Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Sondre Sagstad

Characterization of behaviour in tank rearing of salmon using machine vision and machine learning

Master's thesis in Cybernetics and Robotics
Supervisor: Morten Omholt Alver
June 2020

NTNU
Kunnskap for en bedre verden

Problem Description

For this thesis I was given the following problem description:

NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk
Institutt for teknisk kybernetikk

MASTEROPPGAVE

Kandidatens navn: Sondre Espe Sagstad

Fag: Teknisk Kybernetikk

Oppgavens tittel (norsk): Karakterisering av atferd i karoppdrett av laks ved hjelp av maskinsyn og maskinlæring

Oppgavens tittel (engelsk): Characterization of behaviour in tank rearing of salmin using machine vision and machine learning

Oppgavens tekst:
Landbasert produksjon av fisk er i raskt vekst, både i biomasse og i variasjon av arter, og når det gjelder smolt og post-smolt viser industrien spesielt interesse for å produsere post-smolt opp til 1000 g. Resirkulasjonsanlegg (RAS) velges normalt for vannbehandling ved nybygg, og anleggene blir stadig større. Det blir derfor stadig viktigere å kunne observere, tolke, avgjøre og handle på et rasjonelt og etterprøvbart grunnlag. SINTEF Ocean har introdusert Precision Fish Farming som et konsept der målet er å benytte metoder innenfor instrumentering, reguleringsteknikk og matematisk modellering for å monitorere, styre og dokumentere produksjonen. YNGELSENS er et samarbeidsprosjekt mellom MOWI, SINTEF Ocean, Skala og NTNU, finansiert av Norges Forskningsråd.

Hovedfokus i YNGELSENS er å utvikle nye, kamerabaserte sensorer for å overvåke og parameterisere (tallfeste) fiskeatferd og knytte atferd opp mot regulering og optimalisering av utfôring for å redusere slamproduksjon og bedre vannkvaliteten.

Denne oppgaven bygger på tidligere prosjektoppgave der et nevralt nettverk ble trent opp til å segmentere fisk basert på bilder fra oppdrettskar. I masteroppgaven inngår følgende aktiviteter:

- Utvide og forbedre datasettet etablert i prosjektoppgaven ved hjelp av annotering av flere bilder, preprosessering, data augmentation og evt. opptak av nye bilder.
- Basert på datasettet, etablere et opptrent MASK-RCNN-nettverk som segmenterer fisk på en tilfredsstillende måte.
- Utrede metoder for å beregne egenskaper som fart, aksellerasjon og vertikale bevegelser i bildeserier, og velge en metode for videre arbeid.
- Implementere og teste metode som trekker ut disse egenskapene fra bildeserier.
- Evaluere resultatene og gi anbefalinger for videre arbeid

Oppgaven gitt: 6. januar 2020

Besvarelsen leveres innen: 1. juni 2020

Utført ved Institutt for teknisk kybernetikk
Veileder: Morten Omholt Alver

Trondheim, 6. januar 2019
Morten O. Alver
Fagærer

Summary

Being able to automatically classify salmon behaviour is a sought after solution by the salmon industry. Good classification would help detecting unwanted events such as the spreading of a disease or H_2S congestion, early in the process. Today, these classification processes are mostly done through manual inspection, and as a consequence they are subject to subjective opinions. Especially in the dark, determining the state of the fish tank becomes extremely difficult. Behaviour changes can be subtle and hard to notice. Decisions are based on models and experienced operators. As a consequence, the industry is seeking a more reliable monitoring system.

Recent years have shown that Deep Learning is an excellent tool for both action recognition and segmenting objects in images. In this thesis we explore the possibilities for automatic characterization of salmon behaviour in fish tanks through machine vision and machine learning.
We develop a Mask R-CNN[12] capable of segmenting salmon smolt in images, and by feeding the masks generated by this network to a multiple object tracker, SORT[3], we are able to effectively track salmon smolt in video sequences. The results are used in methods for calculating characteristics such as velocity, acceleration, vertical- and horizontal movements.

Our results show that our setup is able to generate behaviour characteristics from fish tank that can be used to distinguish between behaviour classes. Through visualization of statistics we are effectively able to spot a deviance in the data from a dataset when the fish are spooked compared to a normal behaviour dataset. We hope that the results in this thesis will contribute to the development of automatic monitoring- and support systems in the aquaculture industry. Future work beyond the results in this thesis concerns improvements upon the Mask R-CNN, as well as the development of an automatic approach of finding anomalies in the generated tank statistics.

Sammendrag

Automatisk klassifisering av lakseadferd er en etterspurt løsning i oppdrettsindustrien. God klassifisering vil hjelpe med tidlig deteksjon av uønskede situasjoner, som f. eks spredningen av en sykdom eller H_2S opphopning. I dag er de fleste av disse klassifiseringsmetodene utført gjennom manuell inspeksjon, noe som medfører at de er utsatt for subjektive oppfatninger. Spesielt i mørket, er det å klassifisere tilstanden i et oppdrettskar en vanskelig oppgave. Adferdsendringer kan være subtile og vanskelig å oppdage. Som en konsekvens, søker oppdrettsnæringen bedre og mer pålitelige overvåkingssystem.
De siste årene har vist as Dyp Læring er et ypperlig verktøy for både handlingsgjenkjenning og segmentering av objekter i bilder. I denne masteroppgaven utforsker vi mulighetene for automatisk karakterisering av lakseadferd i oppdrettskar gjennom maskinsyn og maskinlæring.
Vi utvikler et Mask R-CNN[12] nettverk som klarer å segmentere laksesmolt i bilder. Gjennom så å fôre segmenteringen gjort av nettverket gjennom en algoritme, SORT[3], som kan tracke flere objekter samtidig, klarer vi effektivt å tracke laksesmolt i videosekvenser. Resultatene blir brukt i metoder for utregning av hastighet, akselerasjon, vertikale- og horisontale bevegelser.
Resultatene våre viser at oppsettet vårt klarer å genere kjennetegn ved adferden som gjør det mulig å skille adferdsklasser fra hverandre. Gjennom visualisering av statistikk, klarer vi effektivt å oppdage avvik i data fra et datasett som inneholder skremt fisk når man sammenligner det med et normalt datasett. Vi håper resultatene i denne masteroppgaven vil bidra i utviklingen av automatiserte overvåkings- og støttesystem i oppdrettsnæringen. Framtidig arbeid utover resultatene i denne oppgaven, angår forbedringer av Mask R-CNN, samt utviklingen av en automatisk tilnærming for å finne avvik i den genererte karstatistikken.

Preface

This thesis was prepared during the spring of 2020 at the Norwegian University of Science and Technology, Faculty of Information Technology and Electrical Engineering, Department of Engineering Cybernetics. The thesis was accomplished with the help of SINTEF Ocean AS and MOWI AS, Slørdal. The GitHub libraries Mask R-CNN[1], SORT[2] and CLoDSA[14] form the basis of modified versions, which we use in this thesis. The camera equipment we use is made available by NTNU and SINTEF Ocean AS.

I would like to thank my supervisor Morten Omholt Alver for his guidance and clarifying discussions through this work. Secondly, I would like to thank Torfinn Solvang at SINTEF Ocean (now at ScaleAQ) for his help and thoughts throughout the project. Lastly, I would like to thank my fellow student Andres Granberg Drønnen for his discussions and cooperation in collecting the datasets used in this thesis.

Table of Contents

Summary i

Preface iii

Table of Contents v

List of Figures viii

Abbreviations ix

1 Introduction 1
1.1 Motivation . . . 1
1.2 Specialization Project . . . 2

2 Basic Theory and Previous work 4
2.1 Deep Learning . . . 4
2.1.1 NNs - Neural Networks . . . 4
2.1.2 CNN - Convolutional Neural Network . . . 8
2.1.3 Mask R-CNN . . . 11
2.1.4 Transfer Learning . . . 12
2.2 Salmon videos . . . 12
2.3 Kalman filter . . . 13
2.4 Libraries and frameworks . . . 14
2.4.1 Mask R-CNN framework . . . 14
2.4.2 Google Colab . . . 15
2.4.3 SORT - Tracking . . . 15
2.4.4 CLoDSA - Data Augmenting Augmenting . . . 15

3 Materials and method 16
3.1 Data collection . . . 16
3.1.1 Dataset . . . 16

3.2 Method . . . 17
3.2.1 Testing Mask R-CNN in the Specialization Project . . . 17
3.2.2 Pre-processing . . . 18
3.2.3 Other preprocessing techniques . . . 25
3.3 Mask R-CNN . . . 27
3.3.1 Training Process . . . 27
3.3.2 Network variables . . . 29
3.3.3 Generating the input to SORT . . . 30
3.4 Multiple object tracking . . . 33
3.4.1 Kalman Filter . . . 33
3.4.2 Modifying the SORT algorithm . . . 34
3.5 Testing our complete algorithm . . . 37
3.5.1 Second trip to Slørdal . . . 37
3.5.2 Limitations . . . 39
3.5.3 Algorithm workflow . . . 39

4 Results and discussion **41**
4.1 Mask R-CNN results . . . 41
4.2 SORT performance . . . 44
4.3 Movement statistics . . . 45
4.4 Preliminary research in finding tail beat frequency . . . 54
4.5 Overall performance discussion and future work . . . 60
4.5.1 Sources of error . . . 60
4.5.2 Future work . . . 60

5 Conclusion **64**

Bibliography **65**

List of Figures

1.1 A cyclical representation of PFF where operational processes are considered to consist of four phases: Observe, Interpret, Decide and Act. The inner cycle represents the present state-of-the-art in the industry, with manual actions and monitoring, and experience-based interpretation and decision-making. The outer cycle illustrates how the introduction of PFF may influence the different phases of the cycle. Adopted from [8]. 2

2.1 Illustration of how wrong and confident predictions are penalized with a large loss. True label = 1. 7
2.2 The filter/kernel K is sliding or convolving over the image $\mathbf{I}$ and the Convolution operator is computing the feature map by computing the dot product between the filter and its location over the image. Figure is adopted from [22]. 9

3.1 Annotating the **Partial-IR** dataset using COCO annotator. Adopted from [28]. 17
3.2 Training loss vs. validation loss. The curves are the smoothed form of the original losses which is visible in the background. The smoothed graph is an exponential moving average, which is used to smooth out short-term fluctuations and highlight longer-term trends. The blue line represents the augmented dataset, while the orange line represents the original dataset. . 19
3.3 Illustration of the global thresholding method. In our case this method is very sensitive to the threshold value. The optimal threshold value is different for each image, making it unsuited for our application. 20
3.4 Illustration of the Adaptive Mean tresholding. This method very good at extracting the features of the fish in the image. Some noise remains in the image. 21
3.5 Illustration of the Adaptive Gaussian method. This image is very similar to the Mean Method image, but with a little less noise and less distinct features. 22

3.6 Resulting image when applying Otsu's method. This thresholding method has the worst performance. The reason behind this is the image histogram, which we see in figure 3.7. This algorithm wants to find a value between two peaks in the histogram, which makes the variances of the two classes minimal. As can see from the histogram, we only have one peak. 24
3.7 Histogram of our example image. As we can see there is only one peak, which makes Otsu's method unusable. 25
3.8 Augmentation techniques applied to an image. From left to the right, we have: original, sharpen, dropout, elastic deformation 26
3.9 Augmentation techniques applied to an image. From left to the right, we have: histogram equalization, salt and pepper noise, Gaussian noise, Gaussian blur. 26
3.10 Validation loss with Gaussian noise applied on 50% of the training images. The smoothed graph is an exponential moving average, which is used to smooth out short-term fluctuations and highlight longer-term trends. The blue line represents the original dataset, while the orange line represents the augmented dataset. 27
3.11 Configuration for Mask R-CNN. 30
3.12 Illustration of the input image to equation 3.14. 31
3.13 Illustration of ellipse drawn over the masked fish. 32
3.14 Original image of fish we draw an ellipse on. 33
3.15 Illustration of the velocity problem. Object A and Object B are the observed size of two similar sized objects. They move with the same velocity, which is one body length each second. If the body length in reality is $4m$, we have to multiply by the constant $4\frac{2}{\sqrt{6}}m$ to find the real velocity, which is $4m/s$. 36
3.16 Image from the new dataset. 38

4.1 Illustration of good masks. 42
4.2 Illustration of a failed segmentation by an early version of our network. There are feed pellets in the image that disturbs the masking. 42
4.3 Illustration of a bad segmentation when using thresholded images. 43
4.4 Illustration of a good mask when using thresholded images. 44
4.5 Illustration of the visual interface when using the SORT tracker. This image is taken before we added acceleration, area and angle. 45
4.6 Comparison between the number of detections over the last 5 frames for each class. 47
4.7 Comparison between the velocity of each class. 48
4.8 Comparison between the acceleration of each class. 49
4.9 Comparison between average velocity over every detection in the last 5 frames of each class. 50
4.10 Distributions of velocities. Here we can see a clear difference between the spooked class and the two other classes. 51
4.11 Comparison between the average acceleration over every detection the last 5 frames of each class. 52

4.12 Distributions of accelerations. The difference between the spooked class and the two other classes is less distinct here, compared to the velocity distributions. 53
4.13 A circular histogram visualizing the angles of each detected fish. The area of each bin represents how many data points are in each bin. A doubling of data points in a bin results in a doubling of the area(not the radius). As expected, we see that almost all fish are pointed towards the current. . . . 54
4.14 Illustration of the extracted circle around a fish tail. 55
4.15 Illustration of a isolated and masked tail. 56
4.16 Time series of the area of the isolated tail. 57
4.17 Times series of mean pixel intensity of a masked tail. 57
4.18 Optical Flow image of a masked tail. 59
4.19 Time-series of mean pixel intensity when using Optical Flow and converting to gray-scale. 59
4.20 Distribution of how long the fish is detected for. 62

Abbreviations

VIS	=	Visual spectrum of light
NIR	=	Near infrared spectrum
DL	=	Deep Learning
PFF	=	Precision Fish Farming
FPS	=	Frames Per Second
NN	=	Neural Network
CNN	=	Convolutional Neural Network
RoI	=	Region of Interest

Chapter 1

Introduction

In this chapter we will explain the motivation for the thesis, which is building on what we learned from the preceding Specialization Project [28].

