function is given by:

$$\sigma(\mathbf{x}) = \frac{1}{1+\mathbf{e}^{-\mathbf{x}}} \tag{2.11}$$

Tanh is an activation function that is similar to the sigmoid function, however instead of mapping values between 0 and 1, the tanh function maps values between -1 and 1. The tanh function is given by:

$$\mathbf{tanh}(\mathbf{x}) = \frac{\mathbf{e}^{\mathbf{x}} - \mathbf{e}^{-\mathbf{x}}}{\mathbf{e}^{\mathbf{x}} + \mathbf{e}^{-\mathbf{x}}} \tag{2.12}$$

Loss Function

The loss function calculates the error between the predicted output and expected output. The choice of loss function plays a big role in training as it calculates the errors acted upon during backpropagation. Loss functions can be split into two main classes, regression functions and categorical functions.

Regression loss functions are used for predictive models predicting real-valued continuous values. The most widely used regression loss function is *MSE*, or mean squared error loss. MSE is given by:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} e_i^2 \tag{2.13}$$

Where e_i is the error of prediction i and n the number of data points in the prediction. The MSE value is always positive, and higher errors will be punished harder, as a result of squaring the error [8].

Categorical functions are used for classification problems, calculating the difference between probability distributions. The categorical cross-entropy loss function is given by:

$$CE = -\sum_{i=1}^{n} \mathbf{y_i} \log \hat{\mathbf{y}}_i \tag{2.14}$$

Where $\hat{y}_i$ is the i-th output value, y_i the corresponding target value, and n the number of elements in the output [8].

Overfitting

A common problem with neural networks is overfitting. This is when the model learns the training data, instead of creating a generalized model that has learned the underlying problem. A clear sign of overfitting is a model with very good training results, but that performs significantly worse on separate testing data [8].

There are various techniques for minimize overfitting, referred to as regularization methods. One common regularization method is parameter norm penalties, which attaches weights to larger weights. This penalizes networks with larger weights, which are often a sign of overfitting. Common norm penalties are the L1 and L2 vector norm penalties.

Another regularization method is dropout. For each training loop, a given amount of the nodes are blocked, so that the network has to train without them. This makes it harder for weights to be overfitted to the training data, as the model is essentially forced to train a variety of networks. During inference all nodes are active to take advantage of the more generalized model [8].

Dataset splitting

Another way to help testing and preventing overfitting is by splitting the dataset into different sets. Data is usually split into three sets, a training set, a validation set, and a testing set. The split usually heavily favours the training set, for example 70/15/15, 80/10/10, and so on for training/-validation/testing.

- The training set is usually the largest part of the data so that training can be performed on as much data as possible. This set is the one used for actual training of the models.
- The validation set is used to test the performance of a model after each epoch during training. It is important that this data is not used for the actual training, as that would increase the risk of overfitting.
- Lastly, the testing set is used for testing the model after it has been trained. This set should

be different from the training data, so that the generalization power of the model can be tested.

2.4.2 Convolutional Neural Networks

Convolutional neural networks, or convnets, are neural networks that aim to learn high-order features in the data through convolutions. The convolution operation extracts information by convolving a kernel or filter over the input, which is usually in the form of a two- or three-dimensional image (grayscale, rgb). The convolution kernel is usually a $f * f$ filter for 2D images or $f * f * 3$ for 3D-images (rgb), with each element of the kernal determined by trainable weights and biases [4]. Figure 2.2 shows an example of the convolution operation, starting in the top-left corner.

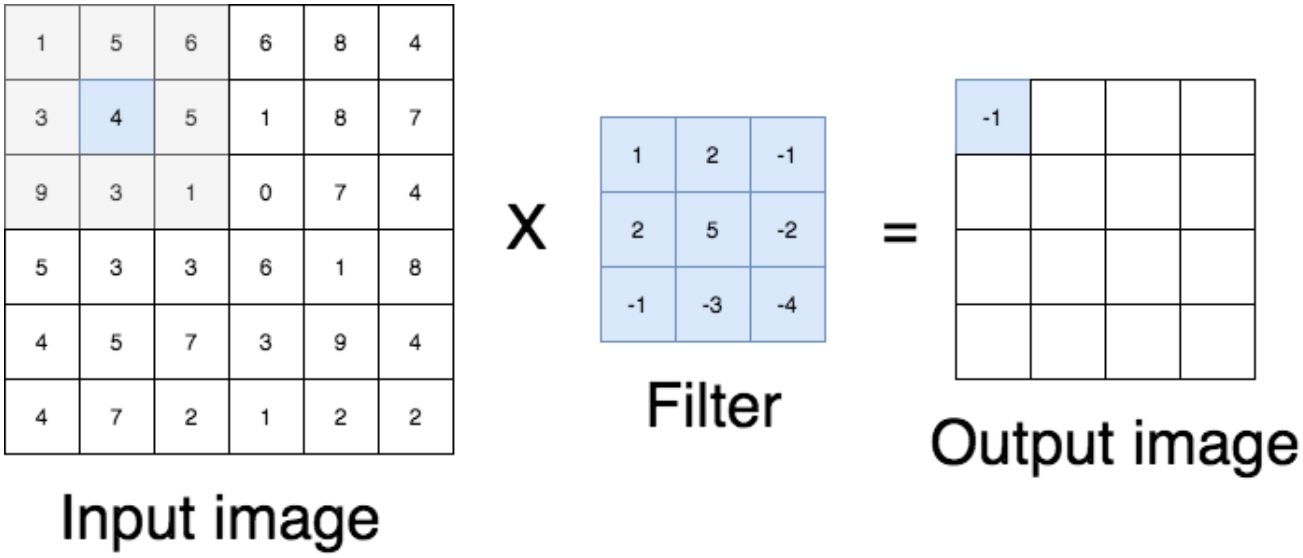


Figure 2.2: Convolution operation on top-left corner, 6x6 input, 3x3 filter, stride 1, no padding

The convolution operation generates an output that is generally smaller than the input. It is possible to correct for this by padding the input before the convolution. During convolution it is also possible to choose how many steps the kernel shifts horizontally and vertically for each step, this is called the stride [4]. In general, the output dimensions are of the form:

$$Output \;\; size = [\frac{n+2p+f}{s}+1] \;\; x \;\; [\frac{n+2p+f}{s}+1] \tag{2.15}$$

Where n is the input dimension, p is the padding, f is the kernel size, and s is the stride. There are two main types of layers in a convnet, convolution layers and pooling layers [4].

In the **convolution layer**, convolution operations are performed on the input data, with the output a feature map based on the trained weights of the filter. Unlike fully connected layers in classic neural networks, the aim of the convolution process is to extract local patterns from the input. These patterns have properties that make them ideal for problems such as computer vision problems. The patterns are *translation invariant*, which means that any patterns learned during training can be recognized anywhere in the input data. A cat is still a cat, independent of where in an image it is located. Convnets can also learn spatial hierarchies of patterns, for example one layer can learn patterns such as edges, with the next layer learning larger patterns based on these edges, and so on. This makes convnets great at learning more abstract visual concepts, such as detecting fish in an image. [4]

In addition to convolutional layers, convnets also have *pooling layers*. There are different types of pooling layers, but the general idea is to reduce the size of the inputs by pooling together smaller sections of the input. For example, a max pooling layer with a $4 * 4$ input and $2 * 2$ filter with stride 2, will result in a $2 * 2$ output, with each value the max value from the 4 corners of the input. Figure 2.3 shows an example of this. There are other pooling layers as well, such a min pooling or average pooling that calculates the minimum and average values, respectively [4].

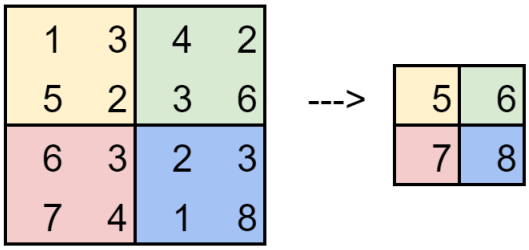


Figure 2.3: Max Pooling Layer, 4x4 input, 2x2 max filter, stride 2

The pooling layer improves the computation speed by reducing the input size, however an additional benefit is that it allows the model to become approximately invariant to small trans-

lations in the input, which makes the network more robust to small changes [8].

By successively layering convolutional and pooling layers, the aim is to extract higher level features from layers of lower level features. For example, locating and classifying objects in an image. The network architecture of a convolutional network is therefore important for any given task. Object detection is an area with a lot of research, with many proposed network architectures. For this work, Yolov4 is used.

Yolov4

Yolov4 is a state-of-the-art single-stage object detection model. It uses a pre-trained network to detect and classify objects in a frame. The output from the detection is a set of bounding boxes and confidence scores of the detected objects in the frame. Yolov4 can be broken into three parts. First stage is the backbone, which is a pre-trained feature extraction network (CSP-Darknet53). Second is the neck (PAN, SAM), which is used to collect feature maps from different stages. Finally is the head (Yolov3), which computes the bounding boxes and classification confidences of detected objects [2].

2.4.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) introduce memory to the neural network by allowing previous outputs to be used as inputs and through an internal hidden state. RNNs are mainly used for handling sequential data, such as time series data, natural language processing problems, and speech recognition [4].

An RNN consists of cells, where the inputs to each cell is a concatenation of the new input and the previous output. This allows the network to memories from previous steps. In short, the goal of a recurrent neural network is to memories past inputs that have an effect on the next output, and use that in the prediction of the next output. For example, predicting the next step in a time series requires knowledge about the previous inputs. The classic RNN is able to memories timesteps in the short term, however the further back in time there is an increasingly high chance the information will be lost. This is due to the vanishing gradient problem, where the effect of previous timesteps become increasingly small. As mentioned the input to a cell

is a concatenation of the new input and the output from the previous cell, which means the degree to which each of the previous cells contribute diminish the further back in time it is. This problem is addressed in the long short-term memory (LSTM) recurrent neural network [4], which is a variation of the classi RNN.

LSTM

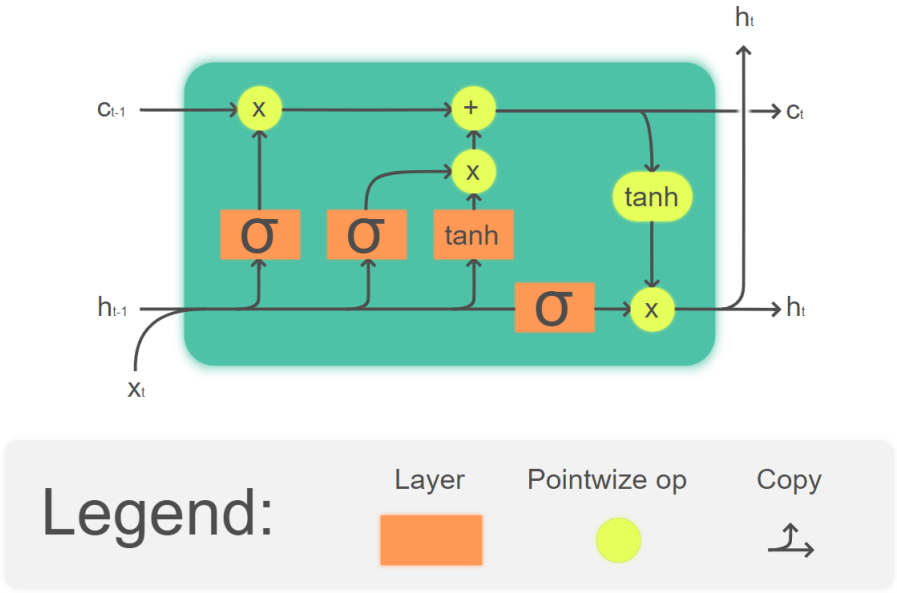


Figure 2.4: LSTM Cell. Image by Guillaume Chevalier, distributed under CC BY-SA 4.0 license. URL: https://commons.wikimedia.org/wiki/File:The_LSTM_Cell.svg

The idea behind the LSTM cell is to have a cell state that carries information through each cell with minimal interaction, such that information from cells far behind in time has a chance of passing through. The interactions between the cell state and inputs to the cell are done through *gates* [18].

The first gate is the *forget gate*. This gate is used to determine what information should be removed from the cell state. A concatenation of the input and the hidden state from last cell are sent through a sigmoid activation function, resulting in a value between 0 and 1. The degree to which information is kept or removed from the cell state depends on the output value from this gate, with 0 being forget all, and 1 keep all. The cell state is updated through a pointwise multiplication with the forget gate's output [18]. The formula for the forget gate is:

$$\mathbf{f_t} = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.16}$$

With W_f and U_f the weight matrices, b_f the bias vector, x_t the input and h_{t-1} the previous hidden state (output) [4].

The next two gates handle which new information should be added to the cell state. The input and hidden state are used as inputs to both a sigmoid and a tanh activation function, and the outputs are multiplied with each other and added to the cell state. The tanh function attempts to extract helpful information from the input and previous hidden state, while the sigmoid function decides which of the tanh outputs to add to the cell state [18]. The two functons are:

$$\mathbf{i_t} = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2.17}$$

$$\mathbf{k_t} = \tanh(W_k x_t + U_k h_{t-1} + b_k) \tag{2.18}$$

With W_i, W_k, U_i and U_k the weight matrices, b_i and b_k the bias vector, x_t the input and h_{t-1} the previous hidden state (output) [4]. The updated cell state is then given by:

$$\mathbf{c_t} = i_t k_t + c_{t-1} f_t \tag{2.19}$$

The final gate is the output gate, which calculates the output and next hidden state. First, the previous hidden state and input values go through a sigmoid function. The updated cell state is sent through a tanh function, as well as to the next cell. The output of the tanh and sigmoid functions are then multiplied using a pointwise operator, giving the new hidden state of the cell. This hidden state is given as the output from the cell, and it is sent to the next cell [18]. The formulas for the output gate and next hidden state is:

$$\mathbf{o_t} = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.20}$$

$$\mathbf{h_t} = o_t \sigma(c_t) \tag{2.21}$$

With W_o and U_o the weight matrices, b_o the bias vector, x_t the input and h_{t-1} the previous hidden state (output) [4].

Chapter 3

Materials and methods

This chapter gives an overview of all the materials and methods used for this thesis.

3.1 Datasets

One of the main challenges for this thesis was getting suitable datasets to work with. A dataset would need enough video material of fish on a moving conveyor belt for both training of networks and testing of developed solutions. Through early research no suitable publicly available dataset was found, and it was decided that a new dataset would be created. This also has the benefit of using video material from actual locations were the solution can be deployed in the future.

To gather video material, a PTZ camera was mounted by Stranda Prolog at one of the fish processing facilities using their equipment. The camera was mounted above an active conveyor belt, and through remote control the camera could be controlled to get a suitable view of the conveyor belt. The main goal when choosing the camera position was to get as much of the conveyor belt in view of the camera, without capturing areas where no fish would pass. In the video materials used the conveyor belt runs horizontally through the camera view, covering most of the video area.

Figure 3.1: Example frame of fish on the conveyor belt

3.1.1 Object Detection

With the video material gathered, the next step was to create a dataset for object detection (Yolov4). Yolov4 requires a dataset with a set of images and corresponding labels (bounding boxes) of the objects in the images. First step in this process was to extract images from the videos, which was done with a simple OpenCV script that extracted frames from the video at a given interval. To prevent overfitting, frames were extracted every two seconds, with varying amounts of fish in the frames. In total 500 frames were extracted to be used for dataset creation, refered to as subset 1.

In addition to detecting fish, this thesis also attempts to detect damages to fish. Therefore, in addition to images extracted at a fixed interval from training videos, a curated video where damage is present was also created. This video contains video sections where one or more fish is damaged, compiled from the training videos. From this video, a further 400 frames were extracted, with at least one instance of damage in each frame, refered to as subset 2.

With frames extracted from the videos, the next task was to manually label the images. This can be a time consuming task, but with the aid of tools the process is fairly simple. For this thesis, the tool *LabelImg* was used, which is a graphical labeling tool that can automatically

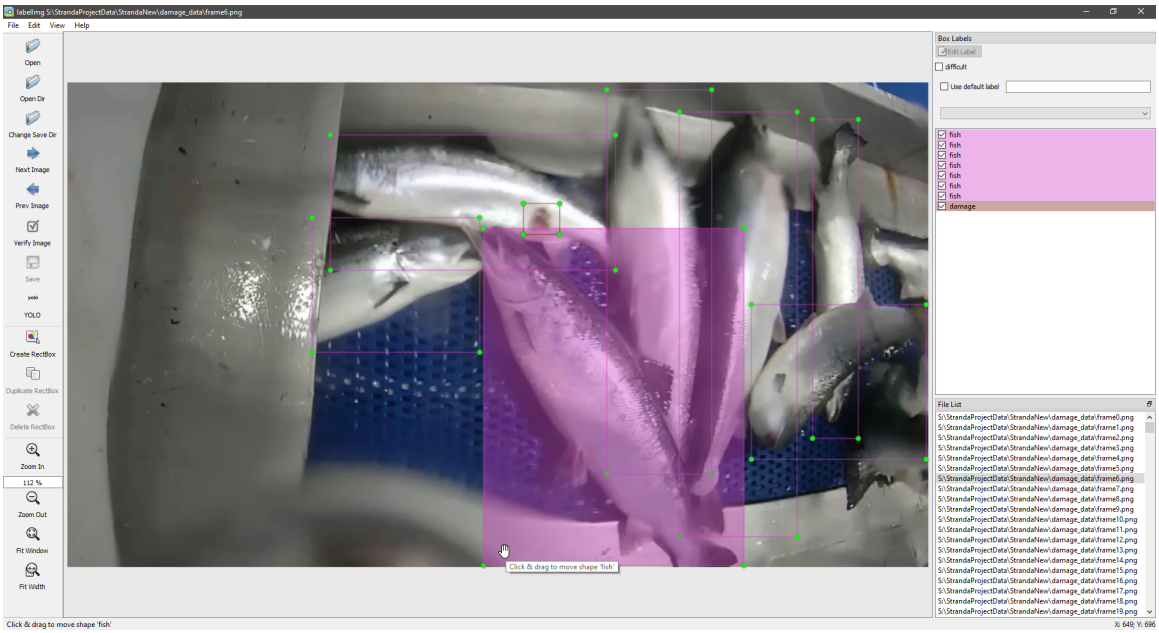


Figure 3.2: Labelling test data using LabelImg

save created labels in the correct format for Yolov4. Yolov4 requires the labels for an image to be in a text file with the same name as the image file, with each label on a seperate line in the file. The labels are on the form [*ID, x, y, w, h*], where ID is the object ID (0 for fish, 1 for damage), x and y are the normalized coordinates of the top-left corner of the bounding box, and w and h are the normalized width and height of the bounding box. In total, 8979 fish and 628 instances of damage was labeled in the 900 images of the dataset, as can be seen in table 3.1.