1.1 Motivation

As described in the Specialization Project, the aquaculture industry is seeking automated monitoring and support systems. Building on what we learned from the project we will continue to explore the possibilities for automatic monitoring and classification of the state within fish tanks. With such systems we hope to discover behaviour changing events early the process. Examples of events to detect include include H_2S congestion, the spread of a disease and general deviance from normal behaviour. Such events can be hard to spot for an operator. Computers, on the other hand, can process enormous amounts of data and providing it with the right data it can help the operator arrive at the right conclusion.

The concept called Precision Fish Farming(PFF) [8], is a good example of this. It is explained in the preceding project[28], but we will repeat the vision behind the idea, which Martin Føre, a researcher behind the concept told *kyst.no*:

"The vision behind the use of high-tech equipment is to provide the farmer a safer and simpler everyday life as well as achieving higher production efficiency, better fish welfare and reduced environmental effects from the production. Examples on such solutions includes underwater cameras which together with automated algorithms provides quantified data on fish swimming speed, and solutions which combines online sensor data with mathematical models to better estimate the biomass and size distribution in the cages"[15].

A figure representing PFF can be seen in Figure 1.1.

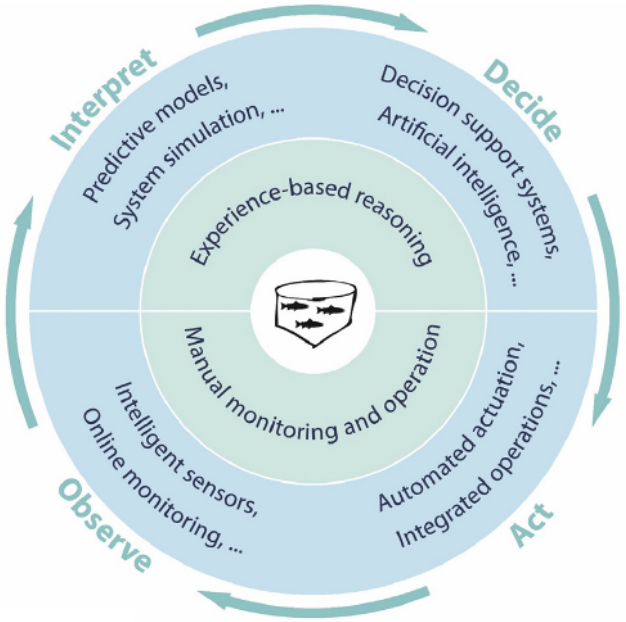


Figure 1.1: A cyclical representation of PFF where operational processes are considered to consist of four phases: Observe, Interpret, Decide and Act. The inner cycle represents the present state-of-the-art in the industry, with manual actions and monitoring, and experience-based interpretation and decision-making. The outer cycle illustrates how the introduction of PFF may influence the different phases of the cycle. Adopted from [8].

Recently, MOWI with SINTEF and NTNU as partners were forming the idea of a very interesting project called Yngelsens. Between this thesis and the preceding project the decision that Yngelsens is not going to move forward was made. However, the ideas behind the project represents the motivation for this thesis and will therefore stand. The main focus in Yngelsens was to develop new camera based sensors to help monitor and quantify fish behaviour. Examples of behaviour characteristics included swimming speed, acceleration, movement patterns and breathing frequency.

Based on our experience from the Specialization Project, the data we collected and the goals from Ynglesens, this thesis's focus will be the Observe quadrant in the outer cycle in figure 1.1(PFF).

1.2 Specialization Project

During the fall of 2019, I started to work towards the goals of Yngelsens in TTK4550 - Engineering Cybernetics, Specialization Project. A part-goal of Yngelsens was to identify state of the art machine learning methods for fish behaviour in video streams, and I chose that as my project. The main objective of the Specialization Project was to collect video

data and prepare it for use in deep learning segmentation algorithms. Initial tests were conducted using a Mask R-CNN to see how well the network would segment fish in a Near-Infrared Spectrum(NIR) image. The reason we use NIR instead of the Visual Light Spectrum(VIS) is because there is interest from the industry to monitor the fish at night as well as during the day. This master thesis is a natural extension of that work and will use the findings and conclusions from the Specialization Project as inspiration. We learned a lot in the Specialization Project regarding the difficulties in capturing high quality underwater images with limited lighting capabilities. As a consequence, the focus in the images will vary, and high detail characteristics such as breathing is hard to effectively spot. Therefore our main focus will first lie on characteristics that doesn't necessarily need the highest level of focus. These characteristics include swimming speed, acceleration and movement patterns.

We then propose the goals for this thesis to be:

1. Create a network that can successfully segment salmon in NIR images.
2. Create a tracker that can successfully track salmon in a video stream.
3. Generate data based on the tracking which includes velocity, acceleration and swimming/moving direction.
4. Test the program on new videos to check if we are able to distinguish between the data generated from three different behaviour classes: feeding, normal and spooked.

Chapter 2

Basic Theory and Previous work

This chapter is intended for readers which are unfamiliar with the topics to help them better understand the content, and covers the theoretical background for this thesis. In 2.1 it will cover Deep Learning. In 2.3 it proceeds to cover Kalman filters, which are used in our object tracking algorithm. Finally, in 2.4 it will give an overview of the tools and frameworks that are used in this thesis.

2.1 Deep Learning

This chapter will give the reader an overview of the field of deep learning. It will also cover previous work on models that are leading up to the model that we use. The field was briefly covered in the Specialization Project, and it will continue from there.

2.1.1 NNs - Neural Networks

Neural networks are graphs that consists of connected neurons or nodes. Each node has a set of learnable weights, W, at its connections and a learnable bias, b. The bias enables the neuron to activate even for zero-valued inputs. This an important part of the network to help it converge or learn 'good' weights and biases. A typical neural network has anything from a couple dozen to millions of nodes arranged in layers. Some of the nodes are input nodes. These receive some form of information from the outside world that the network will attempt to learn about. On the other side of the network are the output nodes. These nodes signal how the network responds to the input. In between the input and output nodes we have the hidden nodes, which form the majority of the neural network. The input nodes, output nodes and hidden nodes are divided into layers; an input layer, an output layer and hidden layers respectively. Most of the neural networks are fully connected. This means that each hidden node and output node is connected to every node in the layer on each side. These connections are represented by the learnable weights, W, which is a number that represents how much a node influences the node it is connected to. This number can both be positive or negative, and the higher the number, the higher influence one node

has on another. Information in neural networks flow two ways. In the learning process, information are fed to the network through the input nodes, which triggers the hidden nodes, which in turn trigger the output nodes. This way of information flow constitutes the common design called a feed forward network. How information flow backwards will be mentioned later in the chapter. When a neuron receives a set of inputs, x, it computes the dot product over these inputs with the weights W. It adds the biases and then feeds the result through a non-linear activation function to produce an output, y. The mathematical expression is shown in 2.1.

$$y = f(\sum_i W_i \cdot x_i + b) \tag{2.1}$$

The use of a non-linear activation function lets the neural networks approximate any function. The universal approximation theorem was first posed by George Cybenko and goes as follows:

A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$.

The most common activation functions are the Sigmoid, tanh and ReLU function. They are mathematically expressed as:

Sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

Tanh:

$$tanh(x) \tag{2.3}$$

ReLU:

$$max(0, x) \tag{2.4}$$

Backpropagation

The way neural networks learn is through a process called backpropagation. This is a feedback process where information flows backwards in the network. It involves comparing the output of the network to ground truth labels in our training data, and using the difference to adjust or train the parameters (W and b) in the network. Starting at the output nodes it works its way through the hidden layers and then to the input layer. In time, this algorithm will cause the network to learn, and the difference between the output and the ground truth labels will go towards zero.

Loss function

The evaluation on how well the network predicts the correct ground truth labels can be seen in the means of a loss function. If predictions deviate too much from the original data, the loss function will output a large number. The goal is to gradually, with the help of an optimization function reduce the deviation in the prediction. While there are several loss functions that are used in the field of deep learning, there are no one-size-fits all. Some of the most used are:

Mean Squared error/L2 Loss

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \tag{2.5}$$

where n is the number of training examples, i is the ith training example in a dataset, y_i is the ground truth label for the ith training example and $\hat{y}_i$ is the prediction for the ith training example. This loss function is concerned with the average magnitude error, irrespective of the direction. Predictions which are far away from the ground truth are penalized heavily due to squaring.

Mean Absolute Error/L1 Loss

$$MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n} \tag{2.6}$$

This function is concerned with the average sum of absolute differences between predictions and ground truth labels. This function does not either consider the direction of the error. Due to the absolute term, calculating gradients for this loss function is a harder task compared to the MSE loss function.

Cross Entropy Loss

$$CrossEntropyLoss = -\frac{1}{N}\sum_{i=1}^{n}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)) \tag{2.7}$$

for binary classification and

$$CrossEntropyLoss = -\sum_{i=1}^{n}\sum_{k=i}^{c} y_i^k \log(\hat{y}_i^k) \tag{2.8}$$

for multiclass classification, where c is the number of categories in the dataset.

This is a common loss function in the field of deep learning and it has the property that confident predictions which are wrong are heavily penalized . This can be seen in figure 2.1.

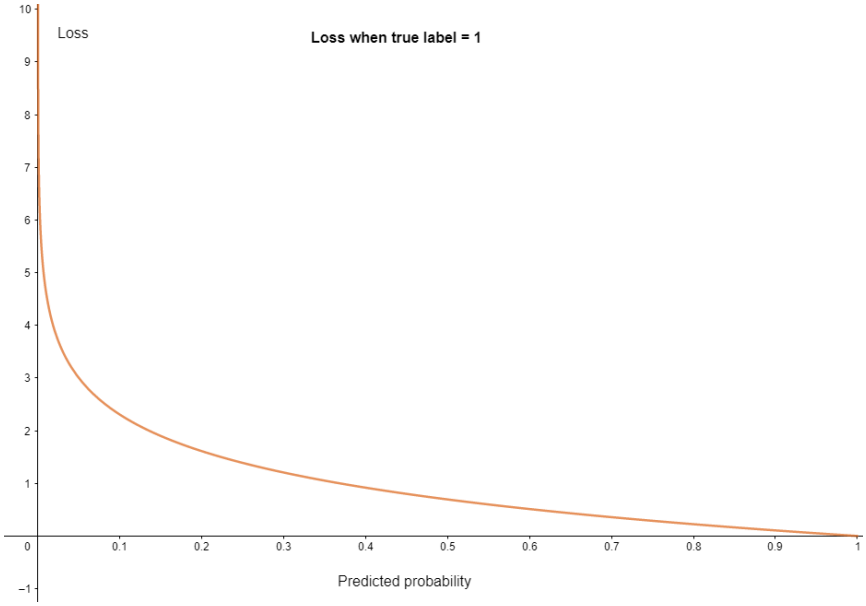


Figure 2.1: Illustration of how wrong and confident predictions are penalized with a large loss. True label = 1.

Optimization algorithm

The goal of of the optimization algorithm is to find a set of parameters that minimizes the loss function. The most common strategy to solve this problem is through gradient descent. In gradient descent we first compute the gradient of the loss function with the current parameters, and then update the parameters in the negative direction of the gradient. This is an iterative process which continues until optimal parameters are found.

If we now wish to calculate the gradient we can use a classifier/activation function such as the softmax function as example. Softmax is typically used for the output layer in a network, while the other mentioned activation functions are used for the hidden layers. Given an input x_i the softmax will output a vector y_i where each element in the vector, y_i^k, represents the probability of the input x_i being a member of category k. We get:

$$\hat{y}_i^k = \frac{e^{a_i^k}}{\sum_{k'} e^{a_i^{k'}}} \tag{2.9}$$

where

$$e^{a_i^k} = w_k^T x_i \tag{2.10}$$

where w_k is a weight vector. Softmax also hast the property that $\sum_k \hat{y}_i^k = 1$. Using the chain rule and the quotient rule, we can find the gradient with respect to the weights as:

$$-\frac{\partial E_i(w)}{\partial w_{kj}} = x_i^k(y_i^k - \hat{y}_i^k) \tag{2.11}$$

where w_{kj} is the weight from node j to node k. The same can be done for the biases. Collectively denoting the weights and biases as θ, the update rule for the parameters becomes:

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial E(n, \theta^t)}{\partial \theta} \tag{2.12}$$

where α is the learning rate and t is the iteration.
The training of a neural network can then be summed up as: feed the network some training data, calculate a loss based on the predictions the network makes, use backpropagation to perform a backwards pass to find adjustments for the parameters in the network and then update the network parameters.

2.1.2 CNN - Convolutional Neural Network

Convolutional Neural Networks are similar to ordinary Neural Networks in many ways as they are made up of nodes/neurons, biases and weights. They use the same activation functions and loss functions. The key property that is different between ordinary NNs and a CNNs is that CNNs assume that the input are images. This enables us to encode key features into the network architecture. The forward pass is more efficient and the network parameters are greatly reduced. Unlike NNs, CNNs have the nodes arranged in three dimensions, depth, width and height. Depth in this context does not refer to the number of layers in the network. Contrasting the fully connected structure of NNs, the neurons in a layer of a CNN are only connected to a small region in the layer before. A layer in a CNN essentially does a transformation from a 3D volume to a new 3D volume through a differentiable function.
The CNN architecture usually consist of convolutional layers, activation function layers, pooling layers and fully connected layers. After the input layer, the next layer is always a convolutional(conv) layer in a CNN. A typical input to such a conv layer is an image of 32 pixels in width, 32 pixel in height and 3 colour channels, RGB. Then, the input will be on the form 32x32x3. The conv layer uses a filter/kernel that slides over the regions of the input image. The local region it is sliding over is called the receptive field or filter size. A typical filter size is 3x3 or 5x5 along the width an height. The depth of a filer is the same as the input, which is 3 in this example. The numbers within the filters are the weights or parameters. As the filter is sliding, or convolving around the image it performs element wise multiplications that are summed up to a single number for each position of the filter. If a 5x5 filter is used it can fit on 784 different locations on the 32x32 input image. This is then mapped to a 28x28 array, which is called the feature map or activation map. The depth on the output will depend on how many filters were used. If 8 filers were used we would get 32x32x8 (zero-padding can be used to preserve the spatial dimensions) as the output from that layer. We calculate the feature map using the **Convolution operator**.It uses a two-dimensional image $\mathbf{I}$ and a filter/kernel $\mathbf{K}$ of size h$\times$ w. The equation is given by equation 2.13, and an example of how it works is seen in figure 2.2.