	Subset 1	**Subset 2**	**Total**
Images	500	400	900
Fish	4591	4388	8979
Damages	147	481	628

Table 3.1: Overview of number of fish and damages in the dataset

3.1.2 Motion Prediction and Classifying Dead / Alive Fish

The second dataset required for this thesis was a dataset to train recurrent neural network models for both motion prediction, and for classifying whether or not fish are alive (moving). Both of these networks require data in the form of sequences of bounding boxes from subsequent

frames from individual fish. This would have been very time consuming to manually create from scratch, so a different approach to creating the dataset was adopted.

As explained further in 5.2.1, the first solution derived used a kalman filter to perform motion prediction, which worked well enough to successfully track most fish correctly. Thus, this solution could be used to automatically create data based on actual detected bounding boxes.

With a slightly modified code, each of the tracked fish also stored all the bounding boxes, and upon the death of a track the entire bounding box history was saved to a file, with each bounding box on the form [*f, x, y, w, h*], where f is the frame number, x and y the normalized coordinated of the top-left corner and w and h the width and height of the bounding box.

In addition to storing the data, a video with the bounding boxes was created, so that the data could be checked for false data and incorrect tracking. By manually going through the video frame by frame, any incorrect or incomplete data could be removed. Examples of this include identity switches, false detections, and duplicate detections with incomplete data. Data that was partially broken was cleaned by removing the broken parts, so that they could be used in the dataset as well.

Additionally, to prepare the data for classifying dead and alive fish, the data was also grouped into two subsets based on whether or not the fish were moving, indicating that they are alive. In total, 89 fish were tracked, with between 33 and 477 frames (around 0.5 to 8 seconds) of bounding boxes depending on how long the fish were in the video (or how intact the tracks were).

	Dead Fish	**Alive Fish**	**Total**
Fish	53	36	89
Bounding boxes	18627	7474	26101

Table 3.2: Overview of dataset for motion prediction and dead / alive classification

As explained further in 5.2.2, the inputs to the networks are of a fixed length, such as 10 or 20 frames of bounding boxes. Thus, the data was split into sections such that each possible fixed length section became one data input. To allow for flexibility during development and testing, the dataset is stored as 89 individual files with the full track history of the fish, and the splitting is done at run-time depending on network requirement.

As can be seen in 5.4, this also allows for flexibility in what form the data is used. The network

for classifying fish state uses the difference between subsequent frames, for example, which can be created from the same dataset during run-time.

3.1.3 Test Videos

Two test videos were used for testing the developed solutions. It was important that these videos were long and varied enough to get good data during testing, such that potential weaknesses could be found. In these videos there are varied amount of fish at any given time, as well as both alive and dead fish. Table 3.3 gives an overview of the test videos.

	Test Video 1	**Test Video 2**	**Total**
Length (m)	12:00	20:00	32:00
FPS	60	60	60
Fish	385	1125	1510
Fish per minute	32.1	56.3	47.2
Damages	20	43	63
Dead fish	289	836	1125
Alive fish	96	289	385

Table 3.3: Overview of test videos

3.2 Object Detection

For object detection, the state-of-the-art Yolov4 convolutional neural network architecture is used [2]. Yolov4 uses the darknet framework, which is an open source neural network framework written in C and CUDA. It supports CPU and GPU computations, and through GPU computation with CUDA enabled graphics cards is a fast framework for training object detection models [19].

For training, the model was configured based on the directions given in [2], and trained using an NVIDIA RTX 2080Ti. See section 5.1.1 for the training results.

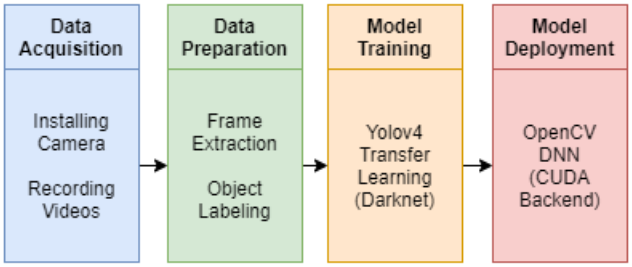


Figure 3.3: Object detection machine learning pipeline

After the model was trained to detect fish and damage, it was deployed using OpenCV's DNN module, with CUDA backend enabled to fully utilize the power of the graphics card [17]. CUDA is a parallel computing platform and programming model made by NVIDIA, to support general computing on CUDA-enabled GPUs. It can dramatically increase the performance of compute heavy models, such as the Yolov4 CNN model used for object detection in this thesis.

3.3 Motion Prediction and Classifying Dead / Alive Fish

TensorFlow[1] was used for creating the models used in motion prediction and classifying dead / alive fish. TensorFlow is an end-to-end open source platform for machine learning, with a focus on easy model building through high-level APIs like Keras[2]. Keras is focused on simple and consistent APIs that are clear and human-readable, with the goal of minimizing time spent on boilerplate, and more time spent on implementing and testing ideas . This is ideal for the adopted methodology of rapid iterations through experimentation.

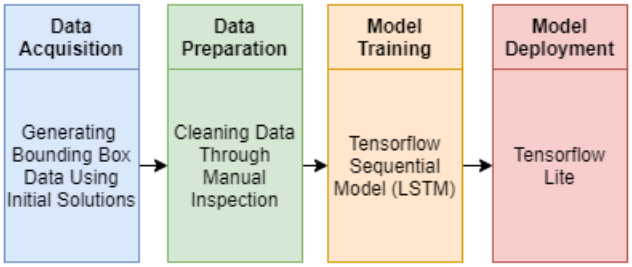


Figure 3.4: Machine learning pipeline for motion prediction and classifying dead / alive fish

[1] TensorFlow: https://www.tensorflow.org/
[2] Keras: https://keras.io/

Both the models were set up as sequential models using Keras' LSTM and Dense (fully connected) layers. The motion prediction models takes a sequence of bounding boxes as input and outputs a predicted bounding box for the next steps. The network has an input LSTM layer, connected to a fully connected dense layer with 4 output units. Both layers use ReLu as the activation function, with a recurrent dropout of 0.5 in the LSTM layer to limit overfitting.

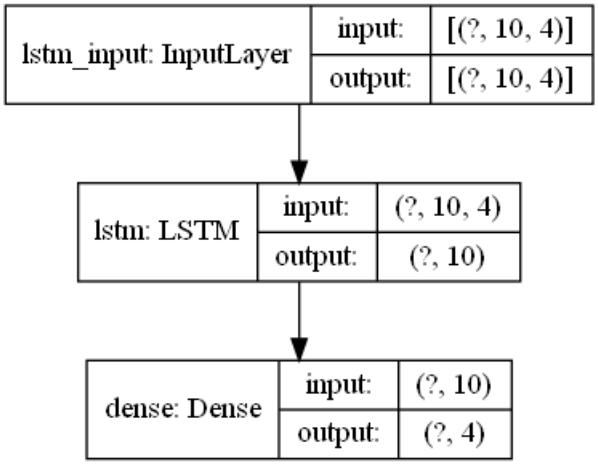


Figure 3.5: Motion prediction network model graph

To perform the training, the prepared dataset was parsed so that the full training data included every sequence of bounding boxes and ground truths from the full bounding box histories of the 89 fish. 20% of the dataset was set aside for testing after the model was trained, with the remaining data used for training. Using a sequence length of 10, the training data consisted of 11,873 input sequences. This data was then split into testing and validation data with a 80/20 split, and used to train and validate the models. Results from different model sizes can be found in section 5.2.2

The model used to classify if fish are dead or alive is largely based on the same model used to predict motion, with a few notable differences. First of all, instead of using a sequence of bounding boxes, the sequence uses the *difference* between the bounding boxes and centroids from frame to frame, as well as the aspect ratio. The other notable difference is the fully connected layer, which consists of 2 units, each with the probability of the sequence belonging to the category dead or alive. For classificaton problems, softmax is the most suited activation function, thus that is used here.

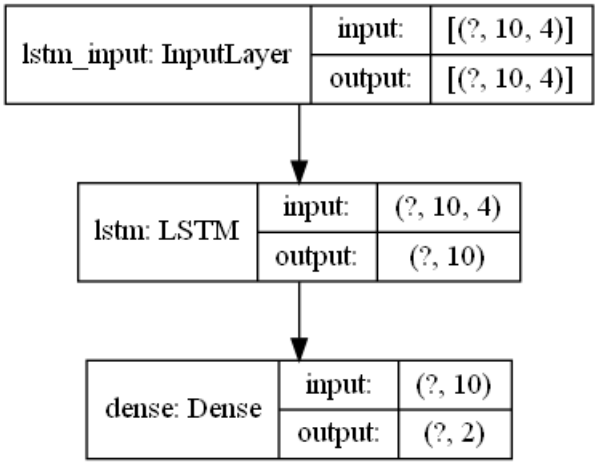


Figure 3.6: Dead / alive classification network model graph

TensorFlow Light[3] was used to deploy both models. TensorFlow Lite is a deep learning framework for on-device inference, and is designed to improve inference speed without sacrificing accuracy.

3.4 Programming Language

The programming language used for the thesis was Python. Python is a high-level general-purpose programming language, and is among the most popular programming languages, especially within machine learning and data science[4]. It supports a range of frameworks and APIs for machine learning, such as TensorFlow and Keras.

3.5 Hardware

The development and testing was done using an Alienware Area-51 desktop. The specifications are described below, though during testing only one of the GPUs were used.

[3] TensorFlow Lite: https://www.tensorflow.org/lite
[4] https://www.python.org/

Processor (CPU)	AMD Ryzen Threadripper 2950X
RAM	64GB 2667 MHz DDR4
Graphics Card (GPU)	2x NVIDIA GeForce RTX 2080 Ti
VRAM	2x 11GB GDDR6
Operating System	Windows 10 Education

Table 3.4: Hardware specifications

Chapter 4

Implementation

This chapter will give a detailed overview of the proposed solution, with descriptions of the different parts of the implementation. It will cover how multiple object tracking (MOT) and deep learning was implemented in order to perform fish counting, and how the fish inspection parts of the solution are connected to the MOT fish tracking.

4.1 Solution Overview

The overview diagram in figure 4.1 shows how the developed solution is structured for each frame of the video. There are four main parts or modules to the implementation, each constituted of various smaller parts. First, each frame is sent to module *A*, which is responsible for the first stage of the MOT algorithm: object detection. This is done with the trained Yolov4 model, deployed using OpenCV's DNN module with CUDA backend enabled. The resulting object detections, if any, are then classified and split into either fish detections or damage detections. The fish detections are then sent to module *B* and damage detections to module *C*, which are both responsible for the object tracking part of the MOT algorithm. Section 4.1.2 goes into detail about the object tracking, with details on how the two modules work, as well as the key differences between them. Module *B* and *C* are linked through a function that assigns each detected damage uniquely to a fish track for damage inspection. Classifying the state of a fish (dead / alive) is done as a step in the fish tracking module, thus both of these inspection metrics are stored in the fish tracks. Module *D* handles the actual counting and inspection. As fish tracks

pass an imaginary line in the frame, they are counted as a detected fish, and the inspection metrics are logged.

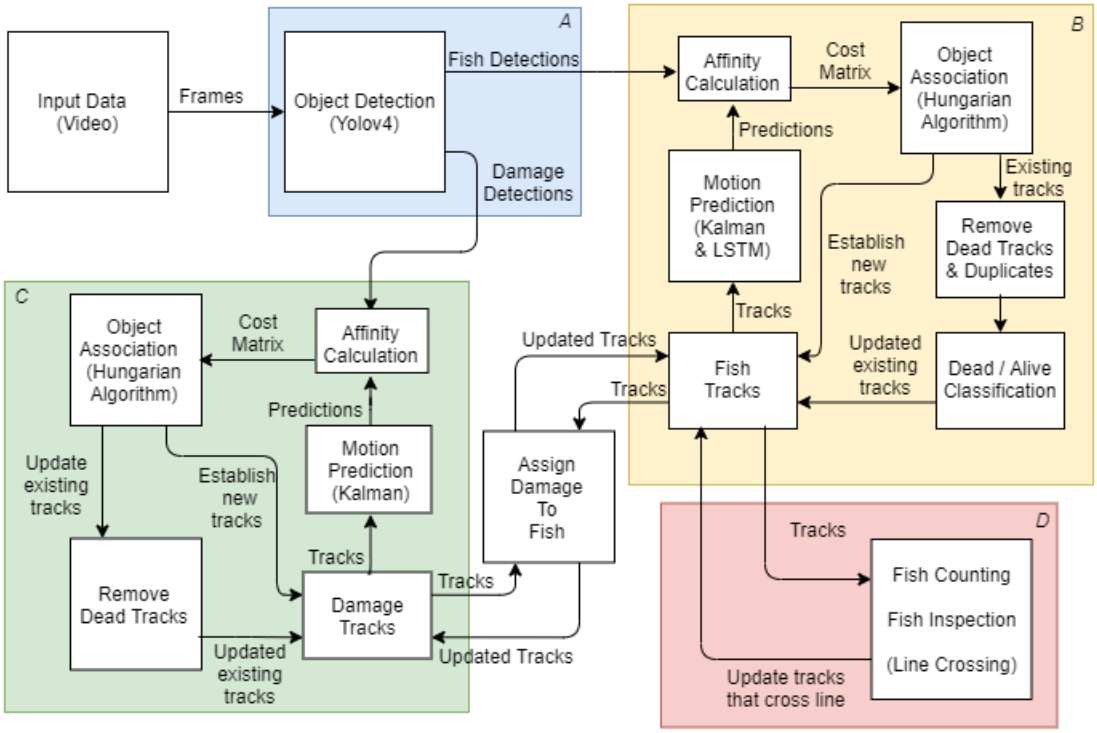


Figure 4.1: Overview diagram of the developed solution

4.1.1 Object Detection (A)

The first step of the object detection is to extract frames, which is done using OpenCV. Throughout testing frames were extracted from recorded videos, however this can easily be adapted to extract frames from a connected camera device in real-time. The same OpenCV methods are used for both, thus no change in the code is necessary beyond changing to reading from camera instead of file.

Next, the extracted frame is used as input in the Yolov4 object detection model. The training results for the model are described in section 5.1.1. The model is deployed with the OpenCV DNN module using CUDA as the backend, which enables it to take full advantage of the CUDA-enabled GPU for better performance. The output from the detection model is a set of bounding boxes and the corresponding classes and confidence scores. These are used to classify if the

detections are fish or damages, as well as to calculate the centroid of the objects. The bounding boxes and centroids are then sent to module *B* and *C*, where they are matched against the existing tracks or established as new tracks.

4.1.2 Object Tracking (B + C)

The next part of the MOT algorithm is the object tracking part, which consists of three stages: motion prediction, affinity calculations and association. Both module *B* and *C* largely use the same methods for these stages, so this section will describe the methods generally, diverging only where there are differences between the modules.

Motion prediction is the main stage where the two modules differ. The predictions for damage tracks (module *C*) are calculated using kalman filters, which predict the next centroid of the objects. Fish tracks (module *B*) on the other hand use a recurrent neural network (LSTM) model to predict the bounding boxes of the objects.

When using kalman filters, motion prediction consists of two main steps. First, the next centroid is predicted based on the internal state of the filter. Then the next step updates the internal state based on the error between the predicted centroid and actual centroid. This step requires the actual centroid to be known, thus this step is performed after the affinity and association stages. See section 2.2 for more details on the kalman filter.

The LSTM motion prediction model consists of a single step, the prediction. The model input is a fixed-length sequence of the previous bounding boxes of the object. At the birth of tracks, this is zero-padded to match the required length. Section 5.2.2 details the accuracy and accuracy loss by using zero-padding. The output from the model is the predicted next bounding box, from which the predicted centroid can also be calculated.

The next stage of the MOT algorithm is the **affinity stage**. In this stage the aim is to calculate a score or cost for all the possible pairs of detection from the object detection stage and predictions from the motion prediction stage. The cost measures how low the distance between all the pairs of centroids are, using the euclidean distance. Distance between centroids is the cost metric for both fish and damage tracking. The result from this stage is a cost matrix, which is

used in the association stage.

The final stage is the **association stage**, where the aim is to pair detections to the correct tracks. This is accomplished using the Hungarian algorithm, which is an assignment algorithm that minimizes the overall cost of assignments. The cost matrix is used as the input, with detection/prediction pairs as the output. The matches are then used to update the tracks with the new detections, and to establish new tracks for detections not matched to any tracks. If a track is not matched to a detection, the prediction is used as the actual value. However, if too many frames have passed without a match the track is assumed to have disappeared from the frame, and the track is removed.

Additionally, one of the issues that arose during testing was the occasional duplicate fish track. To address this there is a function that removes the shortest out of two tracks if the overlap between them is too large for too many frames. The IoU metric is used to calculate the overlap.

4.1.3 Classifying Dead / Alive Fish (B)

To classify whether or not a fish is alive, an LSTM model is used to detect movement. When fish move the bounding boxes tend to change more frequently and to a bigger degree than fish lying still, which is utilized to classify if fish are dead or not. The LSTM model input is a fixed-length sequence of the *difference* in bounding boxes and centroids from frame to frame, as well as the aspect ratio, all calculated from the bounding box history used in motion prediction. The output is the confidence that the fish is moving, which is used to classify if the fish is alive or not. If the confidence is high enough for a long enough period, the fish is classified as alive.