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} \mathbf{K}_{ij} \cdot \mathbf{I}_{x+i-1, y+j-1} \tag{2.13}$$

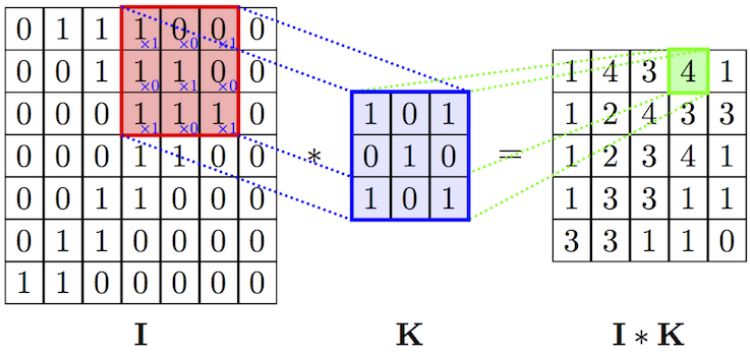


Figure 2.2: The filter/kernel K is sliding or convolving over the image **I** and the Convolution operator is computing the feature map by computing the dot product between the filter and its location over the image. Figure is adopted from [22].

Filters are often thought of as feature identifiers as they are often used to identify features such as curves, horizontal lines, vertical lines etc. in an image.
The next layer is an activation layer which will apply an activation function, such as the ReLU function, element wise. The output volume will stay the same. In our case it would stay at 32x32x8.
The next step is often a pooling layer. This is a much used layer between the convolutional layers in a CNN. This layer applies a down-sampling operation along the spatial dimensions (width and height). This is done to reduce the number of parameters in the network, thereby reducing the computational load. It also counteracts the phenomena of over-fitting due to the reduction of trainable parameters. These layers also use filters(not trainable), with a usual size of 2x2 applied with a stride of two. It works independently on each depth slice. Usually the MAX operation is used on each receptive field, which is called MAX pooling, but there can also be average pooling or L2-norm pooling. With a filter size of 2x2 and a stride of 2, it will down-sample the input height and width by two. In our case the volume will now be 16x16x8.
At the end of the network we find the fully connected layer. This layer takes an input volume and transforms it to a N dimensional vector where N is the number of classes the network can choose from. In our example the volume would now be 1x1xN. Each number in the vector represents the class score. As an example, $N = [0.9, 0.1]$ if we have two classes and the softmax function is used in the last layer.
A typical architecture of a CNN takes the form: input layer $\rightarrow$ convolutional layer $\rightarrow$ ReLU layer $\rightarrow$ convolutional layer $\rightarrow$ ReLU layer $\rightarrow$ pool layer $\rightarrow$ ReLU layer $\rightarrow$ convolutional layer $\rightarrow$ ReLu layer $\rightarrow$ pool layer $\rightarrow$ fully connected layer.
There are several well know architectures in the field of Convolutional networks, which include:

LeNet

LeNet [20] was one of the first successful applications of CNNs. Released in 1998, the network had a very simple architecture consisting of seven layers with around 60 000

parameters in the network.

AlexNet

The AlexNet [18] was released in 2012 and built on the structure of the LeNet, but was deeper and had significantly more parameters with its 60 million parameters.

GoogLeNet/Inception v1

Released in 2014, the GoogLeNet [31] network also built upon the LeNet, and contributed with a new element called an Inception Module. It performed very close to human performance on the task it was set to solve. Even though it was 22 layers deep, the parameters in the network were greatly reduced with only around a 10th of the parameters the AlexNet.

VGGNet

In 2014, the VGGNet [30] showed that depth is an important aspect of a network. Essentially stacking more layers on-top of the AlexNet, this uniformly designed network became the runner up for the ILSVRC 2014[27] contest, which the GoogLeNet won the same year. One of the downsides of this network is that it has close to 140 million parameters, which takes up a lot of space and computational power.

ResNet

ResNet [13] was the winner of the ILSVRC 2015 [27] competition. It introduced skip connections and removed fully connected layers at the end. Thanks tho the skip connections they were able to develop a 152 layer network while still having a lower complexity than the VGGNet.

Region-Based Convolutional Neural Networks (R-CNN)

Introduced by Ross Girshick et al.[9] in 2013, R-CNNs improved regular CNNs through using the Selective Search algorithm [32] to select a manageable number of region proposals. Region proposals are regions in the image where there might be an object. A drawback of ordinary CNNs comes when there are multiple objects in the image to detect. When there are a variable number of objects in the image, the length of the output layer (fully connected) is variable. To overcome this problem, Ross Girshick et al. used the selective search algorithm to identify a manageable number of bounding-box object candidates or ”regions of interest” (RoI). The number of original region proposals were around 2000. After the regions were identified, they used a CNN to extract features from each region independently. Then they classified each region using a class-specific linear SVM(Support Vector Machine). Even though the introduction of R-CNN made improvements upon the regular CNNs there were still drawbacks. The amount of time to train the network was huge due to the fact that it would have to classify $\sim$ 2000 region proposals per image. Real time applications were therefore not possible as it took around 47 seconds to classify each test image.

Fast R-CNN and Faster R-CNN

The same person(Ross Girshick) improved some of the drawback of the R-CNN when he developed the Fast R-CNN [10]. The approach is similar to the original network, but instead of using the CNN to extract features from each region independently, the whole image were fed into a deep CNN at the start of the algorithm. This created a feature map that were then used generate region proposals. A pooling layer, some fully connected layers and a softmax layer were then used to predict the class of the region proposal. This algorithm improved upon the regular R-CNN algorithm by quite a bit in terms of speed. Now, instead of feeding 2000 region proposals to a CNN, the convolution operation is only done once per image. While the training time of R-CNN was 84 hours, the Fast R-CNN "only" used 9,5 hours. Classifying the testing images also showed great improvements by only using 0.32 seconds compared to 47, which made it more applicable for real-time applications. The speed was improved even further with the introduction of Faster R-CNN [26]. This design got rid of the selective search algorithm, which is quite time consuming. It was replaced by a separate network, which were used to predict the region proposals. Selective Search uses 1-2 seconds on each image(not accounted for in the mentioned running times for R-CNN and Fast R-CNN), depending on content, while the Faster R-CNN design only uses 198ms for both proposal and detection.

2.1.3 Mask R-CNN

Building on Fast R-CNN and Faster R-CNN, the Mask R-CNN [12] was developed. This is the network architecture that will be used in this thesis. The goal of Mask R-CNN was to take Faster R-CNN to a level that could also do pixel level segmentation. By adding a branch to Faster R-CNN that outputs a binary mask that tells whether a given pixel is part of an object or not, they made a network that not only detected different objects, but also segmented and classified them. In addition to this branch, they also replaced the Region of Interest Pool Layer with a new Region of Interest Align Layer to increase the alignment of regions throughout the network, which is needed when working with pixel level classification. Both Faster R-CNN and Mask R-CNN uses the ResNet101 as a backbone. The backbone act as the feature extractor in the design (it creates the feature map), before the region proposal happens. For efficiency, during the Region Proposal process, Mask R-CNN and Faster R-CNN uses something called anchors or anchor boxes to detect multiple objects, overlapping objects and objects of different scales. They are a set of predefined bounding boxes with predefined location relative to the images. Ground truth bounding boxes and classes are assigned to individual anchors. Some filtering is done to remain with the anchors that have a high confidence score(a predicted bounding box that overlaps much of the ground truth bounding box). The way Mask R-CNN essentially works can be summed up as:

1. The backbone creates a feature map.
2. RPN proposes regions that may contain objects with the help of anchors.
3. The algorithm uses the proposed region to predict bounding boxes, classifications and masks.

The reason we choose the Mask R-CNN design for our task of segmenting salmon smolt in videos is due to its state-of the art performance since its arrival. Only in the recent year, some methods have outperformed Mask R-CNN on object instance segmentation [25]. However, the amount of resources and documentation on the MASK R-CNN is very large and will therefore be the preferred design.

2.1.4 Transfer Learning

A technique we will be using when training our Mask R-CNN is transfer learning. The general idea behind this technique is to use knowledge from previously learned tasks and apply it in a new situation. In the same way as when humans encounter a new situation and uses previous experiences and knowledge to solve a task, transfer learning will do the same. Creating or labelling our training and validation data requires a lot of time. Transfer learning make use of existing datasets to reduce the size of training data needed. Cases with limited training sets such as ours, with ~ 1000 images, can make great use of models trained with 1 million images to gain low- and mid level feature definitions. We essentially want to make our model generalize to unseen data, so it is able to classify and segment the images as good as possible. Therefore, in this thesis, instead of starting from scratch, we will use a model trained on a similar task as our starting point. More specifically, we will use a model that is pre-trained on the COCO[21] dataset. This model is trained to segment and classify different objects in images.

2.2 Salmon videos

Segmenting objects in videos and images are a common task within the field of computer vision and deep learning. However, most of these videos are captured on land. Filming under water on the other hand introduces several challenges, as light behaves different in air than in water. Depending on the environment, the lighting will vary. When filming in the ocean, lighting condition will depend heavily on the weather, but also on the depth placement of the camera and overall visibility in the water(due to particles). When filming in fish tanks indoor, the environment is more controlled and it is easier to reproduce the same lighting conditions. Yet, light will still be scattered and absorbed. This is is mentioned in greater detail in my Specialization Project report[28]. The videos used and filmed during this thesis will come from an indoor fish farm at Slørdalen, MOWI, where we had relatively controlled conditions. The salmon there are kept in tanks which are $\sim$5m in diameter. The lighting within the room is controlled to control the salmons life cycle, and feeding happens automatically every couple of minutes (small amounts of feed dropped in at a time). The experience we got during the Specialization Project highlighted many of the difficulties when it comes to filming underwater, especially regarding the lighting. For NIR videos this became very prominent as we only had one source of light. Balancing the angle of the camera and the angle of the light beam to capture images without too much reflection from the fish and particles, and still lighting up the the image sufficiently were a difficult task. To accomplish this we had to use a large lens aperture and a slightly adjusted exposure time. The downside was that the focus in the images suffered slightly from these

adjustments. In this thesis, we will use the videos captured in my Specialization Project as training data, and we will capture new videos for testing our setup.

2.3 Kalman filter

In this thesis we will be using Kalman filters[16] for tracking purposes. The Kalman filter or linear quadratic estimator, which it is also called, is essentially used to calculate estimates of unknown variables in a system. It does so based on a model of the system, the uncertainty of the model, measurements and the uncertainty of the measurements. Systems where you have uncertain or noisy information are often a good place to use the filter. Kalman filters are well suited for systems that are continuously changing and have the advantage that they are fast and do not require a lot of memory since only the previous state has to be saved(it is recursive). It has long been regarded as the optimal solution for tracking and prediction tasks [6]. The goal is to minimize a loss function, and we measure the performance through this function. The purpose of using this filter is to extract useful information from a signal while ignoring everything else. The Kalman filter works in two steps, a prediction step and an update step.

In the prediction step it produces a state estimate for the current time-step based on the state estimate in the previous time step. It does not include observation information from the current time step and is therefore known a an *a priori* state estimate. An *a priori* covariance prediction is also is also calculated in the prediction step. This matrix puts a number on the uncertainty in our model.

In the update step, the state estimation is refined by combining the calculations made in the prediction step with current observations. This is called the *a posteriori* state estimate. Writing this mathematically we first define the variables:

- $\hat{\mathbf{x}}_{k|k-1}$ is the state estimate at time step k before the k-th observation is made
- $\mathbf{P}_{k|k-1}$is the a priori covariance matrix (estimate of the accuracy of the model) at time step k
- $\mathbf{F}_k$ is the state transition model
- $\mathbf{H}_k$ is the observation model
- $\mathbf{Q}_k$ is the covariance of the process noise
- $\mathbf{R}_k$ is the the covariance of the measurement noise
- $\mathbf{B}_k$ is the control input
- $\mathbf{z}_k$ is an observation of the true state $\mathbf{x}_k$.
  $\mathbf{z}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k$, where $\mathbf{v}_k$ is the observation noise. This noise is assumed to be zero mean Gaussian white noise with covariance $\mathbf{R}_k$: $\mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k)$

The *a priori*/prediction equations become:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B_k}\mathbf{u}_k \tag{2.14}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k\mathbf{P}_{k-1|k-1}\mathbf{F_k^T} + \mathbf{Q}_k \tag{2.15}$$

The *a posteriori*/update equations become:

$$\tilde{\mathbf{y}}_\mathbf{k} = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1} \tag{2.16}$$

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k \tag{2.17}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1} \tag{2.18}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_\mathbf{k} \tag{2.19}$$

$$\mathbf{P}_{k|k} = (\mathbb{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{2.20}$$

This ordinary version of the filter are used on linear process models with Gaussian distributed process- and measurement noise. This is the basics of the Kalman filter. The reason we are going to use this tool is because we need a way to track the salmon in our videos after the instance segmentation process.

2.4 Libraries and frameworks

This section will present tools, libraries and frameworks used for this thesis.

2.4.1 Mask R-CNN framework

There are multiple implementations of Mask R-CNN that already exist, and for this thesis we will use Matterports[1] implementation of the algorithm. This framework is built using both the Keras and TensorFlow libraries. It uses Feature Pyramid Network as feature extractor with a ResNet101 backbone. Our version will be a modified version of Adam Kellys [17] usage of the Matterport implementation. He trained the network on a dataset consisting of cigarette butts instead of the original COCO dataset. This library is then modified to fit our dataset. (Throughout this thesis modifications were also made to the original Matterport code to extract or add needed functions. Changes also had to be made to combat the continuous updating of Python libraries which would sometimes break the program.)