4.1.4 Assigning Damage To Fish

For damage inspection, each detected damage is associated with a fish through an assignment method. For each frame where one or more damage tracks are present, a cost function is used to determine which fish is closest to the damage. This cost function is a combination of IoU and euclidean distance between damage and fish centroids. The smallest cost indicates the fish that is closest to the damage. If a fish track is marked as the closest for a given number of frames, it is

marked as damaged, and the damage track is marked as counted and can no longer be assigned to any other fish. This way, if the fish flips over or the damage is no longer detected, the fish is still marked as damaged. If the fish track disappears before being counted (as a result of incorrect tracking, for example), the corresponding damage track is marked as not counted, so that it can be assigned to a new fish.

4.1.5 Counting and Inspection (D)

The final module handles the counting of fish and logs the inspection data. As a fish track passes a fixed line in the frame, the fish counter is increased by 1, the inspection data is logged, and the track is marked as counted. Additionally, tracks that originate behind the line are marked as such, and should they pass the line backwards and then forwards again, they will not be counted.

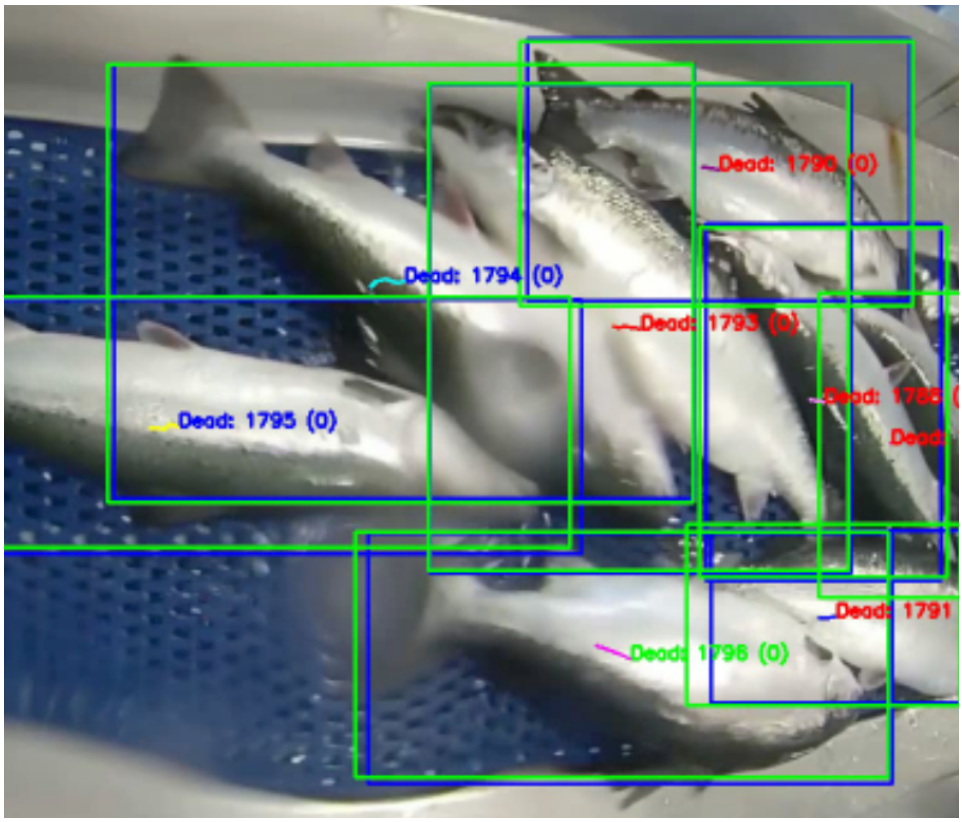


Figure 4.2: Example output frame. Blue tracks are not counted yet, red have been counted, and green entered the frame from behind the counting line

Chapter 5

Experiments and Results

This chapter describes the experiments conducted for the different parts of the solution, and details the results and findings from the experiments.

5.1 Object Detection

The first goal of the project was to develop a dataset and train an object detection model to detect fish and damages in video frames. As described in section 3.1.1, the dataset was prepared through manually labeling objects, which resulted in 900 images, with a total of 8979 labeled fish and 628 labeled instances of damage. To train the yolov4 model the instructions and recommendations from the official yolov4 github page were followed [1], resulting in the following configuration:

batch	**64**
subdivisions	**16**
max batches	**6000**
network size	**width = 416, height = 416**
classes	**2**
filters	**21**

Table 5.1: Yolov4 configuration

[1] https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects

5.1.1 Yolov4 training results

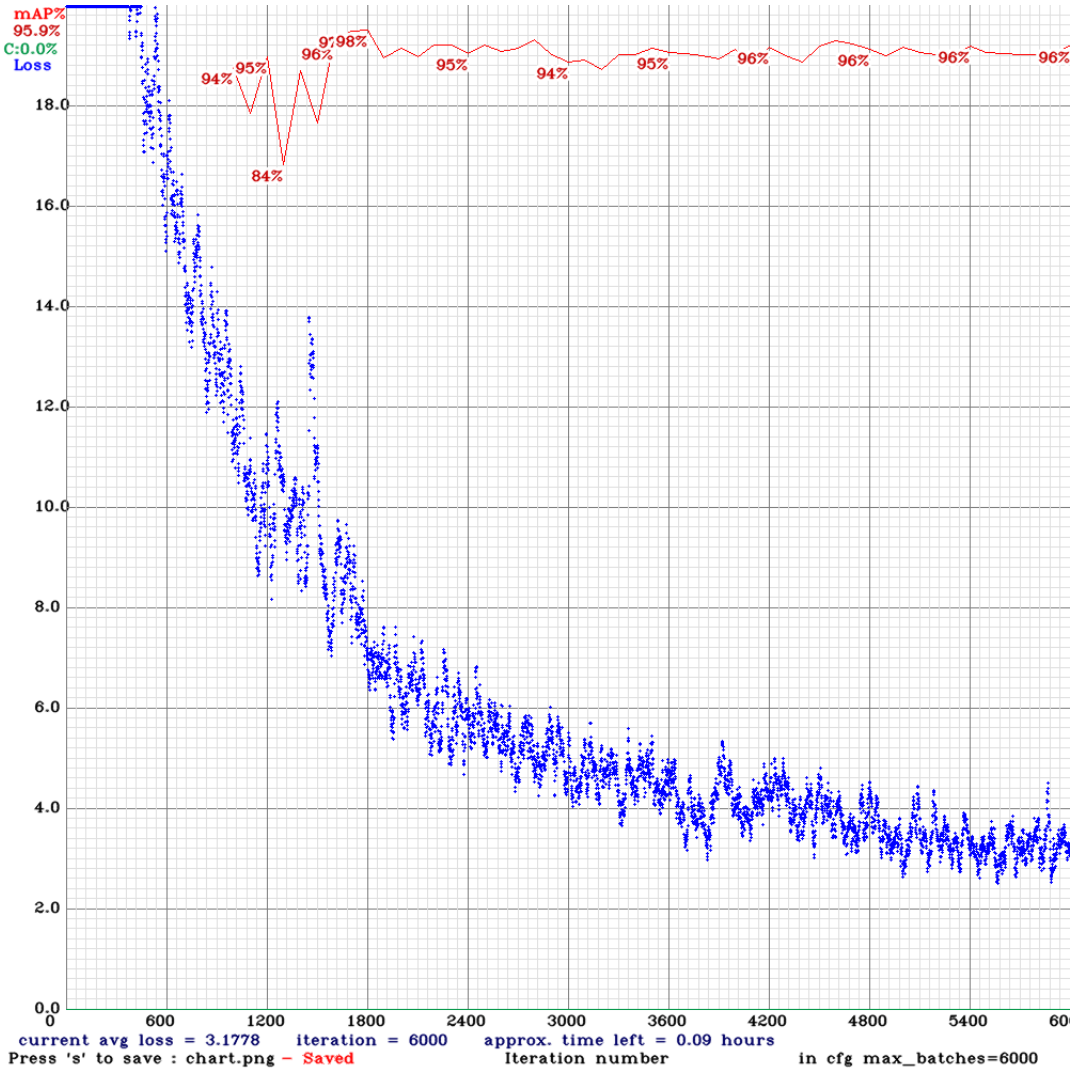
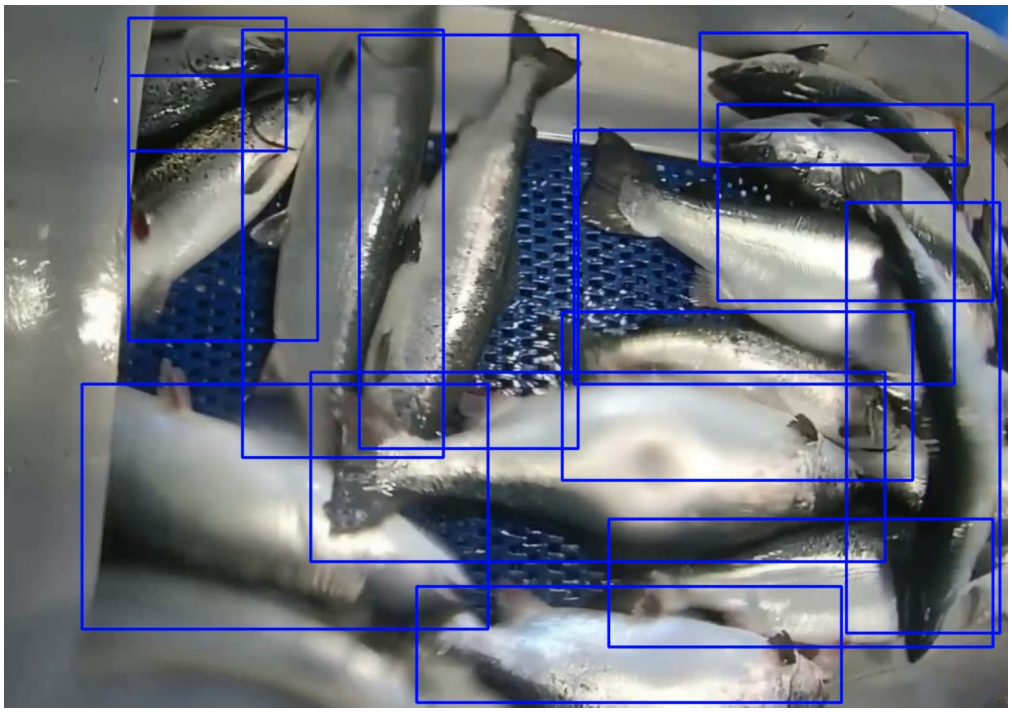
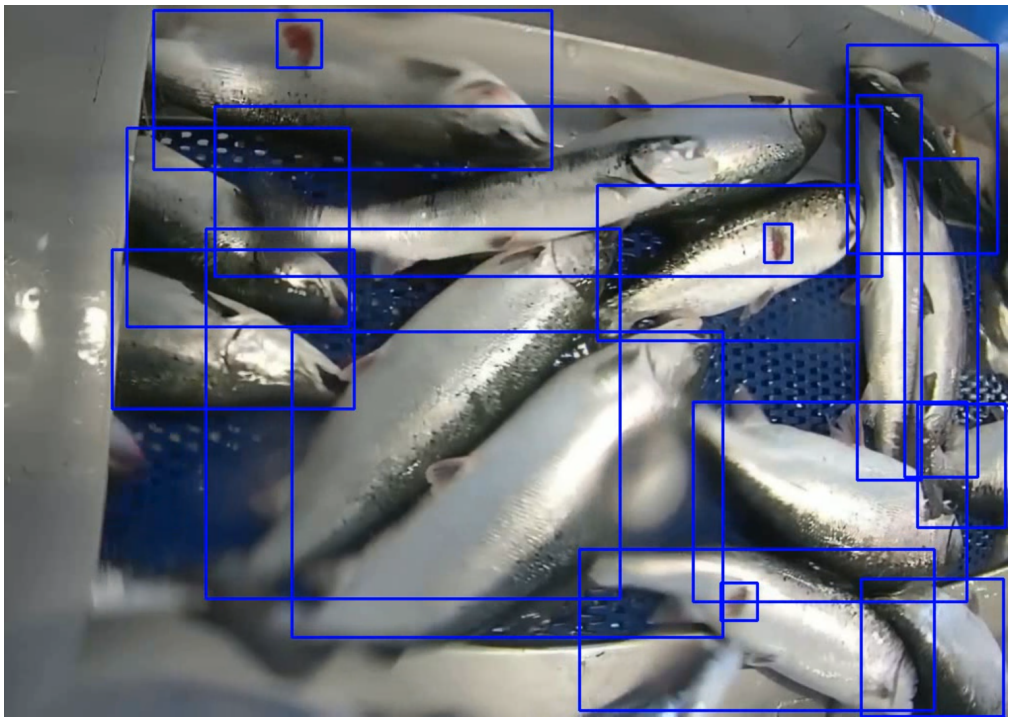


Figure 5.1: Training results from Yolov4 object detection

Figure 5.1 shows the training results from the model training, with the best results after around 1800 iterations. Testing the model on testing videos shows similarly good accuracy, indicating that the model has been able to generalize from the training data. Figure 5.2 depicts the detected bounding boxes from two example frames that have relatively many fish, including instances of damage in the second example.



(a) Example One



(b) Example Two

Figure 5.2: Example of fish and damages successfully detected in more crowded scenes

5.1.2 Deployment using OpenCV w/CUDA

After completed training, the next step was to deploy the model. To take full advantage of the machine hardware described in 3.5, the model was deployed using OpenCV's DNN module with CUDA backend enabled. To evaluate the performance, a series of tests were performed to find the average inference time.

Test Run	Run 1	Run 2	Run 3	Run 4	**Average**
Inference time (ms)	11.88	12.02	11.92	11.92	**11.94**
FPS	84.18	83.19	83.89	83.89	**83.75**

Table 5.2: Yolov4 inference times using OpenCV with CUDA backend on test computer

The results show that the object detection runs consistently above 80 frames per second on the test machine, which is above the 60 frames per second of the test video. This shows that the object detection model can run in real-time with a powerful GPU when deployed using the OpenCV DNN module.

5.2 Motion Prediction

The next stage of development was to find the best motion prediction model for the project. The two main methods looked at for this were the use of kalman filters to predict centroid position and an LSTM model to predict bounding boxes. The kalman filter did not require any training data, thus experiments to test how well it performs were conducted first. The aim was to discover how suited the kalman filters were for this implementation, and should the results be good enough it was planned that this method could be used to gather data to train the LSTM model.

5.2.1 Kalman Filter

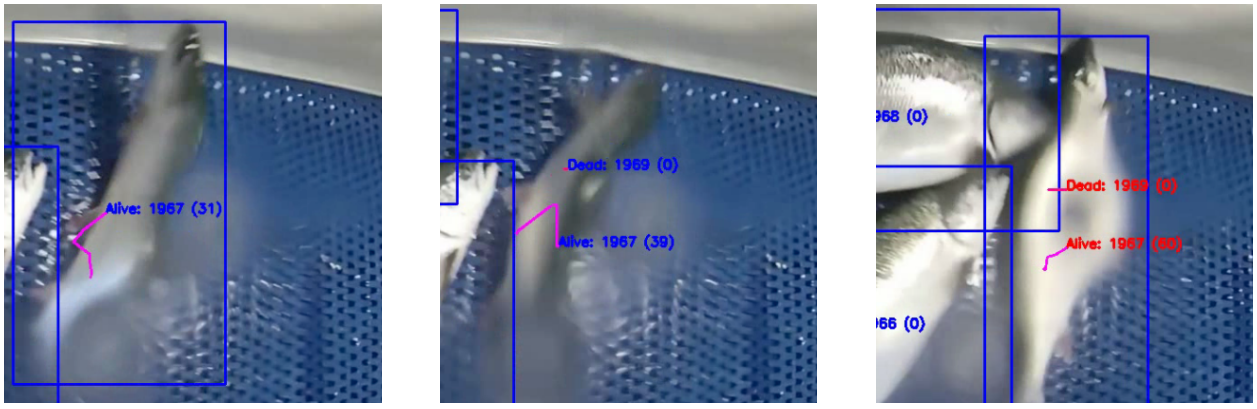
Tests of the kalman filters for motion prediction were conducted on the testing videos described in section 3.1.3. The chosen affinity cost function was the euclidean distance between centroids, and the hungarian algorithm was used for the association stage. The results were mea-

sured using a number of metrics: fish count is the total amount of counted fish in the video, false positives measures how many times fish were measured more than once, false negatives measure how many fish were not counted, identify switches measure how many times the identity between two fish were switched, fragmented tracks measure how many times a track was broken and then restarted again as a new track, and total tracks measure the actual number of tracks started. Total tracks also include tracks that were present for only a few frames and other incorrect tracks, thus this is expected to be higher than the number of counted tracks (fish).

Test Video	Fish Count	False Posities	False Negatives	Identity Switches	Fragmented Tracks	Total Tracks
Video One	390/385 (101.3%)	7 (1.8%)	2 (0.5%)	14	21	499
Video Two	1146/1125 (101.9%)	41 (3.6%)	17 (1.5%)	75	69	2076
Combined	1536/1510 (101.7%)	48 (3.2%)	19 (1.3%)	89	90	2575

Table 5.3: Test results with kalman filters for motion prediction

The results from the test videos show that 1491 out of 1510 fish were correctly counted (98.7%), however due to the amount of false positives the actual count was higher than the expected count (101.7%). This is in large part down to identity switches and track fragmentation occurring while the fish passes the line used for counting.

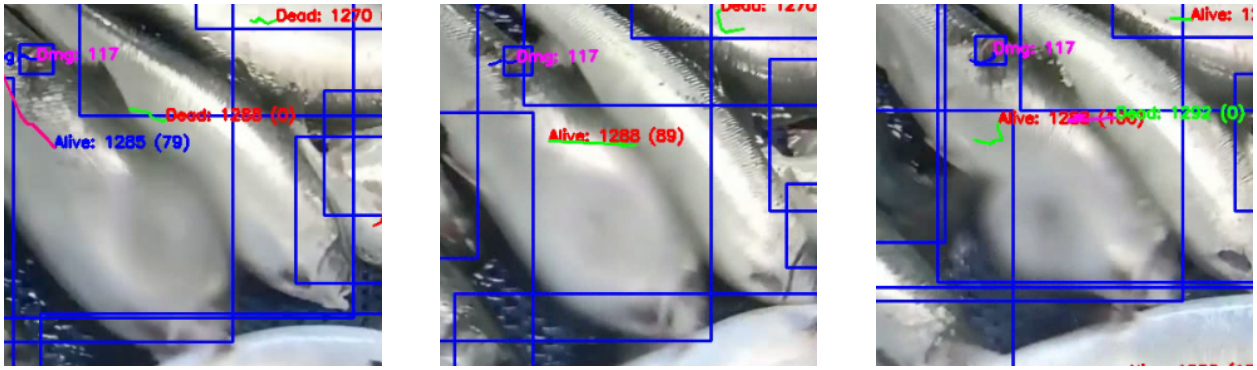


(a) Frame 1 (b) Frame 8 (c) Frame 40

Figure 5.3: Example of a fish being counted twice due to track fragmentation

Figure 5.3 shows an example where a fish is counted twice due to track fragmentation. After frame 1 the track was momentarily lost, and a new track was established due to the next detection being too far away from the previous one. In the following frames both tracks alternate

being the closest to the new detections, so that both tracks were moved through the counting line.

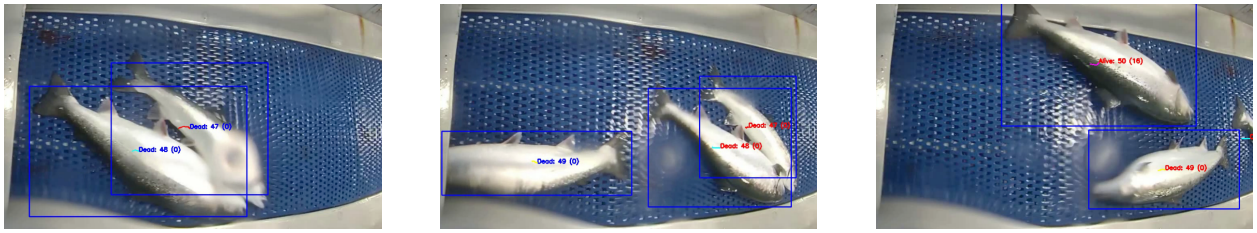


(a) Frame 1 (b) Frame 17 (c) Frame 34

Figure 5.4: Example of a fish not being counted due to identity switching

Figure 5.4 shows an example where a fish is not counted due to identity switching. After frame 1 there is an identity switch where the track of the already counted fish (red 1288) is switched to the one not counted. However, as the track has already been counted it will not be counted again, thus the other fish is not counted when it passes the counting line.

The majority of issues occurred in crowded sections, especially with alive fish, however the motion prediction and tracking using kalman filters proved very stable in sections without a lot of fish. Figure 5.5 shows an example where four fish are tracked and counted, without any identity switches, fragmented tracks, or other issues.



(a) Frame 1 (b) Frame 158 (c) Frame 395

Figure 5.5: Example of fish successfully tracked and counted

5.2.2 Recurrent Neural Network (LSTM)

The next set of experiments revolved around developing and testing LSTM models for motion prediction, and comparing the results to kalman filters.

Creating Dataset

Creating a dataset for motion prediction manually would require a lot of time, thus it was tested if the previously developed solution using kalman filters could be used to automatically create the dataset from test data. As described in section 5.2.1 there were some issues with identity switching and fragmented tracks, but largely the tracks were intact for the duration they were in frame. Thus, the code was altered to track all the bounding boxes for each track, and upon track death the data was written to a text file with the track id as file name.

```
[3911, 0.29375, 0.2375, 0.20078125, 0.2375]
[3912, 0.28671875, 0.20833333333333334, 0.2390625, 0.26805555555555555]
[3913, 0.28671875, 0.20833333333333334, 0.23984375, 0.26805555555555555]
[3914, 0.28671875, 0.20833333333333334, 0.23828125, 0.26666666666666666]
[3915, 0.2734375, 0.1638888888888889, 0.33125, 0.3458333333333333]
[3916, 0.27421875, 0.1638888888888889, 0.33125, 0.3458333333333333]
[3917, 0.33515625, 0.12222222222222222, 0.24296875, 0.5472222222222223]
[3918, 0.33515625, 0.12222222222222222, 0.24296875, 0.5486111111111112]
[3919, 0.31640625, 0.1361111111111111, 0.2625, 0.525]
[3920, 0.31640625, 0.1375, 0.2625, 0.525]
[3921, 0.31640625, 0.1361111111111111, 0.2625, 0.525]
[3922, 0.3296875, 0.1375, 0.2984375, 0.5194444444444445]
[3923, 0.33046875, 0.1375, 0.29765625, 0.5194444444444445]
[3924, 0.3390625, 0.04027777777777778, 0.36953125, 0.5152777777777777]
[3925, 0.3546875, 0.06944444444444445, 0.34453125, 0.49444444444444446]
[3926, 0.35390625, 0.06944444444444445, 0.3453125, 0.49444444444444446]
[3927, 0.3984375, 0.0125, 0.3296875, 0.55]
[3928, 0.3984375, 0.0125, 0.3296875, 0.55]
[3929, 0.38359375, 0.059722222222222225, 0.3515625, 0.5388888888888889]
[3930, 0.384375, 0.06111111111111111, 0.35078125, 0.5375]
[3931, 0.43515625, 0.16527777777777777, 0.296875, 0.525]
[3932, 0.43515625, 0.16527777777777777, 0.29765625, 0.525]
[3933, 0.43515625, 0.16527777777777777, 0.29765625, 0.525]
[3934, 0.35390625, 0.21666666666666667, 0.35234375, 0.3597222222222222]
```

Figure 5.6: Example data generated using kalman filter tracking

Each data point was on the form [frame id, x, y, h, w], where frame id is the frame number in the video, x and y the normalized top-left point of the bounding box, and h and w the normalized height and width of the bounding box. In addition to storing the data, a video was recorded at the same time so that manual inspection could be used to remove any faulty tracks. This method proved to be successful and saved many hours of manual work, resulting in the datasets described in section 3.1.2.

LSTM network architecture

The next challenge was to develop a suitable network architecture for motion prediction. The two main criteria for a good architecture were speed and accuracy, so a series of experiments were conducted to find the right compromise. First test looked at a simple network consisting of two layers, an LSTM layer as the input layer and a fully connected output layer for the predicted bounding box. The input sequence consisted of 10 subsequent bounding boxes, and to find the optimal number of hiden units in the LSTM layer different number of units were tested. The fully connected layer had 4 outputs for the bounding box, with ReLu as the activation function.

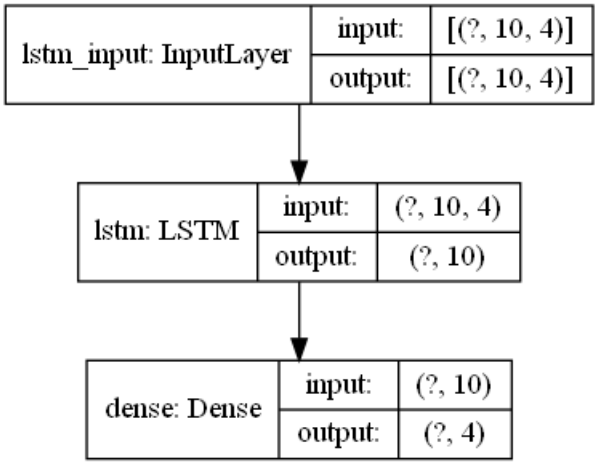


Figure 5.7: Motion prediction network model graph (10 hidden units)

The metrics used to test the performance of the networks were average inference time in milliseconds (Time), the IoU between predicted and actual bounding boxes (IoU), and the euclidean distance in pixels between the predicted and actual centroids (Dist.).

Units	1	5	10	30	100	300	500
Time	4.2 ms	4.3 ms	4.3 ms	4.4 ms	4.5 ms	5.1 ms	6.3 ms
IoU	39.5 %	94.3 %	94.3 %	94.2 %	93.8 %	94.1 %	93.9 %
Dist.	117.2px	4.4 px	4.3 px	4.6 px	5.0 px	4.6 px	4.7 px

Table 5.4: Results for different LSTM network sizes

The results in 5.4 show that the difference in accuracy is quite small for the different number of hidden units (excluding 1), which indicates that the small models learn as well as the larger

models. In fact, the larger models performed slightly worse, which could indicate that they were overfitting to the training data. Additionally, there is a slight increase in inference time with higher number of units due to the networks being more complex, thus 30 hidden units seem to be the optimal size for this implementation.

The input data was zero-padded for the first 9 sequences, meaning that predictions were done with between 1 and 9 true bounding boxes until 10 true bounding boxes had been recorded. To test the impact this had on accuracy for the start of a track, a separate test was conducted using the LSTM model with 30 hidden units.

True Bounding Boxes	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
IoU	83.0 %	85.4 %	85.7 %	81.2 %	91.1 %	93.0 %	93.2 %	93.5 %	93.9 %
Distance	25.7 px	7.8 px	7.9 px	8.0 px	6.3 px	7.9 px	5.1 px	6.7 px	6.6 px

Table 5.5: Testing results of zero-padded sequences (sample size = 18 for each amount of bounding boxes)

The results in table 5.5 show that the accuracy is lower, especially for the first prediction. However, the accuracy is still relatively high, and the accuracy increases rapidly as a few more bounding boxes are added.

Deployment using Tensorflow Lite

To increase the performance of the model, tests were conducted using TensorFlow Lite to deploy the model. TensorFlow Lite is a deep learning framework for on-device inference, and is designed to improve inference speed without sacrificing accuracy. To evaluate this, tests were conducted to check how big of an inference speed improvement was possible, and if that would affect the accuracy. The models were converted to TensorFlow Lite models using built in functions, and run through the same tests as the TensorFlow models.

Units	1	5	10	30	100	300	500
Time	0.11 ms	0.11 ms	0.12 ms	0.13 ms	0.17 ms	0.56 ms	1.23 ms
IoU	39.5 %	94.3 %	94.3 %	94.2 %	93.8 %	94.1 %	93.9 %
Dist.	117.2px	4.4 px	4.3 px	4.6 px	5.0 px	4.6 px	4.7 px

Table 5.6: Results from different LSTM network sizes using TensorFlow Lite

Table 5.6 shows that the inference speed was an order of magnitude lower using TensorFlow Lite for all but the largest models, however there were no detectable difference in the accuracy. The speed increase is significant for real-time performance, thus TensorFlow Lite was used to deploy the models.

Results

Finally, tests were conducted on the test videos to evaluate actual performance in the real application. Affinity cost function was the euclidean distance between centroids, and association algorithm was the hungarian algorithm.

Test Video	**Fish Count**	**False Posities**	**False Negatives**	**Identity Switches**	**Fragmented Tracks**	**Total Tracks**
Video One	385/385 (100.0%)	2 (0.5%)	2 (0.5%)	12	18	487
Video Two	1118/1125 (99.4%)	13 (1.2%)	20 (1.8%)	30	37	1795
Combined	1503/1510 (99.5%)	15 (1.0%)	21(1.4%)	42	55	2282

Table 5.7: Test results with LSTM model for motion prediction

Table 5.7 shows that both the number of identity switches and fragmented tracks were significantly lower compared to the results from using kalman filters (5.3). The number of false negatives is slightly higher using an LSTM model compared to kalman filters, however the number of false positives, identity switches and fragmented tracks were all significantly lower. The total number of tracks was also lower, which in addition to the other metrics show that the LSTM model was more stable than the kalman filters for motion prediction. Especially for fish inspection, the lower numbers of identity switching and fragmented tracks is significant, as it makes methods that rely on correctly tracked fish more reliable.

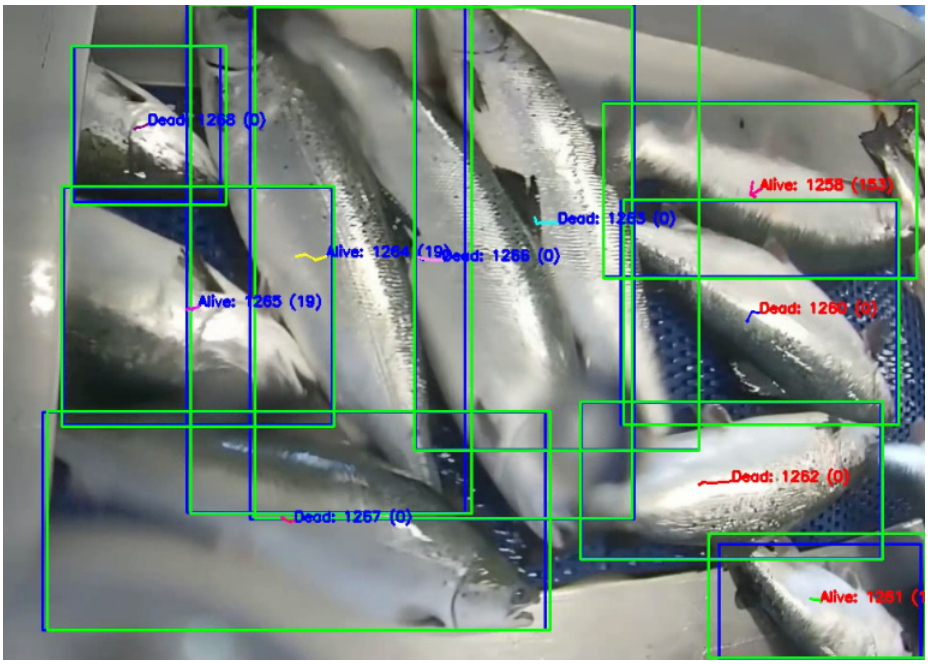


Figure 5.8: Example of fish tracked successfully in crowded frame

Figure 5.8 shows an example of fish tracked successfully in a crowded scene. The green bounding boxes are the predicted bounding boxes, while the blue are the detected bounding boxes.

5.2.3 Kalman Filter vs LSTM Efficiency Comparison

The results from the previous sections showed that using an LSTM model for motion prediction performed better than the kalman filter, however that did not take into account how efficiently the different methods ran. An experiment was conducted to time the two methods, and the results shown in table 5.8 clearly shows that kalman filters are more efficient. This seemed to be especially clear in frames with a lot of fish. However, the overall difference was not significant enough to discard the benefits of LSTM motion prediction, thus that was used going forward.

Kalman filters performed equally as well as LSTM models in scenes with few fish, thus a decision was made to use kalman filters for the damage tracks. There are very rarely multiple instances of damage at the same time, in any case not enough for the kalman filters to perform worse than the LSTM model.

Test Video	Fish / Minute	Kalman filter	LSTM	Difference
Video 1	32.1 Fish	2.9 ms / frame	3.8 ms / frame	+31 %
Video 2	56.3 Fish	6.5 ms / frame	9.1 ms / frame	+40 %

Table 5.8: Comparison of time per frame between tracking with kalman filters and LSTM

5.3 Damage Counting

For counting damage, the first method tested was to use the same method as for fish counting, to count instances of damage as the tracks passed the counting line. However, from the initial tests (table 5.9) it became clear that there were some issues with this. For one, if a fish is moving and only damaged on one side, the damage will be blocked from the camera if the fish flips. Secondly, if the damage is small or occluded by other fish as the damaged fish passes the counting line, the damage might not be detected and counted as it passes the line.

To prevent partial damage tracks from not being counted, a method was devised where damage was associated to a fish track, such that no matter if the damage track disappeared, the damage would be counted as long as the fish track was intact as it crossed the counting line.

The idea behind the method was to calculate a cost between each damage track and each fish track, and if the same fish was the closest for a given amount of frames, the damage was associated with that fish. Two different cost functions were tested: the euclidean distance between damage and fish centroids, and a combination of the euclidean distance and the IoU of the bounding boxes.

Association Method	Total Count (out of 43)	True Positives	False Positives	Missed Damages	Correct Assignment	False Assignment
None	31 (72%)	30 (70%)	1 (2%)	12 (28%)	N/A	N/A
Dist	41 (95%)	38 (88%)	3 (7%)	5 (12%)	33 (80%)	8 (20%)
Dist*IoU	41 (95%)	38 (88%)	3 (7%)	5 (12%)	36 (88%)	5 (12%)

Table 5.9: Damage counting test results from test video 2 with different association methods

Table 5.9 shows that by associating damages to fish, the accuracy was improved significantly. The number of missed damages was lower, in large part down to partial damage tracks also being counted. The tests also showed that a combination of euclidean distance and IoU matched damage better to the correct fish. This was expected, as the damage isn't necessarily closest to the center of the damaged fish. Figure 5.9 shows an example where the damage is closest to another fish. In this specific example the bounding box of the damage is also within the bounding box of this fish, thus the damage was assigned to the incorrect fish. Conversly, 5.10 shows an example where the damage is correctly assigned. The damage track is closer to another fish in distance, however the bounding boxes do not overlap, thus the damage is assigned to the correct fish.