2.4.2 Google Colab

Due to the heavy computational load of training we need to make use of the parallelization properties of a GPU. For this thesis, we use the Google Colaboratory service for training the Mask R-CNN. This is a free cloud service which allows you to run code on powerful GPUs. The GPUs avalable when running your code are usually Nvidia K80s, T4s, P4s and P100s. There is no way, however, to know which exact one you are using. Colab is a hosted Jupyter Notebook service and runs Notebooks which are saved on Google Drive. It also has the property that one can work on the notebook from anywhere (only a web browser is needed).

2.4.3 SORT - Tracking

Simple Online and Realtime Tracking, SORT [3]. This algorithm is able to track multiple 2D objects in a video sequence with the help of a Kalman filters. It is designed for online tacking applications where you only have current and past frames are available. When it was released in 2017, it was ranked as the best open source multiple object tracker on the MOT benchmark 2015 [19]. The library[2] we will be using for this thesis uses detections made by a Faster R-CNN network. Modifications will therefore be made to our Mask R-CNN to output detections in the required format.

2.4.4 CLoDSA - Data Augmenting Augmenting

To improve robustness and reduce overfitting from our Neural Network we will propose several augmenthing techniques. For this task we will use CLoDSA, which is an open-source image augmentation library for object classification, localization, detection, semantic segmentation and instance segmentation. This library will be used to make our dataset bigger as the original dataset is very small for deep learning algorithms.

Chapter 3

Materials and method

This chapter will present the materials and methods used for this thesis. First, we will present the collected data that is used for training the Mask R-CNN. Then, we will present certain pre-processing techniques which we will use before feeding images to the network. The training process will then be explained, before we go through the SORT algorithm and how we modify it for our purpose. Then, we will explain how the behaviour characteristics are calculated. At the end of the chapter we present how we collected a new dataset from Slørdalen fish farm, which we will use to test our setup.

3.1 Data collection

The data used in the training process is a collection of data that I collected during the Specialization Project[28]. This data is a set of underwater videos of smolt from the Slørdalen fish farm.

3.1.1 Dataset

The collected dataset consists of two subsets, one with IR videos and one with visible light(VIS) videos. Both subsets are filmed in the same indoor fish tank. The IR videos were filmed using a waterproof Metaphase Technologies WideBeam LED Spot Light as the only light source, simulating night time. It outputs light at a wavelength of 850nm. The VIS videos were filmed in daylight conditions provided by the fluorescent lamps the facility used at the time. No extra lighting were used for these videos. To capture the underwater videos, we used a FLIR Blackfly S 5.0 MP,22 FPS, monochrome (Black&White) camera with a custom underwater housing. This camera has no IR cut-off filter, which enables us to capture IR videos without having the IR light blocked. For the lens, we used a 16mm C Series VIS-NIR Fixed Focal Length Lens from Edmund Optics. The videos were filmed with 15/18 FPS. They were then split into image sets consisting of 900/1080 images for each video, respectively. A number of random images from different videos were selected to realize the training set. Initially, IR images and VIS images are separated and we will, in

this thesis, only consider the IR sets. Another smaller set of random images were created to constitute the validation set. At last, a test set was created to enable us to check the network performance. Labeling of our data was done using the COCO-annotator tool[5]. This tool creates a dataset with simailar format as the COCO dataset [21]. It outputs a .json file which contains the relevant information about the annotated image such as categories, annotations, licenses etc. Initially, in the Specialization Project we created three different datasets: **Whole-IR**, **Partial-IR** and **Whole-VIS**. The **Whole** datasets consist of images where only whole fish in the image are masked and annotated, while the **Partial** dataset consist of images where all fish, both whole and partial fish are annotated. We decided to only continue with the **Whole-IR** dataset. An example of the annotation process is shown in figure 3.1. For more details about the camera setup and the collection of the dataset we refer to the Specialization Project[28].

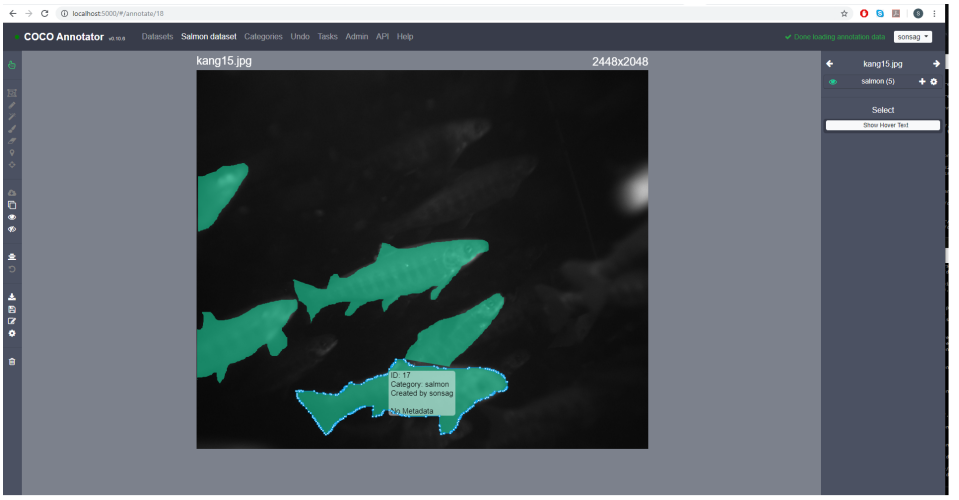


Figure 3.1: Annotating the **Partial-IR** dataset using COCO annotator. Adopted from [28].

3.2 Method

3.2.1 Testing Mask R-CNN in the Specialization Project

As the Mask R-CNN have shown great performance on instance segmentation applications for a long time[25], in addition to the amount resources available, it became the network of choice for our task of segmenting salmon smolt in images. An initial test run on the network was done during the Specialization Project [28]. The initial results showed promise, but was not quite satisfactory. In this thesis we will look to improve upon the network by using some of the improvement possibilities we discussed in the Specialization Project.

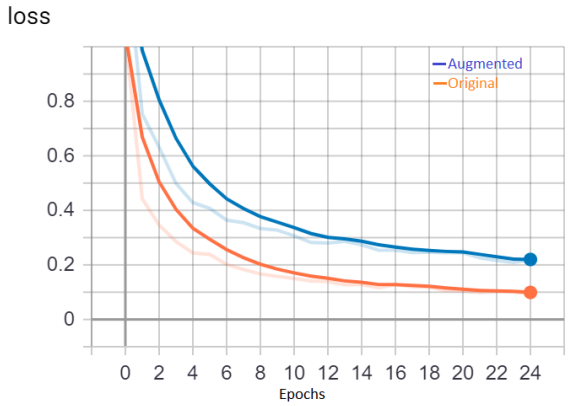
3.2.2 Pre-processing

When training a network, an important aspect is the quality of our data. The higher quality of the data we feed our model, the higher quality the model itself will be. To increase the quality of a dataset, a step called pre-processing is often used. It involves transforming the data in various ways before feeding it into the network. Techniques used for this purpose includes normalization, data centering, shearing, smoothing, thresholding etc. For this thesis, we will explore several pre-processing techniques. First, we will start with data augmenting.

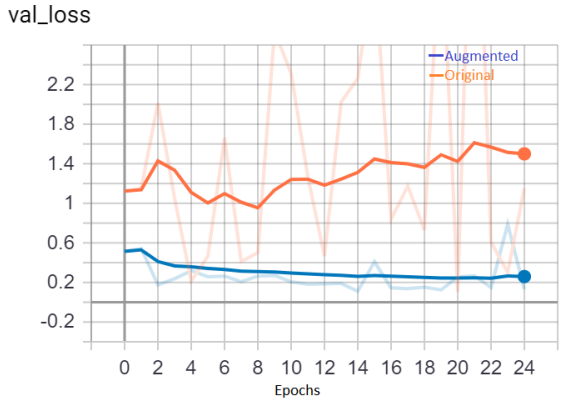
Data Augmenting

As a larger dataset can improve the performance of the network, this is decided as a first step to improve network accuracy. Labeling data with the COCO-annototor tool is a manual and time-consuming process. Therefore, to avoid spending too much time at this step in the thesis, a more time-saving approach called data augmentation is used. First, a certain amount of data is labeled by hand. Then, simple augmentation methods such as flips and rotation are used to increase the dataset. Here we use four rotations,[0,90,180,270], in degrees. Each rotation is also flipped. As the set is doubled for each flip and rotation we get a dataset 8x the size of the original. The Github library called CLoDSA [14] is used to help perform these. It accepts the COCO format as input and outputs the data in the same format. The goal of these augmentations is to achieve better performance on the validation set (lower loss). In addition to increasing the size of the dataset it will also help generalize the network. Many of the images might contain fish which are orientated in a specific direction. This can cause the network to be biased towards that specific orientation of the fish. By flipping and rotating the images, the network will be less sensitive to such properties.

The first runs with the new dataset shows an increased performance in the accuracy of the network. This can be seen in figure 3.2. We see that when the data is augmented it performs better on the validation set while it performs worse on the training set. This is an indication of over-fitting by the network when training on the original images. This is not unexpected as the original set is very small for this type of network architecture.



(a) Training loss.



(b) Validation loss.

Figure 3.2: Training loss vs. validation loss. The curves are the smoothed form of the original losses which is visible in the background. The smoothed graph is an exponential moving average, which is used to smooth out short-term fluctuations and highlight longer-term trends. The blue line represents the augmented dataset, while the orange line represents the original dataset.

Thresholding

Another pre-processing technique we will try is called thresholding. Thresholding is a binary classification of pixels based on a global or local threshold value. Thresholding itself can be looked upon as an image segmentation tool. Instead of using machine learning we can use this technique to segment objects in images. The goal of this process, for this thesis, is to remove unwanted objects and background, highlighting the objects we want to segment. Here, we explore 4 different thresholding methods: global thresholding, adaptive mean tresholding, adaptive Gaussian tresholding and Otsu's tresholding[24] with Gaussian filtering. We will use an example image from our dataset to illustrate the effects these tresholding methods will have. The algorithms uses gray-scale images as input.

Global Thresholding

This is a simple thresholding method, as we for every pixel apply the same threshold value. We will use a binary threshold, meaning that if the pixel value is higher than the threshold value, we will assign it a pixel value of 255(max). If it is lower than that value, we will assign it a pixel value of 0(minimum). The results of this method is shown in figure 3.3.

Figure 3.3: Illustration of the global thresholding method. In our case this method is very sensitive to the threshold value. The optimal threshold value is different for each image, making it unsuited for our application.

Adaptive Mean- and Adaptive Gaussian Thresholding

In the previous method we used a global threshold value. Consequently, with varying lighting conditions the previous method will struggle. Adaptive thresholding uses the region around a pixel to decide its threshold value. This implies that there will be different threshold values around the picture. The Adaptive Mean method uses the mean of the neighbouring area while the Adaptive Gaussian method uses a Gaussian-weighted sum of

the neighborhood around the pixel to calculate the threshold value. The effects of these two methods are shown in figure 3.4 and figure 3.5.

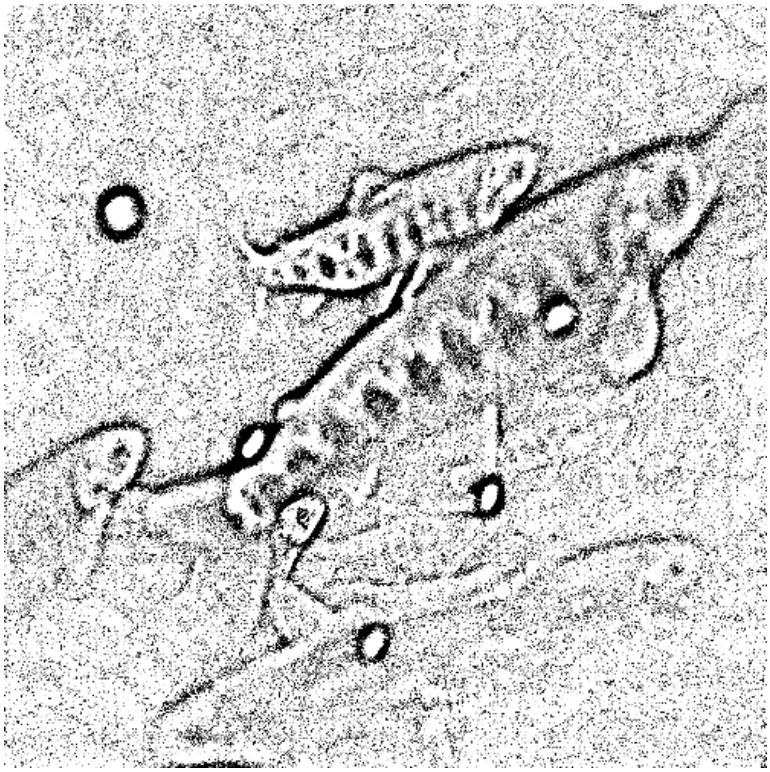


Figure 3.4: Illustration of the Adaptive Mean tresholding. This method very good at extracting the features of the fish in the image. Some noise remains in the image.