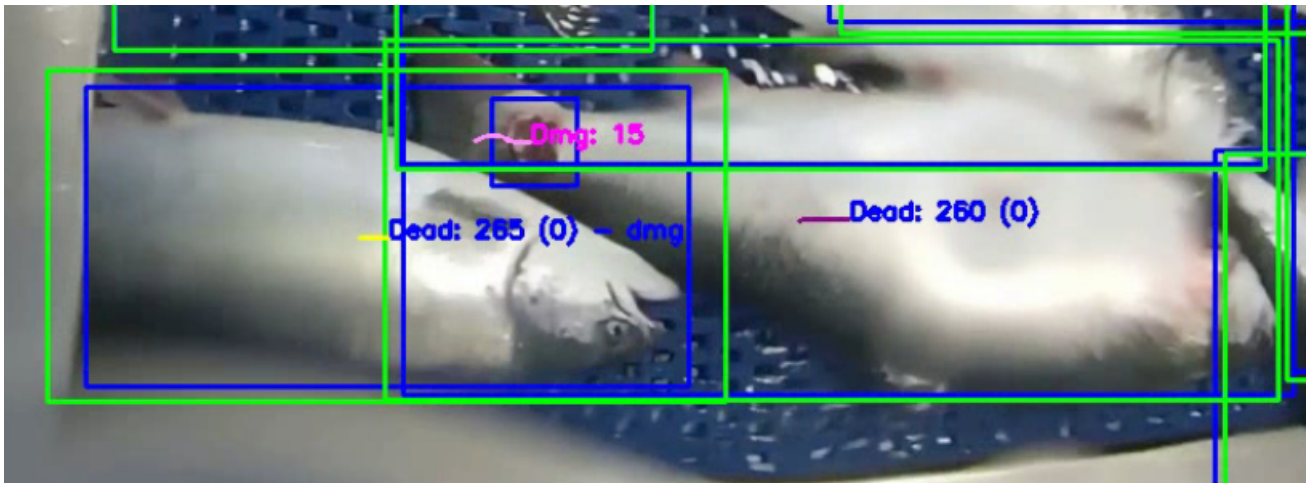


Figure 5.9: Example of damage being associated with the incorrect fish. The damage track is closer to fish "265" (left) and the bounding boxes overlap, thus the damage is incorrectly assigned to fish "265"

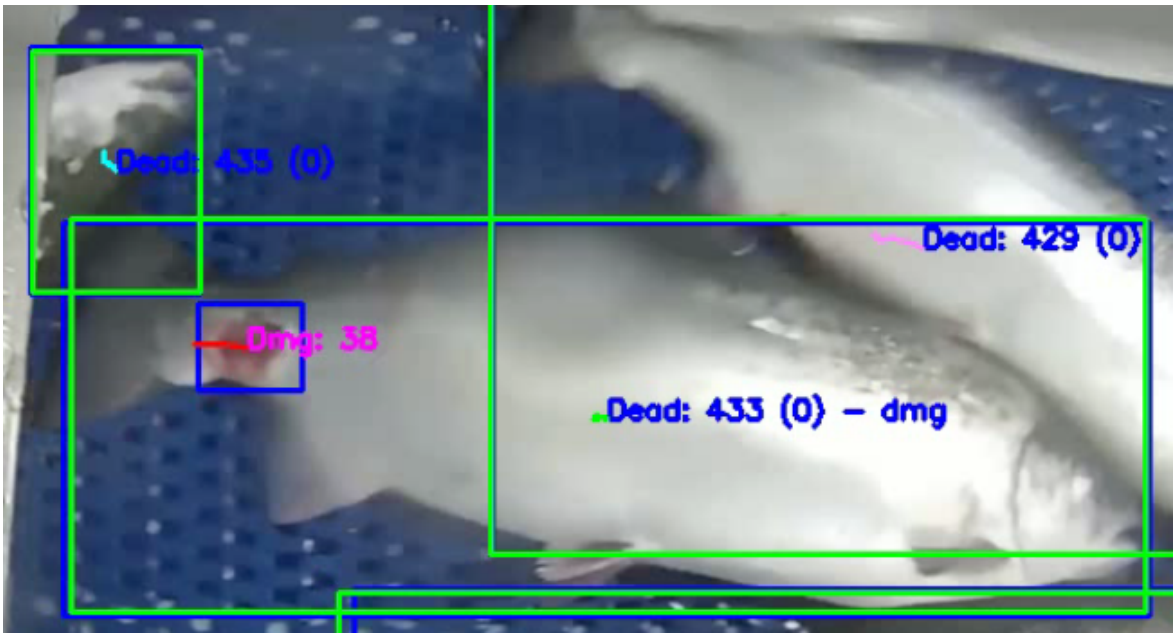


Figure 5.10: Example of damage being associated with the correct fish through a combination of euclidean distance and distance. Fish "435" is closer in distance, however the damage bounding box does not overlap, thus the damage is correctly associated with fish "433".

5.4 Classifying Dead / Alive Fish

In addition to damage detection, the other inspection method explored was a way to classify if a fish was dead or alive. The assumption made for this problem was that fish moving by themselves are alive, and fish lying still on the conveyor belt were dead. Thus the method experimented with for dead / alive classification, was to detect if a fish was moving or not, and assuming that if a fish was moving it was alive, otherwise it was dead. Figure 5.11 shows an example of a fish moving by itself.

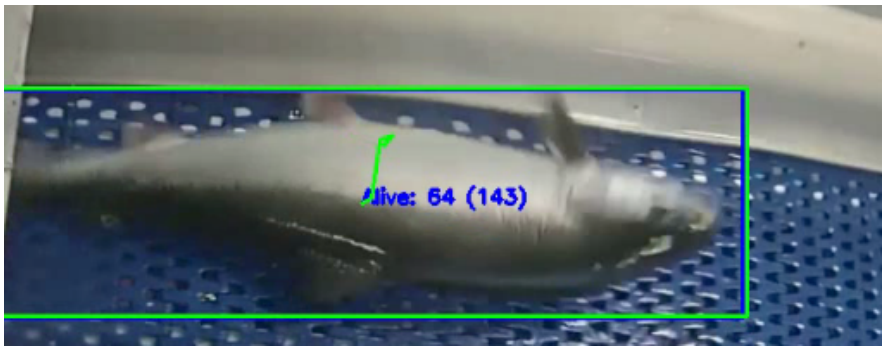
(a) Frame 1

(b) Frame 21

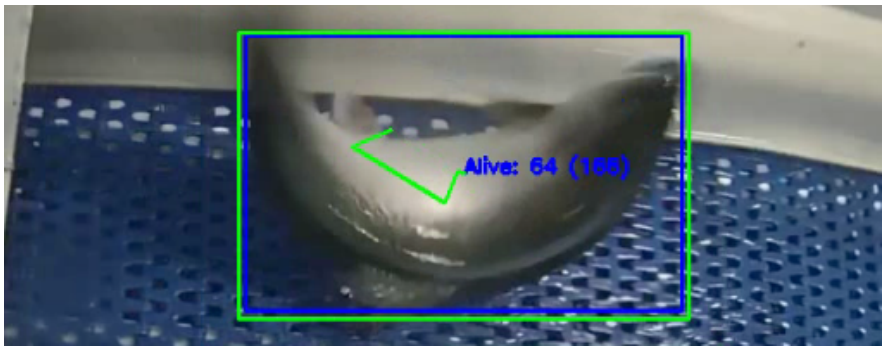
(c) Frame 31

Figure 5.11: Example of a fish moving by itself

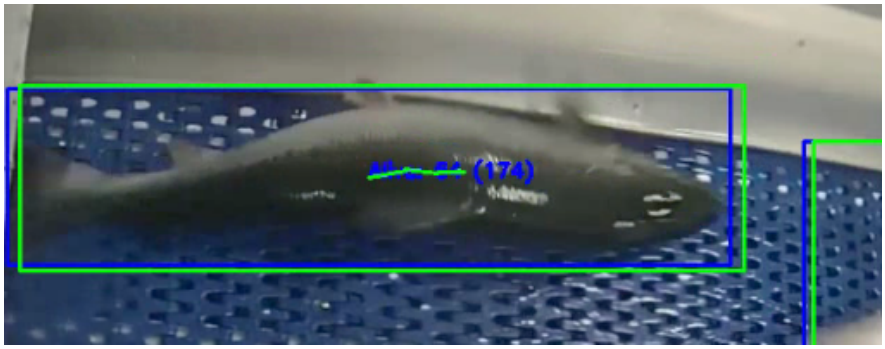
The proposed method for detecting movement utilized a similar network to the one used for motion prediction, however instead of predicting the next bounding box, an LSTM network was used to classify if a sequence of inputs came from a moving or still fish. Figure 5.12 shows an example of the bounding box and centroid of a fish changing from frame to frame as the fish moved by itself. In the following sections the process and results of developing models to capture this movement are explained.



(a) Frame 1



(b) Frame 21



(c) Frame 31

Figure 5.12: Example showing the change in bounding box and centroid of a moving fish

5.4.1 Dataset Variations

The dataset used for this part of the thesis was the same dataset used for the motion prediction training, however there were quite a few different possible ways to represent the data. The motion prediction used a sequence of bounding boxes, however for classifying the sequences based on movement it was theorized that the difference from frame to frame might perform better. There were other representations of interest as well, thus a set of representations were created and tested. Each representation has an abbreviation, which is used in tables to indicate

which representations were used.

- **A - Aspect ratio:** The ratio between width and hight of bounding box (w/h)
- **Ad - Aspect ratio difference:** Change in aspect ratio from frame to frame
- **B - Bounding box:** The normalized x and y coordinates of top-left corner and normalized hight and with
- **Bd - Bounding box difference:** Change in bounding box from frame to frame
- **C - Centroid:** The normalized x and y coordinates of the object center
- **Cd - Centroid difference:** Change in centroid from frame to frame

5.4.2 Network Architecture

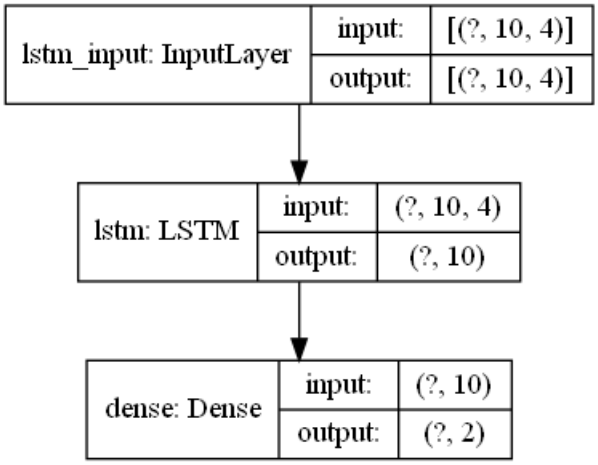


Figure 5.13: Network model graph (10 LSTM units) for classifying dead / alive fish

The network architecture experimented with was similar to the architecture used for motion prediction, with the main difference being the fully connected layer. The final dense layer has 2 outputs, which are the confidence values that an input sequence belong to either the dead or alive class. The activation function chosen for the network was the softmax activation function, which gives a probability distribution summing to 1 that the input belongs to the different classes. The loss function chosen was sparse categorical crossentropy. For the experiments,

tests were done on networks with 10, 50 and 100 LSTM units. Initial results indicated a degree of overfitting, thus a recurrent dropout of 0.5 was used for training the models.

5.4.3 Training Results

The first set of experiements looked at each of the dataset variations in turn, each on different network sizes. Tables 5.10, 5.11 and 5.12 show the accuracy of the models when tested on the testing split, including total accuracy as well as separate accuracy measurements for dead and alive fish. The results were calculated both with a 50% and 90% confidence value, indicating how certain the model has to be a sequence belongs to the moving category for it to count as alive.

The results indicate that using the difference from frame to frame performed significantly better for the bounding box and centroid data, however the reverse seemed to be true for aspect ratio. In general, bounding box difference and centroid difference seemed to have the best accuracy, followed by aspect ratio. The data also showed that with the exception of the bounding box representation, classification was more accurate for dead fish on average. However, because there are usually more dead than alive fish, this could still lead to more false positives on actual test videos.

	Total Accuracy		**Alive Accuracy**		**Dead Accuracy**	
Dataset Variation	50% Confidence	90% Confidence	50% Confidence	90% Confidence	50% Confidence	90% Confidence
A - Aspect Ratio	79.4 %	79.8 %	58.2 %	9.2 %	85.3 %	99.5 %
Ad - Aspect Ratio Diff	77.5 %	80.0 %	52.1 %	10.7 %	84.6 %	**99.4 %**
B - Bounding Box	54.8 %	66.2 %	69.2 %	21.2 %	50.8 %	78.9 %
Bd - Bounding Box Diff	**87.2 %**	**86.5 %**	71.5 %	46.4 %	**91.6 %**	97.8 %
C - Centroid	64.0 %	78.7 %	69.6 %	8.8 %	64.7 %	96.0 %
Cd - Centroid Diff	86.4 %	**86.5 %**	**71.6 %**	**44.8 %**	90.5 %	98.4 %

Table 5.10: Test results using 10 LSTM units

	Total Accuracy		**Alive Accuracy**		**Dead Accuracy**	
Dataset Variation	50% Confidence	90% Confidence	50% Confidence	90% Confidence	50% Confidence	90% Confidence
A - Aspect Ratio	80.1 %	81.7 %	51.3 %	21.7 %	88.9 %	98.5 %
Ad - Aspect Ratio Diff	78.3 %	80.6 %	52.1 %	12.1 %	85.6 %	**99.1 %**
B - Bounding Box	52.7 %	58.0 %	70.1 %	**59.5 %**	47.6 %	57.4 %
Bd - Bounding Box Diff	87.7 %	**87.6 %**	71.9 %	53.5 %	92.2 %	97.2 %
C - Centroid	57.0 %	72.5 %	69.6 %	15.8 %	53.5 %	88.4 %
Cd - Centroid Diff	**88.8 %**	86.7 %	**76.6 %**	45.1 %	**92.3 %**	98.6 %

Table 5.11: Test results using 50 LSTM units

	Total Accuracy		**Alive Accuracy**		**Dead Accuracy**	
Dataset Variation	50% Confidence	90% Confidence	50% Confidence	90% Confidence	50% Confidence	90% Confidence
A - Aspect Ratio	77.9 %	81.9 %	56.9 %	25.0 %	83.8 %	97.8 %
Ad - Aspect Ratio Diff	78.6 %	80.6 %	51.6 %	16.4 %	86.0 %	**98.5 %**
B - Bounding Box	50.4 %	54.7 %	**74.7 %**	**66.1 %**	43.3 %	51.2 %
Bd - Bounding Box Diff	87.2 %	**87.8 %**	74.0 %	57.2 %	90.9 %	96.4 %
C - Centroid	66.9 %	75.4	74.0 %	25.3 %	64.9 %	89.5 %
Cd - Centroid Diff	**88.8 %**	87.7 %	73.5 %	45.1 %	**92.3 %**	**98.5 %**

Table 5.12: Test results using 100 LSTM units

With the baseline results, the next set of experiments revolved around combining the metrics into the same networks, to test if a combination allowed the network to better understand the data. As described above the best accuracy came from bounding box difference and centroid distance, thus a combination of these were tested first. In addition, tests were performed including aspect ratio as well.

		Total Accuracy		**Alive Accuracy**		**Dead Accuracy**	
LSTM Units	**Dataset Variations**	50% Confidence	90% Confidence	50% Confidence	90% Confidence	50% Confidence	90% Confidence
10	Bd + Cd	**89.2 %**	87.6 %	77.4 %	51.3 %	**92.5 %**	**97.8 %**
50	Bd + Cd	88.5 %	87.7 %	74.8 %	57.7 %	92.3 %	96.1 %
100	Bd + Cd	87.3 %	88.0 %	73.2 %	61.1 %	91.2 %	95.6 %
10	A + Bd + Cd	88.4 %	87.6 %	75.4 %	52.4 %	92.1 %	97.4 %
50	A + Bd + Cd	88.0 %	**88.3 %**	**79.7 %**	62.6 %	90.3 %	95.5 %
100	A + Bd + Cd	87.3 %	88.1 %	76.0 %	**66.1 %**	90.5 %	94.2 %
10	Ad + Bd + Cd	88.3 %	87.5 %	73.5 %	51.6 %	**92.5 %**	97.6 %
50	Ad + Bd + Cd	88.1 %	87.9 %	75.5 %	57.4 %	91.6 %	96.4 %
100	Ad + Bd + Cd	88.0 %	**88.3 %**	75.5 %	63.6 %	91.4 %	95.2 %

Table 5.13: Test results using a combination of dataset variations

Results from table 5.13 show that combining data representations improved the accuracy of the model, especially with higher confidence values and for classifying alive fish.

5.4.4 Evaluation on test video

With a model designed and trained, the next step was to deploy and evaluate the model in the complete solution on the test videos. The model was deployed using TensorFlow Lite, and integrated into the object tracking part of the MOT algorithm as described in section 4.1.2. The model chosen for testing was the $A + Bd + Cd$ model with 100 LSTM units, as it had the highest accuracy for classifying alive fish at high confidence variables, while still having acceptable accuracy classifying dead fish.

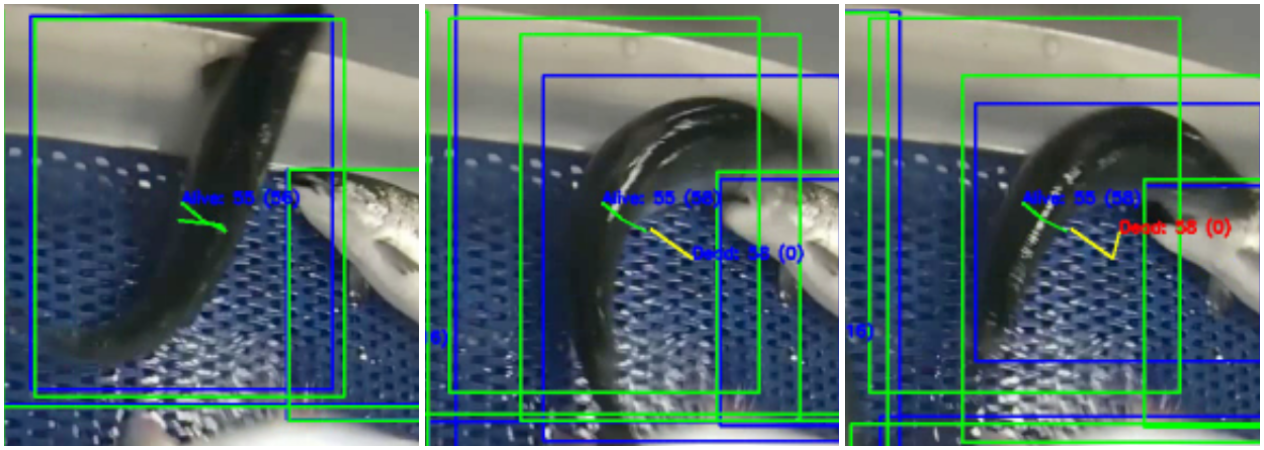
As described, the model has an input of a fixed length sequence of previous bounding boxes. The classification is run every frame after the required number of bounding boxes have been tracked, and if classified as moving an internal counter within the fish track is incremented. To prevent a few incorrect classifications from marking a fish as alive, an experiment was conducted to find an appropriate value for number of times a fish should be marked as moving before it was classified as alive (as the track passes the counting line).

Moving Count	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Alive Accuracy	73.4 %	72.0 %	69.2 %	67.8 %	64.4 %	62.3 %	60.9 %
Alive Accuracy w/o Frag.	83.5 %	71.9 %	78.7 %	77.2 %	73.2 %	70.9 %	69.3 %
Dead Accuracy	88.4 %	89.5 %	90.7 %	92.1 %	93.1 %	93.5 %	93.5 %
Total Count Score	106.2 %	101.7 %	95.5 %	90.0 %	83.7 %	80.7 %	77.9 %
Total Score w/o Frag.	120.9 %	115.7 %	108.7 %	102.4 %	95.3 %	91.3 %	88.6 %

Table 5.14: Test results on test video two, for different moving count target values (fish classified as alive if above). w/o frag (fragments) refers to excluding tracks that were fragmented and thus losing the move count before the counting line, but would have been classified correctly if not for that

Table 5.14 shows the results with different values for the move count target to classify the fish as alive. One issue that became apparent during manual inspection was that track fragmentation was a semi-frequent occurrence for moving fish (35 times, 12.1% of moving fish), and this

caused issues for classifying it as alive. After fragmentation a new track is established for the fish, and often it will pass the counting line before it can be classified as alive again.



(a) Frame 1 (b) Frame 4 (c) Frame 6

Figure 5.14: Example of fragmentation causing fish to lose classification information. In frame two the distance between prediction and detection is too large, thus a new track is started. For the following frames, this new track is tracked, while the old one is eventually removed without being counted.

Figure 5.14 gives an example of a fish losing the classification information through fragmentation. Before the track was lost the move count was 56 for the track, well above the tracking limit. In fact, all the alive fish not classified correctly due to fragmentation would have been classified as alive had the tracks not been broken. Therefore, results excluding these are also included in 5.14, as it is not directly linked to the method for classifying the fish and it is an area for potential improvement in regards to the tracking algorithm.

Another challenge with the proposed method is that fish interactions can cause incorrect classifications. For example, a moving fish might temporarily occlude or otherwise change the bounding box and centroid of a nearby dead fish, which could lead to incorrect classification as moving. As can be seen in the test results, the number of dead fish classified as alive was close to the number of alive fish incorrectly classified as dead, thus the total score was close to and even beyond 100%. However, independently the accuracy was closer to 70% and 80% for correctly classified alive and dead fish, respectively.

From the results, somewhere between 10 and 12 seemed to be the optimal count target for this test video. However, whether this works as a general limit or not for other videos is not guaranteed, and further testing and changes are required for a more reliable classification method, see sections 6.3 and 6.5 for more on this.

5.5 Test Video Results

Finally, with all the fish counting and inspection experiments completed, it was time to get the final results of the complete solution on the two test videos.

5.5.1 Fish Counting

Test Video	Length (m)	Fish	Total Count	True Count	False Count	Frame Time	FPS
Video One	12:00	385	385 (100%)	383 (99.5%)	2 (0.5%)	18.9 ms	52.9
Video Two	20:00	1125	1118 (99.4%)	1107 (98.4%)	11 (1.0%)	24.6 ms	40.7
Combined	32:00	1510	1503 (99.5%)	1490 (98.7%)	13 (0.8%)	22.5 ms	44.4

Table 5.15: Final fish counting results

Table 5.15 shows that the solution has a very high accuracy, correctly counting over 98% of fish, and through a few false positives bringing the total score to above 99%. The inference time depends on how many fish there are, but around 20-25 ms per frame with the current hardware as explained in section 3.5. This equates to around 40-45 frames per second, which is lower than the 60 frames per second of the videos, though with improved hardware and some more focus on optimization, it should be possible to get the frame rate to above 60. See section 6.4 for more discussion on this.

5.5.2 Fish Inspection

Table 5.16 shows that damage detection performs relatively well, with a true positive score close to 89%. The total score including false positives raises the number almost up to 97%.

Test Video	Length (m)	Fish	Damage Count	True Damage	Alive Count	True Alive
Video One	12:00	385	20/20 (100%)	18/20 (90%)	80/96 (83.3%)	68/96 (70.8%)
Video Two	20:00	1125	41/43 (95.3%)	38/43 (88.4%)	310/289 (107.3%)	200/289 (69.2%)
Combined	32:00	1510	61/63 (96.8%)	56/63 (88.9%)	390/385 (101.3%)	268/385 (69.6%)

Table 5.16: Final fish inspection results, with total counts (including false positives) as well as true positive counts

The number of fish that are classified alive is less stable, with below 70% accurately counted using 10 as the movement target for classification. The total number is raised by false positives, and the numbers happen to match up well with the number of alive fish not detected. However, this is not necessarily repeatable in general, and would not work well if the method was combined with another method in future work. Still, the results show a proof-of-concept that can be worked on and improved in future works.

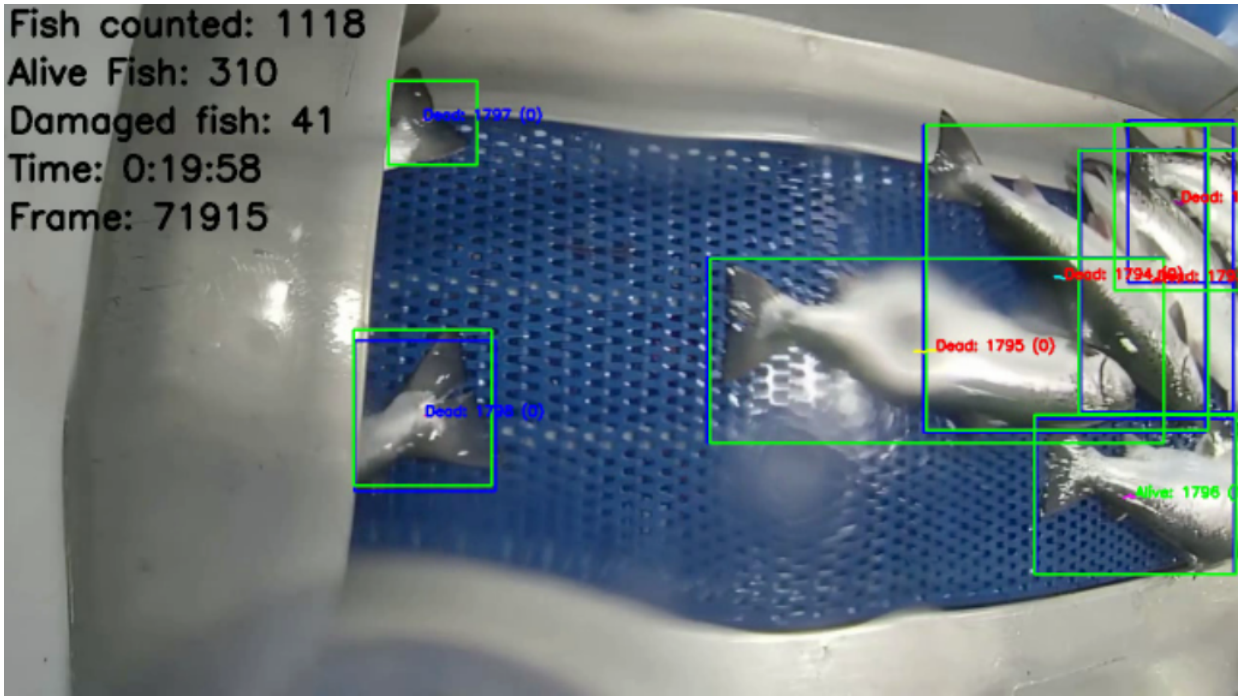


Figure 5.15: Screenshot of test video two after final fish had been counted

Chapter 6

Discussion

In this chapter the developed solution and test results will be discussed. In that regard it might be helpful repeating the goals and research question set out in the introduction.

Goals

- **G1:** *Gather video material and create datasets and testing videos to develop and test solutions proposed in this thesis.*
- **G2:** *Develop solutions for accurate real-time fish counting from video of fish on a conveyor belt.*
- **G3:** *Expand the fish counting solution to include fish inspection and classification.*

Research Questions

- **RQ1:** *How to use multiple object tracking algorithm to perform counting of fish on video?*
- **RQ2:** *Which methods, algorithms, and techniques are most suited for a real-time implementation of multiple object tracking for fish counting?*
- **RQ3:** *How to extend the same system used for fish counting to also perform fish inspection?*
- **RQ4:** *Specifically, how to extend the system so that it can be used for damage detection, and to classify fish as dead or alive?*

6.1 Dataset Creation (G1)

The first goal of this thesis was to gather data and create datasets for both training and testing of the various parts of the solution, as well as the full solution. The first dataset created was a labelled set of images consisting of fish and instances of damage. The yolov4 object detection training results using the dataset were very good, and the accuracy of the object detection model and of the fish counting as a whole prove that the dataset performed well.

The dataset created for both motion prediction and classifying fish as dead or alive also largely performed well. The motion prediction models trained on the dataset showed very good results for predictions, as shown in both testing results and the improvement in fish counting using the model over kalman filters. The classification model didn't perform as well as the motion prediction one, which is most likely partly down to the dataset. One of the main issues during classification was interaction between fish, which is not addressed in the dataset. Disregarding that, though, the performance was still promising, accurately classifying 70-80% of the fish.

6.2 Fish Counting (G2)

The second goal of this thesis was to develop solutions for accurate real-time fish counting. The following sections discuss the research questions behind this goal (RQ1 and RQ2), showing that this goal has successfully been achieved through the implementation of the multiple object tracking algorithm and deep learning methods.

6.2.1 Multiple Object Tracking

The first research question of the thesis (RQ1) was to evaluate if the multiple object tracking (MOT) algorithm could be used for fish counting and inspection. The thesis shows that MOT is a suitable method to track fish for fish counting, outperforming the accuracy rating of dedicated fish counting equipment, while at the same time allowing for visual fish inspection. As mentioned in 1.1 current solutions are rated at 97% accuracy, and as shown from test results in 5.15 the developed solution accurately counts over 98% fish, and including double counting

the total count has an overall accuracy of 99%. This thesis shows that by using the MOT algorithm the central issue for fish counting from video, namely tracking and only counting each fish once, can be solved. Tracking also allows for inspection over time, which is demonstrated in the damage detection and dead / alive classification methods developed in this thesis.

Research question two (RQ2) of the thesis asked which methods were most suited for implementing real-time multiple object tracking for fish counting. The two most important methods for the MOT algorithm were the object detection and motion prediction methods. Both of these were implemented using deep learning networks, and as discussed in the following sections the implementation proved very successful for fish counting.

Object Detection

Yolov4 was chosen as it is a state-of-the-art object detection model, promising both accurate and fast object detection. As demonstrated in 5.1.1, the custom-trained yolov4 model created for this project has a high degree of accuracy (98% mAP), correctly detecting and classifying the vast majority of objects in the frames 5.1.1. Even in crowded scenes most objects are clearly classified, and bounding boxes are accurately bounding objects. The object detection inference time is low, above 60 FPS on the test machine used for this thesis, allowing for real-time detection with a powerful computer.

Motion Prediction: Kalman Filter vs LSTM

One of the main areas of research in this thesis were the motion prediction methods, with the two main methods tested being the use of kalman filters and recurrent neural network models (LSTM). The results detailed in sections 5.2.1 and 5.2.2 showed that both methods worked well, however there were a few more issues with the kalman filters. Both methods resulted in some identity switches and fragmented tracks, which in turn led to some incorrect data, however the LSTM method performed significantly better in this regard.

Kalman filters are computationally cheap and as shown in 5.2.3 this did result in higher speeds using kalman filters. However, through optimization such as deployment using TensorFlow Lite, the LSTM model's performance was close to that of the kalman filters, and the prediction accuracy more than made up for the difference.

Therefore, the best results over all came from using the LSTM method for motion prediction, at least for fish tracks. Both the kalman filters and LSTM method had next to no issues in calm scenes with few objects, thus it was decided to use the kalman filters for damage tracking, as there is very rarely more than one or two instances of damage at the same time.

6.3 Fish Inspection and Classification (G3)

Research question three and four asked if the same system used for fish counting can be extended to perform fish inspection and classification as well. This thesis has laid down the groundwork for this, and through damage inspection and classifying fish as dead or alive, proving that with some more work, the same system can indeed be used for fish inspection as well.

The results for damage detection (5.3) are promising for what can be a challenging problem. Detecting damage on the fish side laying down on the conveyor belt is not possible with the current setup (with video from above), however as shown the current method is able to detect damage visible to the camera. Using the same MOT algorithm used for detecting fish, the damage is tracked from frame to frame thus only counted once. Additionally, by associating damage to fish tracks, damage that is only partly visible (moving fish changing side pointing to camera) is also counted. There are still some issues that need to be addressed in future works, the main one being that fragmented tracks can cause lost damage detections.

The results for classifying if fish are dead or alive (5.4) were not as accurate as the damage detection results, but as a proof of concept this thesis shows that it is possible to detect fish movement using recurrent neural networks. There are two main issues that need to be addressed in future works. Firstly, improving tracking is required to minimize track fragmentation, which can cause alive fish to be classified as dead. Secondly, alive fish temporarily occluding or otherwise changing the bounding boxes of dead fish can cause the dead fish to be classified as alive. In the current implementation with the test videos used, these two errors roughly balance each other out, resulting in a total score close to the target. However, that does not necessarily generalize well, and without improvements it will be hard to combine this classification with other inspection methods in future works.

The two developed inspection methods offer proof-of-concepts that show that it is possible

to expand the fish counting solution to also include fish inspection and classification. There are also good possibilities for adding more methods in future works, such as estimating size and weight, classifying fish species, and more.

6.4 Hardware Requirements

As described in section 3.5, the tests were performed using an NVIDIA RTX 2080Ti graphics card, achieving between 40-45 frames per second (5.15). This is sufficient to run the solution in real-time, though with a newer graphics card the performance should be able to match the 60 frames per second of the camera. Table 6.1 shows that even the low-end RTX 30-Series graphics cards have more CUDA cores and higher clock speeds, as well as a newer CUDA architecture, which will result in improved performance. Thus, for real-time performance a computer equipped with an RTX 30-Series graphics card or higher is recommended.

Graphics Card	**RTX 2080Ti**	**RTX 3070**	**RTX 3070Ti**	**RTX 3080**	**RTX 3080Ti**	**RTX3090**
CUDA Cores	4352	5888	6144	8704	10340	10496
Base Clock (GHz)	1.35	1.50	1.58	1.44	1.37	1.40
Boost Clock (GHz)	1.55	1.77	1.73	1.71	1.67	1.70
Memory	11 GB GDDR6	8 GB GDDR6X	8 GB GDDR6X	10 GB GDDR6X	12 GB GDDR6X	24 GB GDDR6X
Architecture	Turing	Ampere	Ampere	Ampere	Ampere	Ampere

Table 6.1: RTX 2080Ti vs RTX 30-Series. Source: nvidia.com

6.5 Future Work

6.5.1 Fish Counting

For future work there are a few main areas to look at. First of all, as good as the counting results are, there are still improvements to be made for the MOT algorithm to tackle identity switches and fragmented tracks. This is especially significant for inspection methods that rely on tracks staying intact, such as the two methods developed in this thesis.

A possible area to look into in this regard is the use of recurrent neural networks for the affinity and association stages, as proposed in [14]. This could potentially lower the number of

identity switches and fragmented tracks, resulting in better performance, especially for inspection methods relying on tracking fish.

Another possible area to look at is creating a larger dataset for the Yolov4 model, and potentially tweak some of the variables for object detection. The current detection model works well, but there can still be improvements. Additionally, in the future there will undoubtedly be developed more accurate and efficient object detection models, so following the state-of-the-art could lead to improvements in this regard for the future. There is already a Yolov5, however this is not directly an upgrade, but a separate project not directly related to Yolov4 [11]. Still, future works could include testing Yolov5, as well as other object detection models.

For an improved dataset it is also possible to improve the camera angle further. Currently it does point down at the conveyor belt, but not directly down and the conveyor belt doesn't take up the entire field of view, leaving some dead space not being used. Additionally, in some of the test videos there were some water on the lens, which caused some incorrect detections.

6.5.2 Fish Inspection and Classification

Probably the main area for improvement and future work is within fish inspection and classification. This thesis offers a proof-of-concept for two inspection methods, but there is still work to be done to make them more robust. Especially the method for classifying dead and alive fish, where the current method classifies around 70% and 80% of alive and dead fish correctly.

One possible way to improve this could be to look at the full sequence of bounding boxes as a whole, classifying the whole sequence once, instead of using a fixed sequence length every frame. This would improve the efficiency of the algorithm, however if the classification would be better has to be tested.

Another possibility could be to look at all the tracks together, instead of looking at each individually. One of the issues with the current method is that dead fish that are pushed by alive fish are often detected as moving. Similarly, a moving fish often ends up partially occluding parts of other fish, which causes their bounding boxes to rapidly change. This is then detected by the network as movement. So again, one possible way to minimize this issue could be to look at all fish in each image at the same time, or in some other way include information about surround-

ing fish during classification.

The damage detection method shows promising results, however there are still work to be done to make it more reliable. The main issue to address in future work is the loss of information as a result of fragmented tracks. This can be improved by better detection and tracking as discussed previously, however alternative methods for counting and reporting detected damages could also be explored. Accurately detecting and counting damage can be really valuable. It can help detect faults with the fish processing and transportation systems, and it can be used by veterinarians to inspect the fish. Thus, in addition to improving the damage detection, future work could also be to look at methods for reporting the damage, for example through automated reports with screenshots/video attached of the damages.