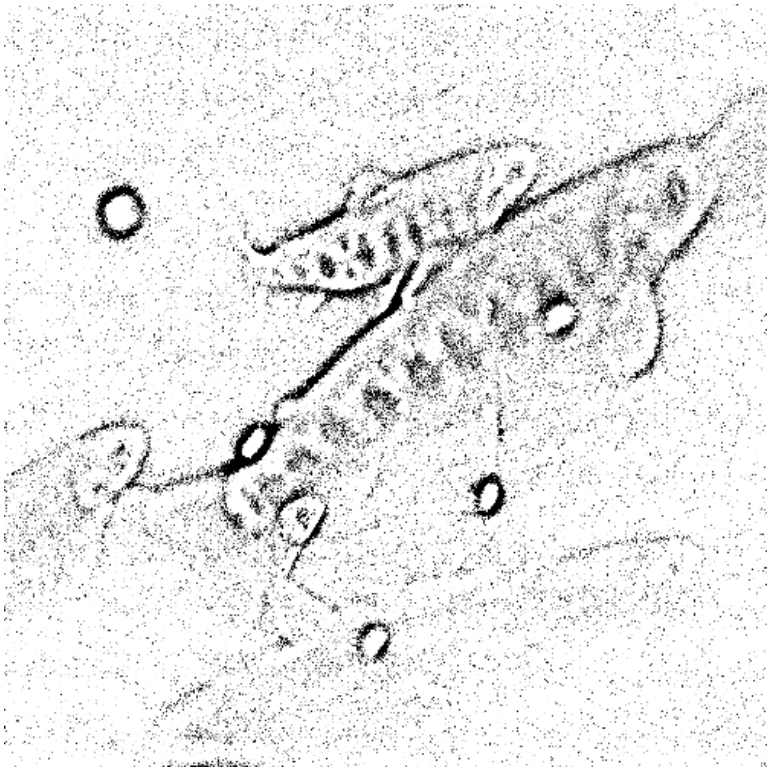


Figure 3.5: Illustration of the Adaptive Gaussian method. This image is very similar to the Mean Method image, but with a little less noise and less distinct features.

Otsu's method

Instead of choosing the threshold value, Otsu developed an algorithm that determines it automatically. It chooses the optimal threshold value based on the image histogram. It works in the way that it searches for the threshold which minimizes the intra-class variance, defined as the weighted sum of variances of the two classes. The weighted sum of variances is given as:

$$\sigma_w^2(t) = \phi_1(t)\sigma_1^2(t) + \phi_2(t)\sigma_2^2(t) \tag{3.1}$$

Where:

$$\phi_1(t) = \sum_{i=1}^{t} P(i), \tag{3.2}$$

$$\phi_2(t) = \sum_{i=1+t}^{255} P(i), \tag{3.3}$$

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{\phi_1(t)}, \tag{3.4}$$

$$\mu_2(t) = \sum_{i=1+t}^{255} \frac{iP(i)}{\phi_2(t)}, \tag{3.5}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} (i - \mu_1(t))^2 \frac{P(i)}{\phi_1(t)}, \tag{3.6}$$

$$\sigma_2^2(t) = \sum_{i=1+t}^{255} (i - \mu_2(t))^2 \frac{P(i)}{\phi_2(t)} \tag{3.7}$$

P(i) is the probability of a gray level in the image histogram. We iterate on t (from 0 to 255) and choose t so that $\sigma_w^2(t)$ is at its minimum.
Before we feed images into the algorithm we use a Gaussian filter to get rid of the noise in the images. The result can be seen in figure 3.6. The reason behind the poor performance of this method lies in the image histogram, which is seen in figure 3.7.

Figure 3.6: Resulting image when applying Otsu's method. This thresholding method has the worst performance. The reason behind this is the image histogram, which we see in figure 3.7. This algorithm wants to find a value between two peaks in the histogram, which makes the variances of the two classes minimal. As can see from the histogram, we only have one peak.

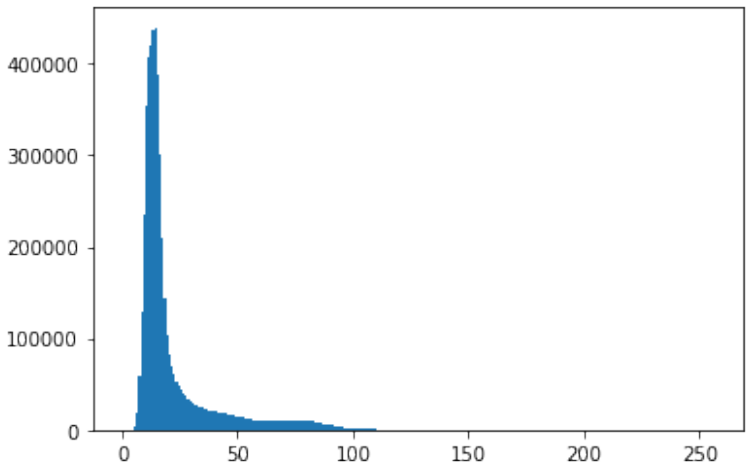


Figure 3.7: Histogram of our example image. As we can see there is only one peak, which makes Otsu's method unusable.

Thresholding conclusion

As we see from the examples provided, tresholding can do a good job at segmenting the fish, especially the adaptive methods. The other methods looses a lot of information in the image. We will do an initial test on our mask R-CNN with the images that are thresholded using the Adaptive Mean method to see how the network performs. Furthermore, in chapter 4.5, we will be discussing a dual stream network architecture approach, which uses both unprocessed images and thresholded images.

3.2.3 Other preprocessing techniques

Before we feed the images to the network we also propose several other techniques. These are intended to make our network more robust when the quality of the input data is lacking. The techniques are:

Sharpen: This will sharpen the image.
Dropout: This will set some pixels to zero.
Elastic deformation: This will apply an elastic deformation given by paper [29].
Histogram equalization: This will apply a histogram equalization to the image.
Salt and pepper noise: This will add salt and pepper noise to the image.
Gaussian noise: This will apply Gaussian noise to the image.
Gaussian blur: This will blur the image, using a Gaussian filter.
The results of applying these techniques can be seen in figure 3.8 and figure 3.9.

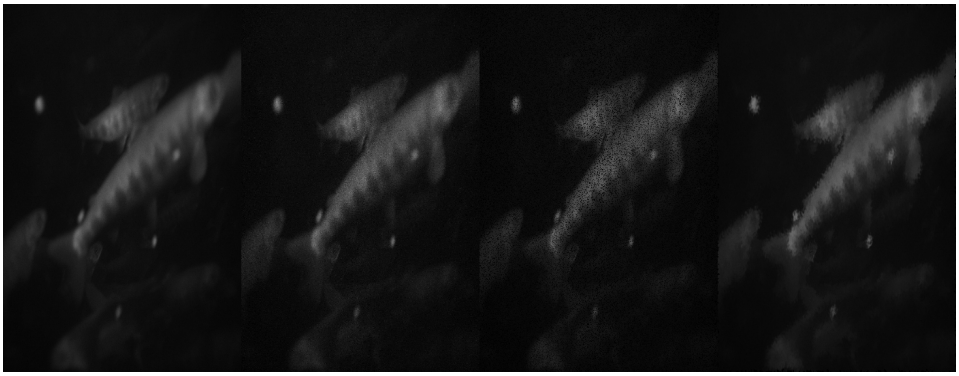


Figure 3.8: Augmentation techniques applied to an image. From left to the right, we have: original, sharpen, dropout, elastic deformation

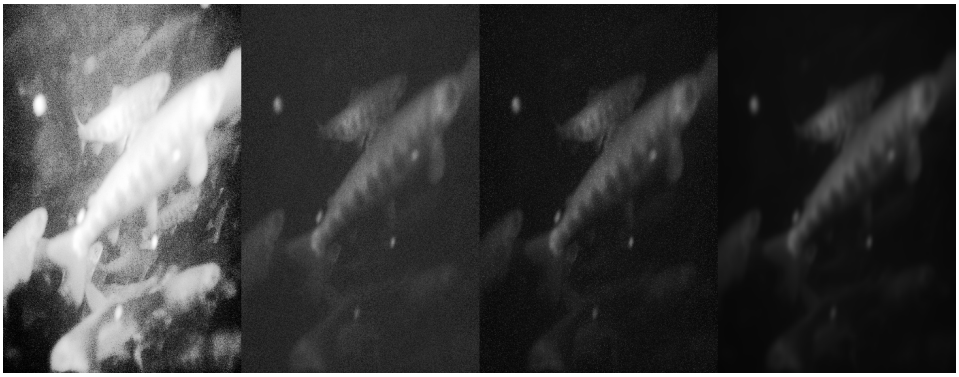


Figure 3.9: Augmentation techniques applied to an image. From left to the right, we have: histogram equalization, salt and pepper noise, Gaussian noise, Gaussian blur.

The network is first trained without using any techniques, and it becomes clear that it performs very well. The reason we want to apply these techniques to our dataset is to improve generalization. Generally, CNN networks seems to have a bias towards texture rather than form[4]. To avoid over-fitting towards high frequency features(patterns that occur a lot) we chose to apply Gaussian noise on some of the images. Gaussian noise (zero mean) has data points in all frequencies, meaning we effectively distort high frequency features. The Gaussian filter was applied on 50% on the training images we fed to the network. The results are seen in figure 3.10. We can see that the data augmentation actually seems to deteriorate the performance of our model. We believe that there are two possibilities for this deterioration. One is that the training set contains enough variation in the data to produce a robust model. The other possibility is that the validation set we use is taken from the same tank with the same conditions, which makes the training data and validation data similar. If we had used a validation set with different conditions, we might have seen a better effect from using the augmentation technique. The reason we do not

explore all augmentation techniques is partly due to the result when applying the Gaussian filter and partly due to the time-schedule of the thesis (the amount of time each training period takes is $\sim$ 8 hours).

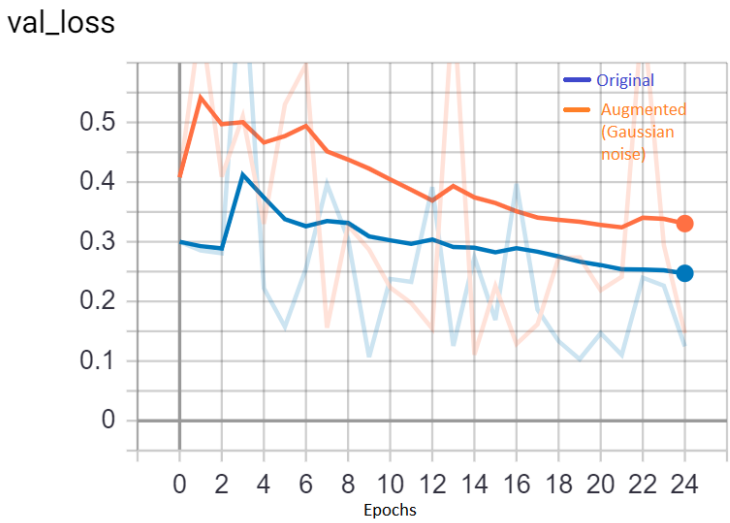


Figure 3.10: Validation loss with Gaussian noise applied on 50% of the training images. The smoothed graph is an exponential moving average, which is used to smooth out short-term fluctuations and highlight longer-term trends. The blue line represents the original dataset, while the orange line represents the augmented dataset.

3.3 Mask R-CNN

As mentioned, the Mask R-CNN [12] is a network intended for object instance segmentation and had state of the art performance on the COCO dataset when the paper was released in 2017. Our project uses this network architecture. The implementation is based on Adam Kellys usage [17] of Matterports implementation of the Mask R-CNN [1], with ResNet101 as backbone. The network is trained using Google Colabs computing services. After initial training on the three datasets in our Specialization Project, the **Whole-IR** dataset provided the most promising results. As time is of the issue, this is the only set that is expanded and considered further for this thesis. It is described in greater detail in the Specialization Project.

3.3.1 Training Process

This chapter will go through the complete process of training the network. Relevant code used will be appended with the thesis.

1. We first import relevant libraries, such as the Mask R-CNN library. In this thesis we use Adam Kellys version which fixed a bug in Matterports original version that

would break the program when loading an existing model. In addition, the file model.py is replaced with a slightly modified version to combat another bug that appeared.

2. Since Google Colab is used, our data is not stored locally, so it has to be loaded to the virtual machine. The training data is stored in Google Drive and imported from there. The data-folder that is loaded consists of a training set, a validation set, and their respective .json files containing all the relevant information, such as categories, annotations etc.

3. A directory to save the different versions of our network is set up, and a pre-trained COCO model is downloaded. This model will be used as a starting point when training the network, instead of starting from scratch. This is the technique called transfer learning, which we mentioned in chapter 2.1.4.

4. Now, we set up all the configurations in the network. These can be tuned and changed to see if it improves performance. A more detailed explanation of which variables that affects our network the most are mentioned in 3.3.2. A full overview of the configuration is seen in figure 3.11.

5. While we use COCO format for our dataset, there are many variations of the format itself. Therefore, a way to use the different variations is set up when the dataset class is defined.

6. Now, the training and validation set is loaded with the dataset class defined in the last step. The model is created in training mode and is initialized with the pre-trained coco weights from step 4. In this step, the network can alternatively be initialized with a set of weights that we have previously trained.

7. Now, we can choose which layers to train as well as adding additional augmentation. We then choose how many epochs the model will train for. Lastly, the training process is started using the chosen configuration.

We evaluate the performance based on the loss (see figure 3.10 and 3.2), which should be as low as possible. This is achieved by minimizing the loss function given by:

$$\mathcal{L} = \mathcal{L}_{cls} + \mathcal{L}_{box} + \mathcal{L}_{mask} \tag{3.8}$$

$\mathcal{L}_{cls} + \mathcal{L}_{box}$ is the loss function of Fast R-CNN and defined as follows:

$$\mathcal{L}_{cls}(p, u) = -\log p_u \tag{3.9}$$

$$\mathcal{L}_{box}(t^u, v) = \lambda[u \geq 1] \sum_{x,y,w,h} L_1^{smooth}(t_i^u - v_i), \tag{3.10}$$

where

$$L_1^{smooth}(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases} \tag{3.11}$$

and

$$\lambda[u \geq 1] = \begin{cases} 1, & \text{if } u \geq 1 \\ 0, & \text{otherwise} \end{cases} \tag{3.12}$$

Symbol	Explanation
p	Discrete probability distribution per RoI, $p = (p_0, ..., p_K)$, over $K + 1$ categories.
u	True class label, $u \in (0, 1, ..., K)$ Background class is labeled $u = 0$ by convention.
λ	Hyper-parameter that controls the balance between the two task losses. $\lambda = 1$ in the original paper.
t^u	Predicted bounding box. $t^u = (t^u_x, t^u_y, t^u_w, t^u_h)$. The parameterization of t^k is given in[11].
v	True bounding box. $v = (v_x, v_y, v_w, v_h)$.