In addition to improving current solution, there is also a lot of future work in implementing other inspection methods, such as estimating size or weight of fish, classifying different species of fish, and so on. The choices of methods and how to implement them will depend on what is desired, but there is definitely room for new additions to the current methods.

6.5.3 Hardware and Interface

Other future work includes a more detailed look at the hardware, including which camera to use and hardware for performing the processing. When determining this, future work should also look at additional optimization methods to further improve the time to process each frame.

Another important area for future work is the creation or implementation of a proper interface, such that a user or other equipment can interact with the fish counting, inspection and classification results. For example, can existing solutions like OPC UA[1] be used, or is a custom interface more appropriate?

[1] OPC UA: https://opcfoundation.org/about/opc-technologies/opc-ua/ (As of June 2021)

Chapter 7

Conclusion

This thesis has shown that by using multiple object tracking and state-of-the-art deep learning models, the developed solution for visual real-time fish counting can deliver results exceeding existing fish counting equipment. The proposed solution accurately counts above 98% of fish in test videos with over 1500 fish, and if false positives are included the total score is above 99%. The solution runs at 40-45 frames per second on a powerful last-gen graphics card, which is sufficient for real-time performance, and with upgrades to the hardware it is easily possible to improve the performance to get higher frame rates.

The accuracy and performance is in large part down to implementation of state-of-the-art deep learning models. The Yolov4 object detection model used can accurately detect fish in video frames, with real-time inference speeds. The implementation of recurrent neural networks (LSTM) for motion prediction in the MOT algorithm saw significant improvements in terms of stability of object tracking compared to classic methods like the kalman filter, with only a minor increase in inference times.

This thesis has also laid the groundwork for expanding the solution to include fish inspection. The developed damage detection algorithm accurately detects close to 90% of visible damage, and the developed algorithm for classifying fish as dead or alive accurately classifies around 70% of alive fish, and 80% of dead fish.

Both of these methods work as proof-of-concepts for fish inspection using the detection and tracking methods developed for fish counting. They both require further development to

become more stable, however they show that there is a huge potential in using deep learning for visual-based fish inspection.

Bibliography

[1] Atlassian. What is agile? URL https://www.atlassian.com/agile.

[2] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[3] Bruff, D. The assignment problem and the hungarian method. URL https://web.archive.org/web/20120105112913/http://www.math.harvard.edu/archive/20_spring_05/handouts/assignment_overheads.pdf.

[4] François Chollet. *Deep Learning with Python*. Manning, November 2017. ISBN 9781617294433.

[5] Gioele Ciaparrone, Francisco Luque Sánchez, Siham Tabik, Luigi Troiano, Roberto Tagliaferri, and Francisco Herrera. Deep learning in video multi-object tracking: A survey. *Neurocomputing*, 381:61–88, Mar 2020. ISSN 0925-2312. doi: 10.1016/j.neucom.2019.11.023. URL http://dx.doi.org/10.1016/j.neucom.2019.11.023.

[6] Food and Agriculture Organization of the United Nations. The state of world fisheries and aquaculture 2020. URL http://www.fao.org/state-of-fisheries-aquaculture.

[7] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks.

[8] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN 9780262035613. URL https://books.google.co.in/books?id=Np9SDQAAQBAJ.

[9] Furkan Gulsen. Detecting and counting objects with opencv. URL https://medium.com/analytics-vidhya/detecting-and-counting-objects-with-opencv-b0f59bc1e111.

[10] Shaunak Halbe. Object detection and instance segmentation: A detailed overview. URL https://medium.com/swlh/object-detection-and-instance-segmentation-a-detailed-overview-94ca109274f2.

[11] Ildar Idrisov. Yolov4 vs yolov5. URL https://medium.com/deelvin-machine-learning/yolov4-vs-yolov5-db1e0ac7962b.

[12] Youngjoo Kim and Hyochoong Bang. Introduction to kalman filter and its applications. In Felix Govaers, editor, *Introduction and Implementations of the Kalman Filter*, chapter 2. IntechOpen, Rijeka, 2019. doi: 10.5772/intechopen.80600. URL https://doi.org/10.5772/intechopen.80600.

[13] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, Xiaowei Zhao, and Tae-Kyun Kim. Multiple object tracking: A literature review, 2017.

[14] Anton Milan, S. Hamid Rezatofighi, Anthony Dick, Ian Reid, and Konrad Schindler. Online multi-target tracking using recurrent neural networks, 2017. URL https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14184.

[15] Miljødirektoratet. Fiskeoppdrett. URL https://miljostatus.miljodirektoratet.no/Fiskeoppdrett/.

[16] Bård Misund and Ragnar Tveterås. Et blått taktskifte. URL https://sjomatnorge.no/wp-content/uploads/2019/04/Blått-Taktskifte-Investeringsbehov.pdf.

[17] OpenCV. Deep neural network module. URL https://docs.opencv.org/4.4.0/d6/d0f/group__dnn.html.

[18] Michael Phi. Illustrated guide to lstm's and gru's: A step by step explanation. URL https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[19] Joseph Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

Appendices

A Specialisation Project Report

Fish counting using cameras and machine learning

Vebjørn Bjørlo-Larsen
Department of ICT and Engineering
Norwegian University of Science and Technology, NTNU
Ålesund, Norway
Email: vebjorbj@stud.ntnu.no

***Abstract*—This paper provides a proof of concept for the use of cameras and machine learning to perform counting of fish on a conveyor belt, designed to be used in fish processing facilities. The proposed solution is based on multi object tracking techniques, using Yolov4 for object detection and a combination of Kalman filter and Hungarian Algorithm for object tracking. Counting is done using an algorithm that counts a tracked fish as it passes an imaginary line in the video.**

Keywords—Object detection, convolutional neural networks, object tracking, Kalman filters, Hungarian algorithm

I. Motivational Aspects

The fish farming industry is a rapidly growing market [1], and thus the desire for new technology to improve productivity is big. Some areas where new technology can be used are for fish counting and monitoring in fish processing facilities. Existing solutions for fish counting use dedicated equipment installed as a step in the fish transportation setup, usually as a counting module inserted between pipes, or as a separate counting table that the fish are fed through [2][3]. This means planning and dedicated space need to be used for the fish counters. This is especially challenging when it comes to upgrading or adding counters to a system, as it would require changes to the layout and at least partial halts to the production as parts are changed.

For these reasons, this work aims to explore the possibility of using cameras and machine learning technology to perform fish counting. There are several benefits of using cameras instead of dedicated equipment for fish counting. They take up less space and can easily be mounted to new or already existing equipment without the need to change the layout. Cameras are non-intrusive, as they do not interact or interfere directly with the fish. While existing solutions are made to have as little impact on the fish as possible [2], they do add an additional step in the transportation, which can increase the chance of damaging the fish. Cameras are also versatile; they can be mounted throughout a fish processing facility without having to redesign and overhaul it.

For a camera to be competitive, it needs to offer similar or better results than the existing technologies. While actual accuracy will depend on the implementation conditions, current solutions are rated at 97% or above accuracy [2][3] in optimal conditions, which means there is a high burden of accuracy to be met.

In addition to performing counting, cameras can also be used for other types of inspections simultaneously, such as quality or damage controls. The focus of this project is on fish counting, but there are a multitude of other areas of interest, and the flexibility of using cameras and video analysis allows for easy expansion in the future without having to redesign the system.

This project was proposed by and done in collaboration with Stranda Prolog AS. After initial discussions with the company a set of objectives and requirements were established. Stranda Prolog AS were also responsible for organizing and installing cameras for data acquisition.

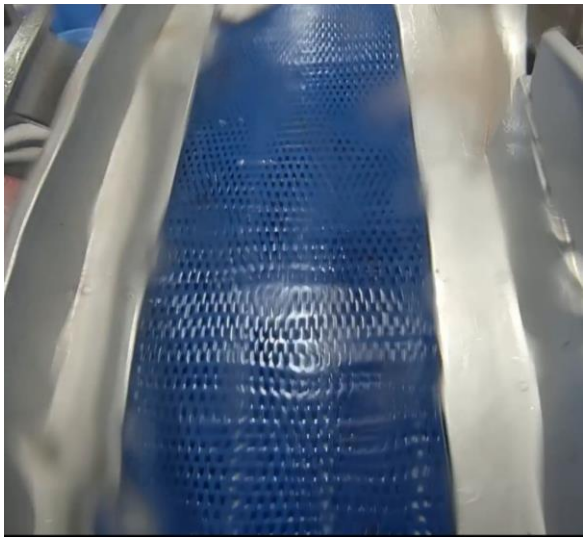


Fig. 1. Cropped image of current camera setup. Height is accurate, but the width is roughly 2/3rds of full width.

II. Objectives

The project has three main objectives. The first objective of this project is to perform a literature review of existing solutions and technology from other research areas that can be applied to this topic. There are three main challenges that needs to be addressed:

1) Detecting fish in frames of the video/camera feed;
2) Applying methods to only count each fish once (each fish will appear in many frames);
3) Identifying methods that can run in real-time.

The second objective is to gather video material of fish on a conveyor belt and create a labelled dataset out of this video material to be used to train machine learning models.

The third objective is to apply methods and algorithms from the literature review and perform fish counting on test video, with the goal of getting promising results and to lay a foundation for further work in the area.

III. ADOPTED METHODOLOGY

To address the challenges laid out during the initial review phase, multi object tracking (MOT) techniques were chosen. Multi object tracking consists of two main parts, object detection and object tracking. Object detection deals with detection of objects in each individual frame, and object tracking attempts to track each object from frame to frame [4]. The goal is that by successfully implementing both parts, requirements for challenge 1) and 2) are met. Additional care must be taken in choosing which methods to use to address challenge 3).

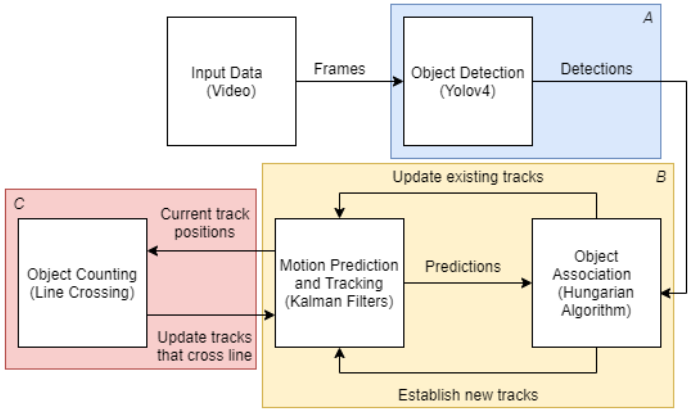


Fig. 2. Overview diagram of the developed solution

Figure 2 gives an overview of the developed solution, which consists of three main parts. Object detection (a) and object tracking (b) using multi object tracking techniques, followed by a counting algorithm (c) to count each detected fish once (and only once). In the following paragraphs these parts are described in more detail.

A. Object Detection and Dataset Creation

Object detection is an area with a lot of interest and research, with a large variety of machine learning methods and models to choose from [5]. For this project, both speed and accuracy are important, so Yolov4 was chosen as the object detection model. Yolov4 is a state-of-the-art object detection model, using a convolutional neural network to detect objects with high accuracy in real-time [6].

The first step of the machine learning pipeline was to acquire data in the form of video recordings of fish being transported on a conveyor belt. The installation of cameras was done by Stranda Prolog AS in collaboration with one of their partners. For this project, a remote computer was set up to remotely record video from the camera, so that all the necessary video material could be acquired. Due to limitations with the setup, sections of video were recorded at 720p, 60 FPS.

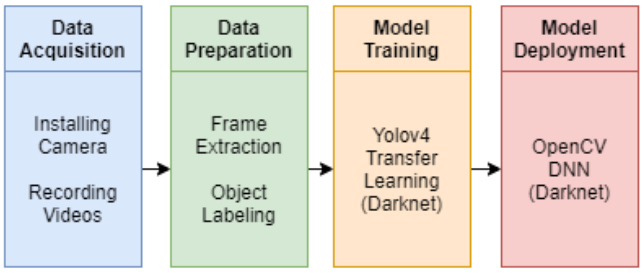


Fig. 3. Machine learning pipeline

After obtaining the video material the data was prepared for use in the model training. First, a section of video was converted into images using OpenCV [7]. These images were then manually labelled using the open-source labelling tool labelImg [8], saving the labels in the Yolov4 format [10]. This saves the labels for each image in a separate .txt file in the same folder and with the same name as the image.Yolov4 uses a normalized format to store labels, so each label consists of a class variable and four normalized pixel values indicating the rectangular label. Only a single class was used (fish), so all labels have 0 as the class variable. Below is an example label:

0 0.418359 0.706944 0.302344 0.563889

Currently 575 images have been labelled for training, with between 1 and 12 labels per image, totalling 2948 labels. An additional 45 images have been labelled from a different video, which will be used to test the model on new data.

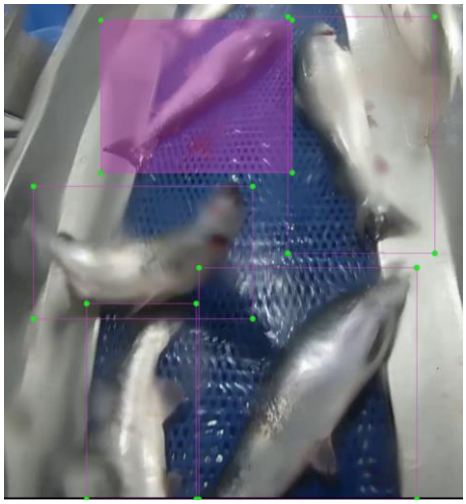


Fig. 4. Example of image with 5 labelled fishes.

Training of a custom model was done using the Yolov4 darknet framework, following the steps outlined by the Yolov4 authors [10]. The model is trained using transfer learning, with pre-trained weights from the Yolov4 authors and custom parameters recommended by the Yolov4 guide based on the custom dataset. Using transfer learning speeds up the training process and can achieve higher accuracy [9]. This especially applies when future improvements to the model are made. Using new data to train, the already trained weights can be used as the

starting point, which both speeds up the training and improves the accuracy of the new model.

The initial training was done with a 90/10 training/validation split, with the model set to be trained over 6000 iterations. However, the best results came after around 1500 iterations, so these weights were used when implementing the model.

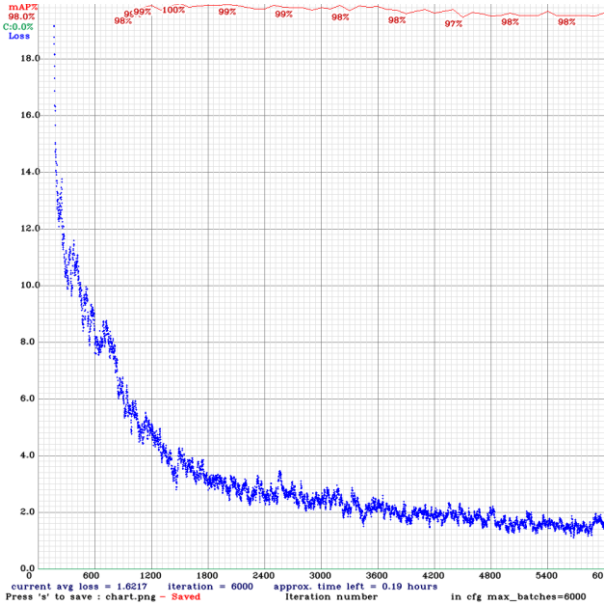


Fig. 5. Chart graphing training results, with blue indicating loss, and red indicating mAP%.

As can be seen in Figure 5 the training resulted in a very high mAP% of above 99%, which could be an indication of overfitting [11]. Testing the performance on the testing dataset, which was created from a different video section, gives a mAP% of 95%. This shows that there is some discrepancy between the validation and testing precision, so there is likely some degree of overfitting. The cause of this is most likely that the frames extracted from the video were too close to each other in time, so subsequent frames would be too similar. This meant that images in the validation set were too similar to the training data.

Despite some concerns of overfitting, for the purpose of this project the training results were very good. Whether the results would generalize well to videos from other cameras is uncertain, but for this project only one camera angle was available, so this is something that can be improved in the future when more camera angles are available.

The trained model was implemented using OpenCV's deep neural network module in Python. Detections are made by sending a frame through the model, which returns boundary boxes with classification and probability scores for each detection. The results from subsequent frames are then used to perform object tracking.

B. Object Tracking

Object tracking is usually done in three stages; Feature extraction or motion prediction stage, affinity stage, and association stage [4]. Each detected object is stored and tracked as its own object (track). This object includes its own Kalman filter instance, as well as other parameters such as a unique identification, and whether the object has been counted or not.

For this project, a motion prediction model was adopted for the first stage. The first step of the motion prediction model is to predict where each tracked object (fish) will be in the next frame. This is done using a Kalman filter for each object, which predicts the next position based on the physical model of the object (position, velocity) and the previous measurements of the object [12]. The Kalman filter is an iterative process that updates its own prediction model after each iteration based on how accurately it predicted the new position. To update the filter, each prediction must be matched to new detections from the next frame. This is done using the results from the affinity and association stages.

The affinity stage calculates the cost between each prediction and each new detection, where the cost is the distance between each prediction / detection pair. For this implementation the Euclidean distance of each pair was used. In addition to the distance, a hard limit can be set, which means a prediction / detection pair with a distance above this limit will never be matched.

In the association stage the affinity costs are used to match predictions to the new detections, using the Hungarian Algorithm to perform the matching. The Hungarian Algorithm is an assignment algorithm, which minimizes the overall cost of assigning predictions to detections [13].

With the assignment done, the Kalman filters of each object can be updated based on the difference in the prediction and actual detection, or a new tracker can be created if the detection is a new detection (un-matched to any prediction). If a tracked object is not matched to a new detection, it could mean that the fish has left the frame, that it is occluded, or that the fish was not detected by the object detector. When this happens the Kalman filter will continue to predict new positions until a threshold has been reached. After this threshold the tracker is removed, and if the fish is detected again in future frames, it will be assigned a new tracker.

C. Fish Counting

Counting the detection is the final step in the process. Two main approaches were tested for this step; counting unique detections and counting a detection as it passes an imaginary line.