$\mathcal{L}_{mask}$, which is new for Mask R-CNN is defined as the average binary cross-entropy loss:

$$\mathcal{L}_{mask} = -\frac{1}{m^2} \sum_{1 \leq i,j \leq m} [y_{ij} \log(\hat{y}^u_{ij}) + (1 - y_{ij}) \log(1 - \hat{y}^u_{ij}))], \tag{3.13}$$

where u is the ground truth class, m^2 is the mask generated by the mask branch in the Mask R-CNN, y_{ij} is ground truth value of a cell within the mask and $\hat{y}^u_{ij}$ is the predicted value of the same cell for class u.

3.3.2 Network variables

To increase the accuracy of the network further, tuning of the network variables is tried. As tuning many of the variables is a somewhat trial and error process, the network is trained multiple times with different combinations of variables. Of the variables that was tuned, RPN_ANCHOR_SCALES is found to have a great impact on how well the network performs. This is the length of the square anchor side in pixels. Therefore, RPN_ANCHOR_SCALES is changed to better match the pixel size of salmon in images. The most notable variables that are tuned are RPN_ANCHOR_SCALES, TRAIN_ROIS_PER_IMAGE, which is the number of of RoIs per image to feed to the classifier/mask heads and MAX_GT_INSTANCES, which is the maximum number of ground truth instances to use in one image. The images don't contain a high number of whole smolt. Therefore, this number were set lower than the original value. The specific settings of the network can be seen in figure 3.11.

```
Configurations:
BACKBONE                       resnet101
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     1
BBOX_STD_DEV                   [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE         None
DETECTION_MAX_INSTANCES        100
DETECTION_MIN_CONFIDENCE       0.7
DETECTION_NMS_THRESHOLD        0.3
FPN_CLASSIF_FC_LAYERS_SIZE     1024
GPU_COUNT                      1
GRADIENT_CLIP_NORM             5.0
IMAGES_PER_GPU                 1
IMAGE_MAX_DIM                  512
IMAGE_META_SIZE                15
IMAGE_MIN_DIM                  512
IMAGE_MIN_SCALE                0
IMAGE_RESIZE_MODE              square
IMAGE_SHAPE                    [512 512   3]
LEARNING_MOMENTUM              0.9
LEARNING_RATE                  0.001
LOSS_WEIGHTS                   {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0, 'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0, 'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE                 14
MASK_SHAPE                     [28, 28]
MAX_GT_INSTANCES               8
MEAN_PIXEL                     [123.7 116.8 103.9]
MINI_MASK_SHAPE                (56, 56)
NAME                           salmon
NUM_CLASSES                    3
POOL_SIZE                      7
POST_NMS_ROIS_INFERENCE        1000
POST_NMS_ROIS_TRAINING         2000
ROI_POSITIVE_RATIO             0.33
RPN_ANCHOR_RATIOS              [0.5, 1, 2]
RPN_ANCHOR_SCALES              (64, 128, 256, 512)
RPN_ANCHOR_STRIDE              1
RPN_BBOX_STD_DEV               [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD              0.7
RPN_TRAIN_ANCHORS_PER_IMAGE    256
STEPS_PER_EPOCH                600
TOP_DOWN_PYRAMID_SIZE          256
TRAIN_BN                       False
TRAIN_ROIS_PER_IMAGE           100
USE_MINI_MASK                  False
USE_RPN_ROIS                   True
VALIDATION_STEPS               50
WEIGHT_DECAY                   0.0001
```

Figure 3.11: Configuration for Mask R-CNN.

3.3.3 Generating the input to SORT

To be able to classify the behaviour of the fish, looking at the movement patterns is chosen as the element to explore in this thesis.
When the network is tuned to a satisfactory level where it is able to segment smolt in the images consistently, a new dataset is created. This new dataset contains information about the placement of each detected smolt in the image(the bounding box), a confident score(how certain the network is that it has detected a fish) and a frame ID which connects the data to each image. In addition it contains information of the angle of the fish, i.e the orientation in degrees from 0-360. This is not a property which Mask R-CNN calculates, but a property we have added for this thesis. To calculate the angle, we first calculate the raw moments for a gray-scale image with pixel intensity $I(x, y)$ as:

$$m_{ij} = \sum_{x} \sum_{y} x^i y^y I(x, y) \tag{3.14}$$

The input we give the function is a binary masking of one fish at a time as seen in fugure 3.12.

Figure 3.12: Illustration of the input image to equation 3.14.

We then fit an ellipse over the masked fish. The reason we do this is because the body of a salmon resembles an ellipse. We calculate the ellipse with the following equations based on the paper *Fitting an ellipse to an arbitrary shape: implications for strain analysis* [23]:

$$x_{center} = \frac{m_{10}}{m_{00}} \tag{3.15}$$

$$y_{center} = \frac{m_{01}}{m_{00}} \tag{3.16}$$

$$u_{00} = m_{00} \tag{3.17}$$

$$u_{20} = \frac{m_{20}}{m_{00}} - \frac{m_{10}^2}{m_{00}^2} \tag{3.18}$$

$$u_{20} = \frac{m_{02}}{m_{00}} - \frac{m_{01}^2}{m_{00}^2} \tag{3.19}$$

$$u_{11} = \frac{m_{11}}{m_{00}} - \frac{m_{10}m_{01}}{m_{00}^2} \tag{3.20}$$

$$\Delta = \sqrt{4u_{11}^2 + (u_{20} - u_{02})^2} \tag{3.21}$$

$$\theta = \frac{1}{2}\arctan(\frac{2u_{11}}{u_{20} - u_{02}}) \tag{3.22}$$

$$width = \sqrt{6(u_{20} + u_{02} - \Delta)} \tag{3.23}$$

$$length = \sqrt{6(u_{20} + u_{02} + \Delta)}, \tag{3.24}$$

where u_{ij} represents the central moments of order $i + j$. If we now draw the ellipse with the calculated center coordinates, the angle, the width and the length we get a resulting image as seen in figure 3.13.

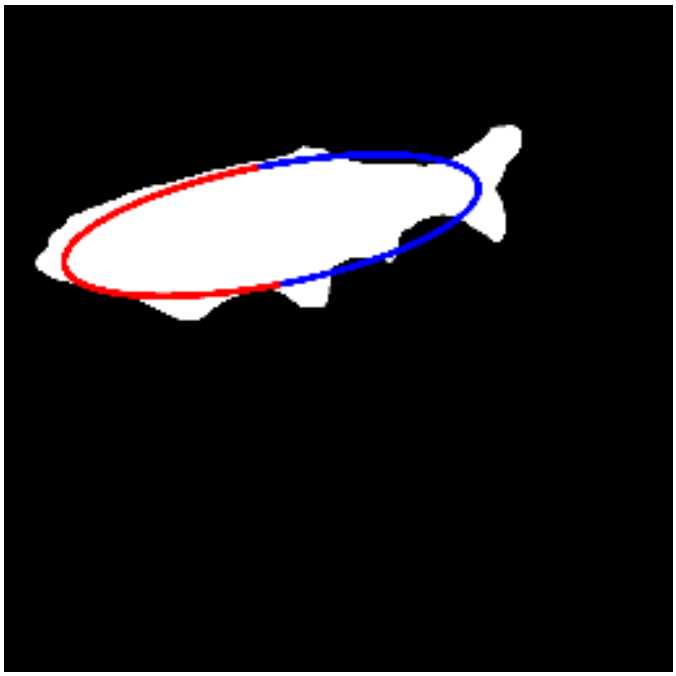


Figure 3.13: Illustration of ellipse drawn over the masked fish.

To find the front- and tail region of the fish we fill the ellipse and split it along the minor axis. As the fish is thinner in the tail region, we count the overlapping pixels from the ellipse we placed on the masked fish. The region with fewer overlapping pixels is defined to be the tail region of the fish. From this we can determine which angle the head points. The original image is seen in figure 3.14. With this technique it is also possible to segment certain parts of the fish for further analysis which we will discuss further in chapter 4.4.

Figure 3.14: Original image of fish we draw an ellipse on.

The data we generate (frame, angle, position, etc.) is stored in a text document where each line represents one detected fish. It will be used as input to SORT to track the fish in image sequences.

3.4 Multiple object tracking

In this chapter we will explain how we will track salmon in video sequences. We will also assign attributes such as velocity, acceleration and moving direction to each salmon.

3.4.1 Kalman Filter

The MASK R-CNN is able to segment fish at a decent level. However, it is not able to tell us anything about the behaviour of fish. Furthermore, it is not able to tell if the fish it detects is the same fish as it detected in the frame before. Movement patterns are our main focus for this thesis, and therefore tracking fish between frames is a key factor in being able to describe these movement patterns. A widely used technique for this purpose is the Kalman filter, and the choice of algorithm to track the smolt is the SORT algorithm, described in the paper *SIMPLE ONLINE AND REALTIME TRACKING* [3]. The estimation model they use approximate the inter-frame displacement of each object with a constant velocity model, independent of other objects and the camera motion. They

model the state of each object as:

$$\mathbf{x} = [u, v, s, r, \dot{u}, \dot{v}, \dot{r}] \tag{3.25}$$

where u and v represents the horizontal location of the center of each target, while s and r represent the area and the aspect ratio of each target. When a detection is made, the detected bounding box is used to update the target state. Then, by using a Kalman filter framework, the velocity components are optimally calculated. If there is no detection, a prediction of the state is made using the linear velocity model. To assign detections to an existing target, they use an assignment cost matrix that is optimally calculated using the Hungarian algorithm. The data we export from the Mask R-CNN, which we mentioned in the previous chapter, is used as the true state in this algorithm.

As we now have an algorithm to track objects from frame to frame, we feed the Mask R-CNN generated data to the SORT framework. Since all the information needed is saved in the text document(displaying the images is an optional property), this a fast algorithm, suited for real-time applications. We see that the algorithm is able to track salmon smolt at a high level in the images. However, there are certain limitations in the current framework when it comes to describing the movement of each fish. Therefore, we have to make several modifications in order to output attributes such as velocity, acceleration and vertical/horizontal movement.

3.4.2 Modifying the SORT algorithm

To describe the velocity of each fish, we will use the average speed over the last n frames, where n is a variable of our choosing. We use the standard equation for calculating the average velocity:

$$\overline{v} = \frac{\Delta x}{\Delta t}, \tag{3.26}$$

where Δx is the displacement and Δt is the change in time, which in our case will be n. Δx is calculated as the distance between the center point of two bounding boxes (which represents the same object) after n frames. We save each object position in an array over the last n frames and continuously calculate the velocity of each object. We create a velocity array, which saves the velocities over n frames, which we will use to calculate the acceleration. We save both the total velocity as well as the x-component and the y-component. It is important to note that we actually have to detect the same fish over n frames before we can begin to calculate velocity. Calculating the direction each fish is moving comes naturally as we can look at which way the center point of each bounding box moves. Directions are divided into horizontal and vertical direction. If the velocity in a direction lower than a given threshold, the fish is said to be stationary. Horizontal direction is divided into the categories: left, right and stationary, and vertical direction is divided into the categories: up, down and stationary. In calculating the acceleration we use the formula for average acceleration:

$$\overline{a} = \frac{\Delta v}{\Delta t}, \tag{3.27}$$

where Δv is the change in velocity and Δt is the change in time (n). We then calculate the acceleration using the saved velocities. As for the velocity, we calculate total acceleration, and the x- and y component. The reason we also calculate the x- and y component is that we believe that it can prove helpful when describing the behaviour of the fish. Here, it is also important to note that we need to detect the fish in n x 2 frames before we start to calculate the accelerations.

There is, however, a problem with our approach of calculating velocity as we only calculate the relative velocity with respect to the captured image. A fish that is closer to the camera will appear larger than one that is further away. Naturally, if they are the same size and swim with the same velocity, the fish that is the closest to the camera will appear to swim faster than the fish that is further away.

To overcome this problem we first made the assumption that each fish have the same size. From a geometric perspective we know that if the distance from an object to an observer is doubled, the observer will see the object as twice as small in both vertical and horizontal direction, meaning the observed area will be $\frac{1}{4}$ of the original area. The area is inversely proportional to the squared distance:

$$d^2 \propto \frac{1}{A}, \tag{3.28}$$

where d is the distance to the object and A is the area of the object. The implications of this is that if two objects of equal size moves with the same velocity, but one is twice as far away, it will appear to move twice as slow. However, if we multiply both velocities by their respective distance to the observer we will end up with the same number. In our case we have no way to calculate the distance to each fish, so instead we multiply by $\frac{1}{\sqrt{A}}$. The area of the fish is a new property we will export from the Mask R-CNN algorithm. An illustration of the described problem is seen in figure 3.15.

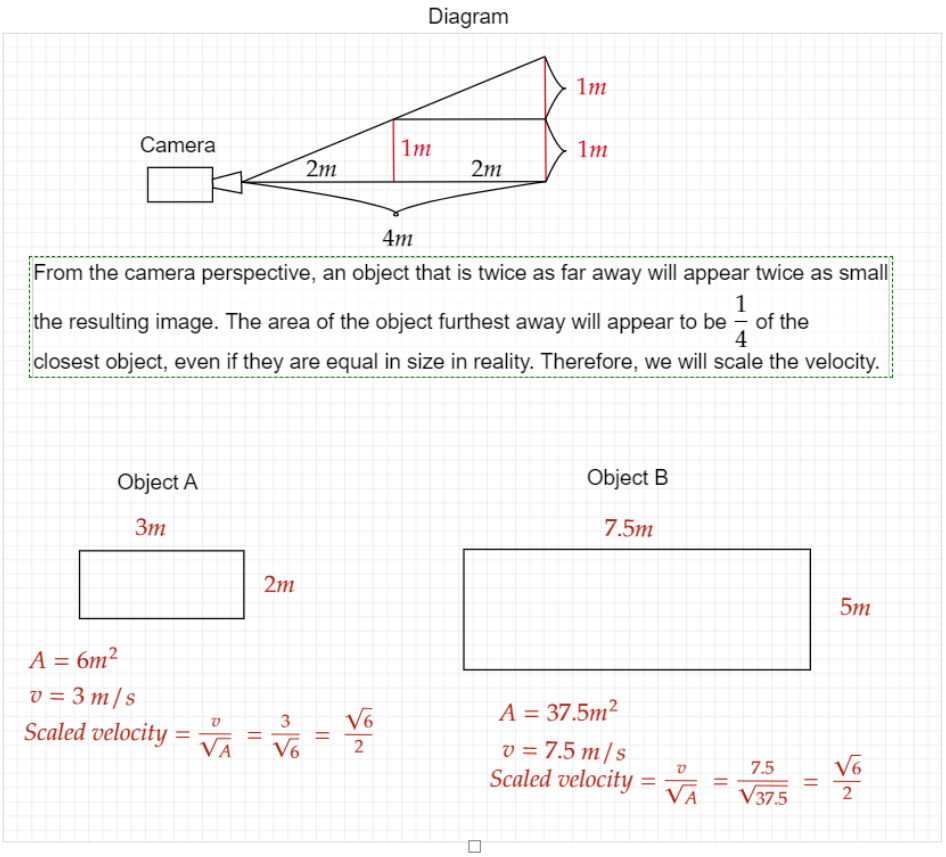


Figure 3.15: Illustration of the velocity problem. Object A and Object B are the observed size of two similar sized objects. They move with the same velocity, which is one body length each second. If the body length in reality is $4m$, we have to multiply by the constant $4\frac{2}{\sqrt{6}}m$ to find the real velocity, which is $4m/s$.

The exported properties for each detected object in the SORT algorithm is as follows:

1. **Frame ID**: Tells us which frame in the video we are referring to.
2. **Fish ID**: Tells us which fish in the image we are referring to.
3. **x-position**: Tells us the x-coordinate of the center point of the bounding box around the fish.
4. **y-position**: Tells us the y-coordinate of the center point of the bounding box around the fish.
5. **y-velocity**: Tells us the x-component of the velocity.
6. **x-velocity**: Tells us the y-component of the velocity.

7. **Total velocity**: Tells us the total velocity.
8. **x-acceleration**: Tells us the x-component of the acceleration.
9. **y-acceleration**: Tells us the y-component of the acceleration.
10. **Total acceleration**: Tells us the total acceleration.
11. **x-direction**: Tells us whether the fish moves left or right in the image.
12. **y-direction**: Tells us whether the fish moves up or down in the image.
13. **Area**: tells us the masked area (not the area of the bounding box), in pixels, of the fish we are referring to.
14. **Fish angle**: Tells us the angle of the fish (which way the head is pointing).
15. **Prediction**: Tells us whether a prediction or the true state was used in the current frame (true/false).

These calculations are only done in the 2D-plane, meaning we have no way to tell of the fish is moving towards or away from the camera. The velocities and accelerations calculated also have no physical meaning as they are. To find the actual velocity one would have to multiply by a constant based on real measurements. It is also important to note that there is a current in the water, meaning the fish can appear to stay still but in reality swim with a certain velocity. The goal of this thesis, however, is not accurately describe the true velocity or acceleration of each fish, but rather to look for differences in the generated data based on the situation in the fish tank.

3.5 Testing our complete algorithm

This section will explain how we will test our complete setup for classifying simple behaviour classes.

3.5.1 Second trip to Slørdal

In order to see how well our setup is able to detect different situations in the fish tank we need new videos, which represents different salmon behaviour. As we are not able to dramatically change the behaviour of the fish in any way, such starving or making the fish sick, we propose three simple categories: feeding behaviour, normal behaviour and spooked behaviour.

Equipment and setup

The camera and lighting equipment used in this trip is similar to what we used in our Spesialization Project. However, as the salmon does not feed at night we need to film in normal lighting. Therefore, we add a UV/VIS cut-off filter so that we end up with IR videos.

IR Pass filter Model:M25.5 x 0.5 Mounted UV/VIS Cut-Off, IR Pass Filter (R-72). These filters absorb most of the ultra violet and visible region. Our filter lets through waves of length 720nm and above. To film approximately the same size of salmon we had to film in a different part of the facility than in the preceding project. The smolt were now in much larger fish tanks than compared to the Specialization Project. The lens aperture we ended up with was both f1.4 and f2.8. Ideally we want this number a little larger (smaller aperture) so that our focus range increases, but due to the difficult lighting capabilities(only one IR-lamp), we had to keep it relatively small. The IR-lamp was placed on top of, and also submerged together with the camera rig. This is a slightly different setup than what we used in the Specialization Project. This was due to the nature of our working environment, which provided it impossible to hold the lamp in the same way as in the project. In the software SpinView, we also changed the variables Gain and Gamma to $\sim$36 and $\sim$0.9 to increase the apparent brightness and luminance in the videos. The shutter speed was set to automatic. The general goal was to capture similar videos to the ones we collected in our Specialization Project. We believe we achieved that goal and actually increased the quality of the images, especially in terms of focus. An image from the new video set can be seen in figure 3.16.

Figure 3.16: Image from the new dataset.

Feeding behaviour, normal behaviour and spooked behaviour

To capture the natural response in the three different situations we first had to familiarize the fish to the camera rig. After the rig was submerged we waited 30 minutes to let the fish enter their natural state. The videos were filmed at 18 FPS, and we used the same compression as in the preceding project to save both disk space and preventing our RAM to run out of memory. The rig was placed in a position which allowed us to capture all three situations from the same spot.

Feeding: We stared filming when the food entered the water, leaving it passing by the camera, before stopping the recording after 20 seconds. The salmon were automatically fed a little amount of food approximately every 2 minutes. This means that the salmon will not be particularly hungry when the food is dropped. Looking at the videos, there were no clear sign that the fish were behaving differently. Even when looking from above the water, having an overview of the whole tank, it was difficult to spot changes in behaviour when the food was dropped in. Therefore, before we test our algorithm we form the hypothesis: Differentiating between normal behaviour and feeding behaviour will be particularly tricky for our algorithm, unless we have much more data.

Normal behaviour: Normal behaviour was filmed for 20 seconds between the automatic feeding. Nothing special is happening within the fish tank when these are filmed.

Spooked: In order to film this behaviour we let the fish enter their natural behaviour, before quickly submerging a stick next to the camera. These videos were also filmed for 20 seconds. Looking at the videos, this behaviour is easily distinguishable from the normal behavior as when the stick is submerged, all the salmon quickly disappear from the camera view, before they return some time after the stick is gone. As a result, we believe that this behaviour is easier to spot for our algorithm than the feeding behaviour.

3.5.2 Limitations

Since Slørdal fish farm is a commercial facility there were certain limitations to how we could affect the fish and create different situations in the tank. Longer lasting situations in the fish tank would create an interesting addition to the collected dataset. An example is data from a tank where the fish is not fed for a longer amount of time. Other examples could be data with different lighting- or temperature conditions. Collecting such datasets is a component in the suggestions for future work, which we discuss further in chapter 4.5.

3.5.3 Algorithm workflow

Now that we collected the videos, we will test our setup, where the complete workflow is as follows:

1. Extract the images from the video sequences.
2. Feed the images to the Mask R-CNN.

3. Extract image data from the Mask R-CNN
4. Feed the extracted data to our multiple object tracker, SORT.
5. Calculate movement statistics such as velocity, position, acceleration and direction.
6. Plot the collected statistics to look for difference in behavior in our three video classes: normal, feeding and spooked.

We will discuss the results of our setup in the next chapter.

Chapter 4

Results and discussion

We will divide this chapter into five parts. In part 1 we will discuss the performance of the Mask R-CNN. In part 2 we will discuss the performance of the SORT algorithm, in part 3 we will discuss the plotted statistics, in part 4 we will discuss a preliminary exploration in finding the tail beat frequency of a fish, and in part 5 we will discuss the overall performance and future work.

4.1 Mask R-CNN results

Compared our Specialization Project, we can see a great improvement in how well the Mask R-CNN segments and detects salmon in each image. The reason behind this is partly due to better tuning of the network parameters/variables and partly due to a larger dataset. A comparison between the loss of the old- and new dataset was shown in figure 3.2 in chapter 3.2.2.

For the most part, the network does a good job at masking the fish, as seen in figure 4.1. However, the network still suffers from problems where it fails to mask the fish as seen in figure 4.2.

Figure 4.1: Illustration of good masks.

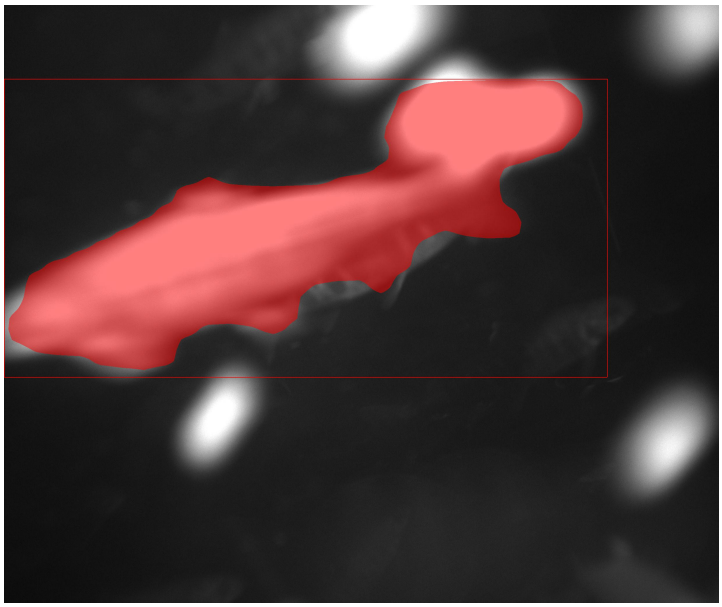


Figure 4.2: Illustration of a failed segmentation by an early version of our network. There are feed pellets in the image that disturbs the masking.

To overcome the problem of bad masks, we set the parameter

DETECTION_MIN_CONFIDENCE to 0,996. This is a measure of how sure the network is that it has detected a fish. By setting this parameter higher we exclude several bad masks when exporting the generated data. The downside, however, is that some good masks also will disappear.
While the overall performance of the network is good, we could still see improvements. A much larger dataset (10 000+ images) would be ideal for improving the network further and combat the over-fitting problem, which particularly occurs with small datasets. More tuning of network variables could also be beneficial for the performance. Generally improving the quality of the dataset would also be a way to increase the network accuracy. Yet, these improvements requires a lot of time and are infeasible with this thesis' time schedule.

In addition to our original dataset, we also do a test on thresholded images, which shows promising results for segmenting the images. In terms of loss, the two methods preformed close to the same, and the issue of bad masks is also present here. As for the original set, we can use the technique with the variable DETECTION_MIN_CONFIDENCE to overcome some of the bad masks. Illustrations of segmenting using thresholded images are shown in figure 4.4 and figure 4.3.

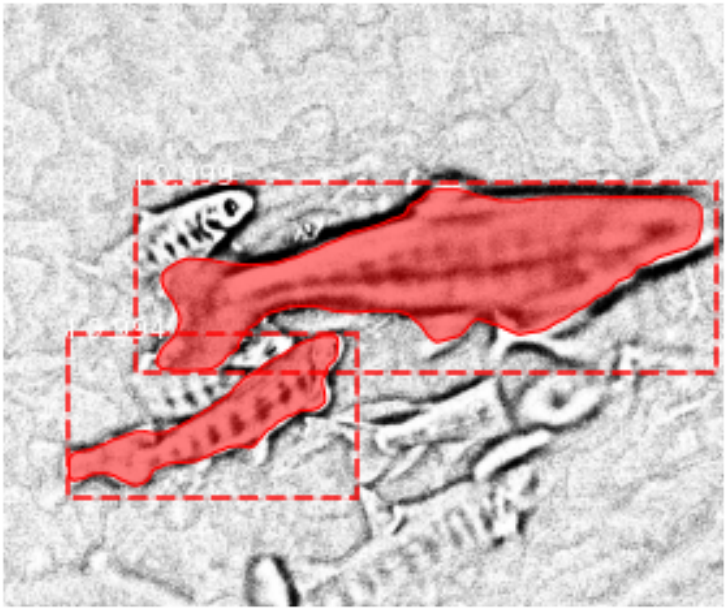


Figure 4.3: Illustration of a bad segmentation when using thresholded images.

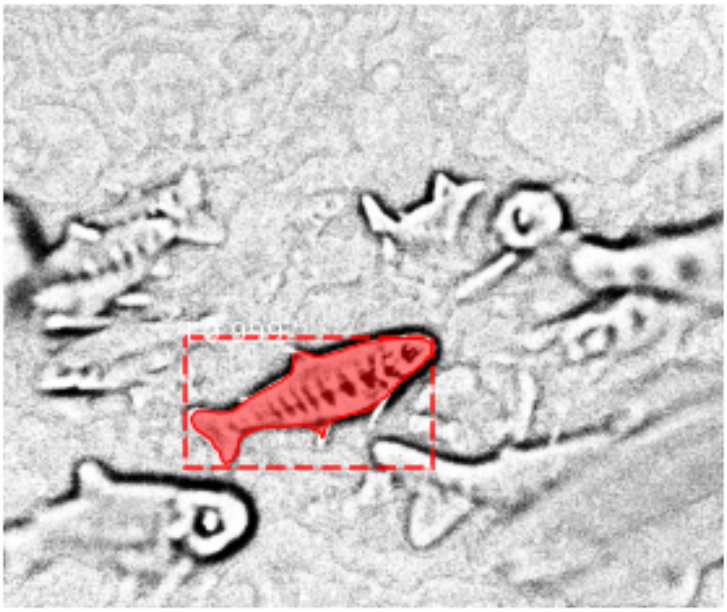


Figure 4.4: Illustration of a good mask when using thresholded images.