The first approach is to count each unique detection as a unique fish. This relies on the tracking algorithm perfectly tracking each fish, which is not always the case. For example, if a fish is occluded for too long it can be assigned a new unique detection when it reappears, or if a fish moves too quickly through the video it can be deemed to be a new fish in the next frame. During testing with the current tracking algorithm implementation this approach invariably resulted in over-counting, the degree depending on algorithm settings.

The other option tested is to count a unique detection as it passes an imaginary line in the frame. This gives much more control, as any misidentification that happens outside of the line will not affect the count. For example, if a fish is counted as a unique detection two times before passing the line, it wouldn't

matter as only the second unique detection would cross the line, and thus be the only one counted. Generally, this method tends to under-count, because if a fish is not detected properly as it passes the line it will not be counted at all.

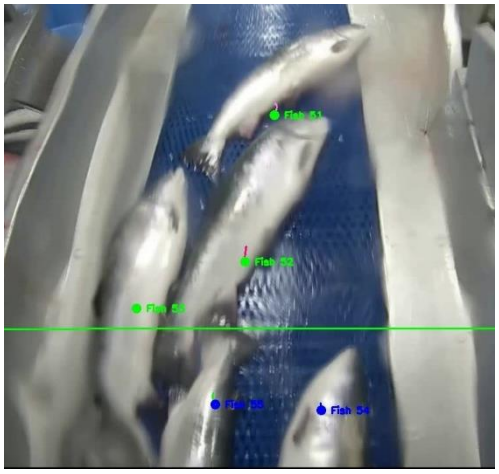


Fig. 6. Example frame showing fish being counted as they pass an imaginary line (green line). The green labels indicate that the fish has been counted, blue indicates that it is yet to be counted.

Of the two methods tested, counting a fish as it passes an imaginary line proved most accurate. It deals with common problems such as misidentification and multiple detections per object better than counting unique detections.

IV. RESULTS

To perform tests a separate test video was recorded and split into 5 sections based on their perceived difficulty. The decisions on where to start and stop the sections were made subjectively based on believed difficulty factors, such as crowding, occlusions, and fast-moving fish. The sections were given a difficulty level between 1 and 5, as illustrated in Figure 7.

Fig. 7. Example frames from each test video section, from difficulty level 1 (top-left) to difficulty level 5 (bottom-right)

The test videos were all saved in 30 FPs, while the full video was recorded at 60 FPs. This was done so that a comparison between framerates could be made. The test videos were used during testing to quicker test changes to the configuration, so the choice was made to make these 30 FPs as it would further speed up the testing.

The expected number of fish per section was manually counted going through the video frame by frame. This gives an evaluation criteria for how well the algorithm performs on the entire section of the test video, but doesn't necessarily say how well each fish is counted. There are some instances of double counting, which would incorrectly improve the total accuracy assuming a base of under-counting.

All of the video sections, with detections marked, are included in the separate zip file showing the project demos.

	Video Section based on difficulty (30 FPS)					**Full Video 30 FPS**
	1	2	3	4	5	
Video Length	0:13 Minutes	0:47 Minutes	1:01 Minutes	1:32 Minutes	3:39 Minutes	7:15 Minutes
Expected Fish	5	34	67	104	341	551
Fish Counted	5	33	65	98	304	505
Percentage Counted	100%	97%	97%	94%	89%	92%

Table 1. Results performing fish counting on each video section. Expected fish count is manually counted going through the video frame by frame.

The results in Table 1 shows some promising results, particularly for sections with fewer fish. In sections with few fish, or a moderate amount of non-moving fish, the current algorithm can successfully detect and count most fish once and only once. Manually reviewing the results shows that all fish are accurately counted in section 1, whereas the missing fish from section 2 is detected, but the detection is lost (merged with neighbouring fish) right before it is counted.

A manual review of section 3 shows that all 65/67 fish counted were accurately counted, with only 2 fish not being counted and no instances of double counting. The two missed detections were both caused by fast moving fish not being counted.

In more crowded sections the accuracy is lower, and it is especially poor when there are fast-moving fish (which often occlude other fish temporarily) or fish not laying still, which causes the detection centre to shift back and forth rapidly.

Manually reviewing section 4 shows that in fact 9 fish were missed by the counter, either due to being occluded at the time of counting, misidentified as a previously counted fish, or two fish detected as a single fish by the detector at the time of passing the line. There were 3 cases of double counting, which incorrectly improved the apparent accuracy to 98/104 instead of 95/104 fish counted. The reason for double counting were fish being detected as two unique fishes at the time of passing the line, so both detections were counted.

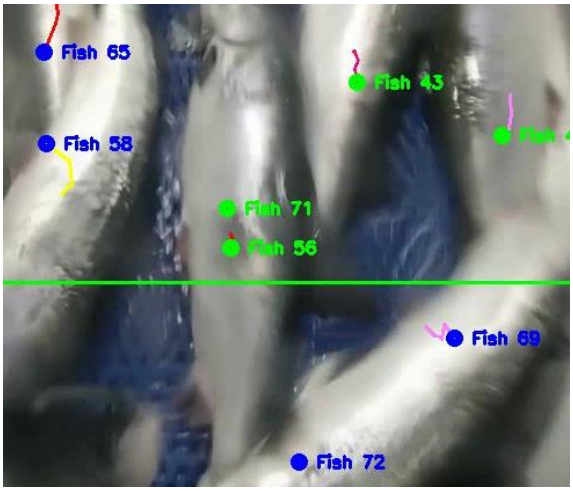


Fig. 8. Example showing a fish being counted twice. Middle fish was wrongly detected as two fish (Fish 71, Fish 56), which were both counted. Fish 65 and 58 were both counted as different fish, but were falsly assigned new unique identifications after crossing the line. This is a counter example showing how the line also prevents double counting.

Section 5 had by far the worst results, which was expected due to the number of fish and long periods of crowded frames. In this section at least 12 fish, but probably more, were counted twice (it is sometimes difficult to determine). Most of the time this is caused by a crowded scene where the fish are still moving by themselves, causing the centre of the detection to move back and forth rapidly. When this happens as the fish crosses the line it might incorrectly be detected as two separate fishes frame to frame, and because the centre of the new detection was before the line, it is counted as it passes the line. This means that while the total accuracy for the section was 89%, only around 85% of the fish were counted.

	Video Section based on difficulty					**Full Video**
	1	2	3	4	5	
Video Length	0:13 Minutes	0:47 Minutes	1:01 Minutes	1:32 Minutes	3:39 Minutes	7:15 Minutes
Fish	5	34	67	104	341	551
Fish / Second	0.38	0.72	1.10	1.13	1.56	1.27
Average FPS	64.0	63.7	58.8	60.9	54.6	57.4

Table 2. Frame rate performance of running algorithm on video sections. Performed using a NVIDIA RTX 2080TI graphics card.

As laid out in the objectives, another target for the project was to develop an algorithm that can run in real-time. As Table 2 shows, the current algorithm can run at between 55-65 FPS (which includes drawing time to draw detections for each frame) using a powerful GPU. The average FPS was calculated running each section three times, averaging the average FPS from the three runs.

This means the test setup can easily run at a target framerate of 30 FPS, with room for either more resource intense upgrades to the algorithm or running at a higher framerate. To test whether running at a higher framerate is advantageous, the same test that was performed on the 5 sections was performed on a full video at 60 FPs.

	Full Video (30 FPS Sections)	**Full Video (60 FPS)**
Video Length	7:15 Minutes	7:15 Minutes
Expected Fish	551	551
Fish Counted	505	527
Percentage Counted	91.7%	95.6%

Table 3. Comparison of performance between 30 and 60 FPS. 30 FPs is combined performance from the 5 sections, while 60 FPs is of the full video.

The comparison in Table 3 shows a clear improvement from 30 FPs to 60 FPs. After manual review of the video, the main reason appears to be that fast-moving fish are better detected. There are more frames for the fish to be detected, so it has an improved chance of being counted.

Test computer specifications
AMD Ryzen Threadripper 2950X 16-Core Processor (3.50 GHz)
2x NVIDIA GeForce RTX 2080 Ti, 11 GB GDDR6-Memory (only one GPU used for testing)
64 GB 2666MHz DDR4 RAM
64-bit Windows 10 Education

Table 4. Computer specifications of test computer

V. Discussion and Future Work

The results of the project are promising, especially considering the current limits of the implementation. There is much room for improvements, which makes the current results promising for future development upon the work laid out in this project.

One possible area of improvement is the dataset used for training and testing the current implementation. The video quality is good enough for a proof of concept, but there are some significant issues with the current dataset that can help improve the overall results. The current dataset was created from video recorded with some water on the lens. This creates some noticeable blurring and distortions in certain frames, making it more difficult to distinguish between fish.

Another possible improvement to the camera is its orientation relative to the conveyor belt. Currently, the conveyor belt is vertical in the frame and recorded at an angle, meaning it only takes up a small portion of and loses focus towards the top of the frame. This could be improved by rotating the camera 90 degrees and pointing it directly downwards, so that the conveyor belt is horizontal relative to the camera.

When it comes to the training dataset, another improvement that can be made is the choice of frames. Taking more diverse frames, and preferably frames from different angles or camera positions, could help improve generalization and prevent overfitting.

The object detection itself performs well for the current data and is for the most part very reliable. There are a few instances where fish close together or over each other are detected as one fish, but most of the fish that are not counted are not caused by missing detections. With improved video material these possible issues should be even less likely to happen. When it comes to generalization, this project lacks tests to validate how well the trained model generalizes to different camera angles. However, this should also be possible to test and if necessary, improve with an improved dataset.

The area with most room for improvement is the object tracking. For calm scenarios with clearly distinguished fish the tracking works well, but for chaotic scenes with lots of movement and occlusion there are some clear issues with misidentification and fish "losing" its track and being assigned as a new unique detection.

There are two main ways to improve the object tracking. The first is to continue tuning the algorithm parameters or possibly change the prediction and matching algorithms. The current results were achieved through tuning the current algorithms, but there should still be room for improving the tuning. Improved video quality should also have a noticeable effect on the tracking accuracy.

The other way is to completely change the way the object tracking is performed. The object tracking is currently implemented using a physical model of the system, predicting and matching based on velocity and position of objects (fish). An alternative to this is to use machine learning techniques for the tracking part as well. This is an area with a lot of research [3], with some of the possible architectures to explore being convolutional neural networks (CNN), long short-term memory (LSTM), or other recurrent neural network (RNN) architectures.

There are also other areas of future work that are important for a proper implementation of fish counting. The current software setup is conducive to testing, but needs to be streamlined for an actual implementation. Additionally, a proper interface should be chosen or created to allow for communication with other equipment in the facility and for controlling the counting process.

Finally, a proper assessment of hardware needs to be made based on future test results and any other requirements the final system might have. The current test hardware performed well giving an indication of what might be acceptable or even too powerful hardware, but further testing needs to be done to determine what best fits the full requirements of the system. For example, with improved detection and tracking methods, is it necessary to run the code at a high framerate for accuracy, or can a lower framerate, lowering computing requirements, be used?

VI. Conclusion

This work provides a working proof of concept for the use of video cameras and machine learning techniques to perform fish counting of fish on a conveyor belt. By using multi object tracking techniques, the objectives and challenges laid out in the objectives section are all met to varying degrees. The current implementation can both detect and track fish in videos at real-time speeds, with decent to very good results depending on how crowded sections of the video are.

There are still room for improvements, especially when there are many fish in a scene. The key areas of improvements are improved datasets for training and testing, and improved object tracking methods, potentially using machine learning techniques instead of the current physical tracking model.

Acknowledgment

The author would like to thank supervisor Ibrahim A. Hameed of Department of ICT and Natural Sciences at NTNU for his assistance during the project, and Stranda Prolog AS for the project proposal, for organising video material gathering, and for any other assistance required. The author would like to thank Kjetil Osland Brekken as the contact person for Stranda Prolog AS.

References

[1] Fiskedirektoratet. (2019, May 29*). Hvor stor er oppdrettsnæringen i Norge?* [Online]. Available: https://www.fiskeridir.no/Akvakultur/Nyheter/2019/0519/Hvor-stor-er-oppdrettsnaeringen-i-Norge

[2] AquaScan. *Fast, accurate & reliable fish counting equipment.* 2020. Available: https://www.aquascan.com/

[3] CALITRI TECHNOLOGY. *Fish counters.* 2018. Available: https://www.calitri-technology.com/en/fish-counters/

[4] G. Ciaparrone, F. L. Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, F. Herrera. (2019, November). "Deep Learning in Video Multi-Object Tracking: A Survey". arXiv:1907.12740v4 [cs.CV]. Available: https://arxiv.org/abs/1907.12740

[5] J. Hui. "Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3)". [Online]. Available: https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

[6] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao. (2020, April). "Yolov4: Optimal Speed and Accuracy of Object Detection". arXiv: 2004.10934 [cs.CV]. Available: https://arxiv.org/abs/2004.10934

[7] OpenCV. Open Source Computer Vision Library. 2020. Available: https://opencv.org/

[8] Tzutalin. LabelImg. Git code (2015). Available: https://github.com/tzutalin/labelImg

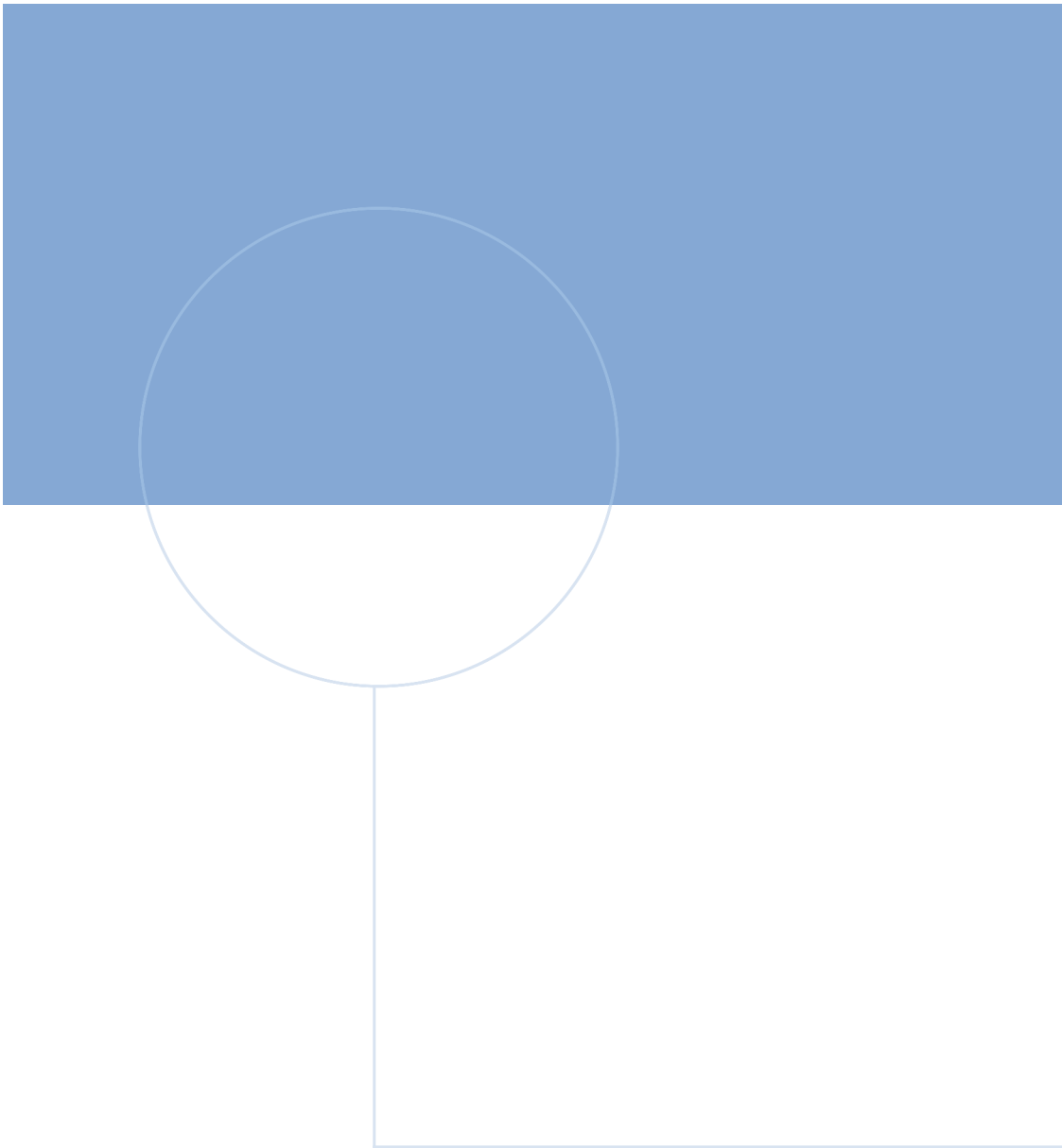
[9] J. Brownlee. "A Gentle Introduction to Transfer Learning for Deep Learning". [Online]. Available: https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[10] A. Bochkovskiy. "Yolo v4, v3 and v2 for Windows and Linux". Git code (2020). Available: https://github.com/AlexeyAB/darknet

[11] I. A. Hameed, Class Lecture, Topic: "Fundamentals of machine learning". IE500618, Department of ICT and Natural Sciences, NTNU, Ålesund. Aug. 2020.

[12] Kenshi Saho (December 20th 2017). Kalman Filter for Moving Object Tracking: Performance Analysis and Filter Design, Kalman Filters - Theory for Advanced Applications, Ginalber Luiz de Oliveira Serra, IntechOpen, DOI: 10.5772/intechopen.71731. Available from: https://www.intechopen.com/books/kalman-filters-theory-for-advanced-applications/kalman-filter-for-moving-object-tracking-performance-analysis-and-filter-design

[13] Hungarian Maximum Matching Algorithm. Brilliant.org. Retrieved 12:23, December 15, 2020, from https://brilliant.org/wiki/hungarian-matching/



NTNU

Norwegian University of
Science and Technology