4.2 SORT performance

This algorithm does an excellent job at tracking the detected objects. A visualization of the interface is shown in figure 4.5. While it does a great job at tracking objects, it needs stable detections from the Mask R-CNN, meaning the network has to mask the same fish multiple frames in a row. This is usually not a problem when the masked salmon stays within the frame. However, the captured angle is quite small, meaning that fish that would partially or completely go outside the frame would not be tracked anymore. The same would also happen if an untracked fish swims in front of a tracked fish. A solution to this problem could be to adjust the parameter for how many skipped detections we allow. However, for this thesis, this parameter is set to 1, meaning we allow one skipped detection. A prediction with the help of a Kalman filter is then used as the position. The reason we want to keep it quite low is to avoid false tracking.
To allow calculations of velocity we also need to track the fish for 5(n) frames to find the average velocity over these frames. 5 more frames is needed for the acceleration calculations. These numbers can be adjusted, but for this thesis they are set to 5 to make the calculations somewhat robust to noise.

When the visual interface is not running, we are able to run the whole modified SORT algorithm in real-time, achieving 76.9 FPS, making it suitable for real-time applications. It stores the generated statistics in a separate text file for each video sequence.

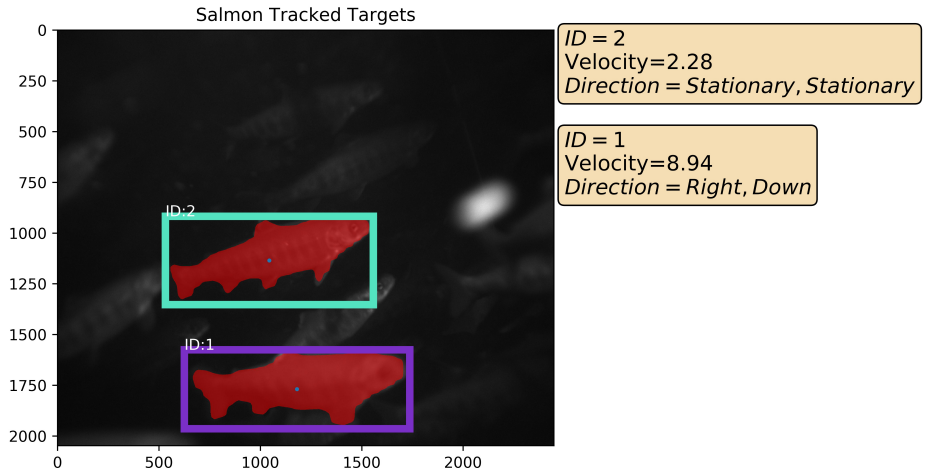


Figure 4.5: Illustration of the visual interface when using the SORT tracker. This image is taken before we added acceleration, area and angle.

4.3 Movement statistics

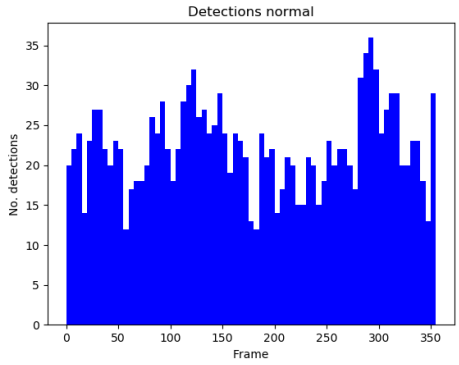
Based on the generated statistics by the modified SORT algorithm, we plot the data for each class: normal, feeding and spooked. In figure 4.6 we see the number of salmon detected in the frames. We use a step of 5 frames to make the graph smoother and easier to read. As we can see, there is no visible difference between the feeding and normal class. However, we see a large difference from the spooked class. There is next to no detected fish after the fish is spooked. We have used 5 videos of 20s for the feeding and normal class, while we have used 10 videos for the spooked class. The reason we have used 10 videos for the spooked class is that we want to make sure we capture the change in behaviour. The fish disappears in a split second. This means that if we do not track a fish just as it is spooked, we are not able to detect the change in behaviour. Looking through the masked images, we see that this problem happens sometimes. To enable our setup to better spot the sudden change in behaviour we decided to use more videos.

What we look for in the statistics is deviance from the normal behaviour. We do not look at individual statistics, but rather shoal statistics. When it comes to acceleration and velocity we expect higher velocities and accelerations in the spooked class. Deviance can also come from the direction the fish swims or the angle of the fish. Usually, the head of the fish is pointing towards the current and detections where the fish points away from the current could indicate an anomaly. This method of looking through the plotted data is visual and manual approach. Later, we will discuss possibility of adding automatic anomaly detection in the future.

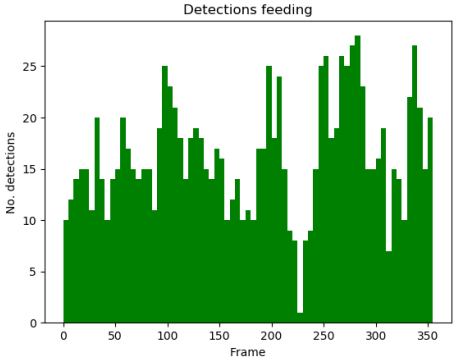
Statistic	Normal behaviour	Feeding behaviour	Spooked behaviour
Number of detections	1600	1204	1115
Number of velcoity detections	1236	906	763
Number of acceleration detections	973	735	528
Average velocity	2.95	2.67	4.60
Average acceleration	0.36	0.30	0.46
Number of apparent movements in x-direction	Left: 97 Right: 258 Stationary: 881	Left: 53 Right: 151 Stationary: 702	Left: 216 Right: 136 Stationary: 411
Number of apparent movements in y-direction	Up: 84 Down: 26 Stationary: 1126	Up: 51 Down: 26 Stationary: 829	Up: 124 Down: 44 Stationary: 595
Cases handled by Kalman filter	27	30	21

Table 4.1: Statistics generated from the SORT algorithm.

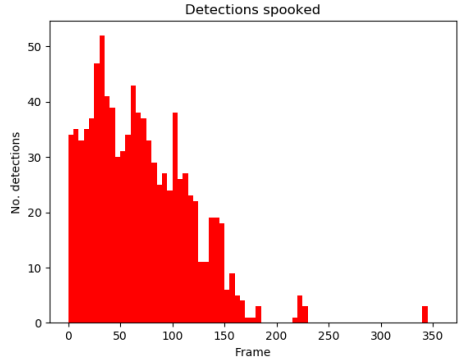
The plotted velocities and accelerations of each class is shown in figure 4.7 and figure 4.8. We also plot the average velocity and acceleration over the last 5 frames to make the graphs easier to read when looking changes in behaviour. They are seen in figure 4.9 and figure 4.11. Distributions of velocities is seen in 4.10 and distributions of accelerations is seen 4.12. Again it is hard to spot a difference between the normal and feeding class. One could say that the feeding behaviour is more uneven than the normal behaviour, but without more data it is impossible to say. The clear difference again is seen with the spooked class. We can clearly see when the fish are spooked both from the acceleration- and velocity plots. The time at which they are spooked in each video may vary a little, but effort was put into spooking them at the same time in each video. Looking at angles, i.e the angle direction the fish head points, we can not spot a clear difference between the classes. The angles are seen in 4.13. A summary of other statistics is seen in table 4.1. In this table we see that the apparent movement of the fish dominated to the right side, except for the spooked class, which is another indication of the spooked class behaving different. From the angles in figure 4.13, we see that most fish has their head pointed in the left direction. The reason the normal class and the feeding class have a higher amount of apparent movements to the right side, when most angles are pointing towards left, is due to the current in the water, which flows to the right side. When the fish is pointed to the right it is much easier to swim with the current and trigger the threshold for a velocity higher than stationary.
The total number of detections refers to the count of every tracked object in each frame over the total number of frames (does not necessarily mean different ID). Stationary refers to a velocity of less than 3. This number can be adjusted in the SORT algorithm.



(a) Normal detections.

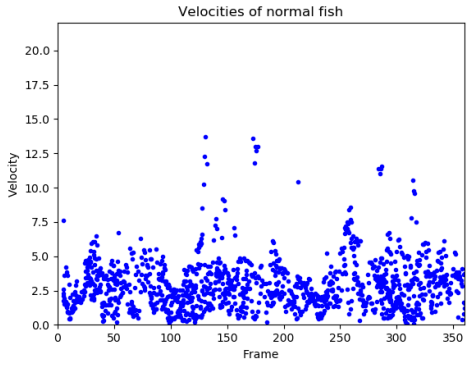


(b) Feeding detections.

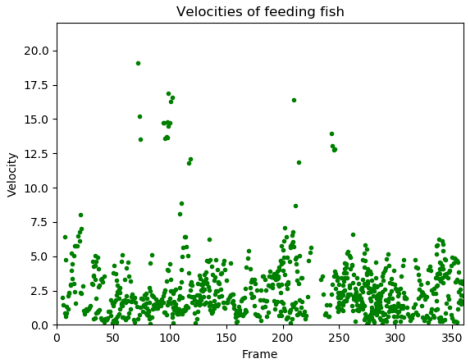


(c) Spooked detections.

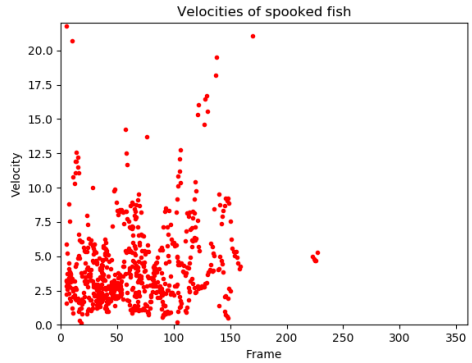
Figure 4.6: Comparison between the number of detections over the last 5 frames for each class.



(a) Normal Velocity.

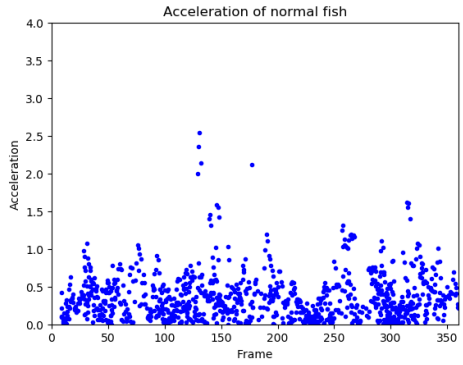


(b) Feeding velocity.

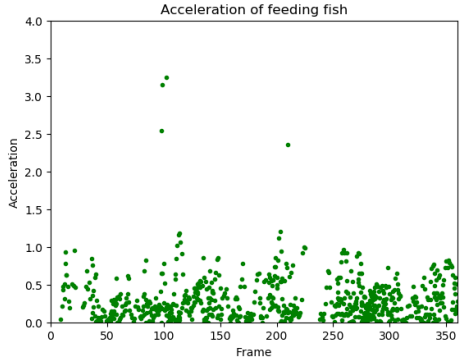


(c) Spooked velocity.

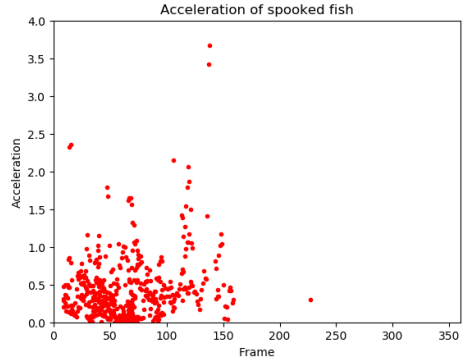
Figure 4.7: Comparison between the velocity of each class.



(a) Normal acceleration.

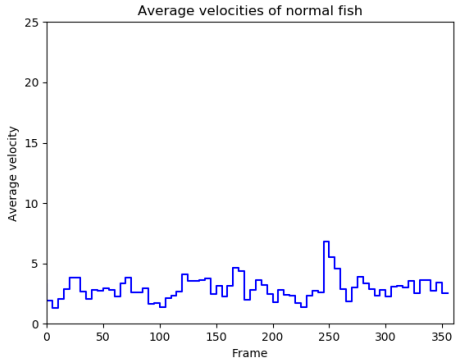


(b) Feeding acceleration.

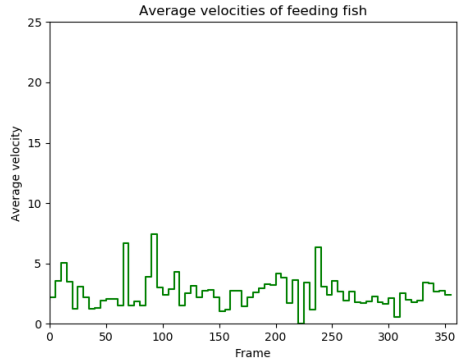


(c) Spooked acceleration.

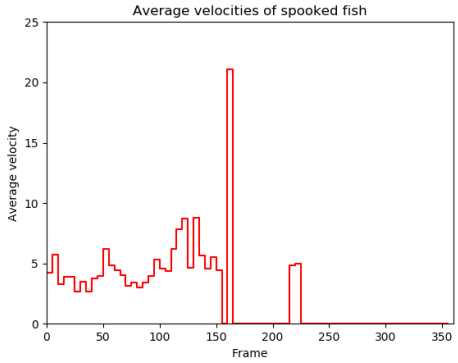
Figure 4.8: Comparison between the acceleration of each class.



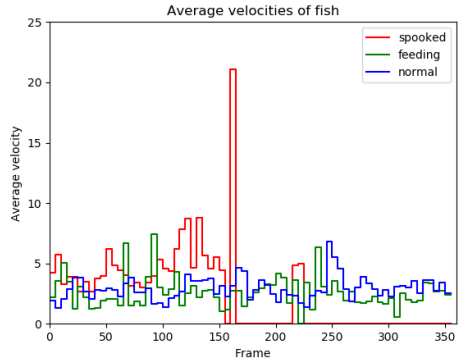
(a) Average velocity, normal.



(b) Average velocity, feeding.



(c) Average velocity, spooked.



(d) Average velocity in the same plot.

Figure 4.9: Comparison between average velocity over every detection in the last 5 frames of each class.

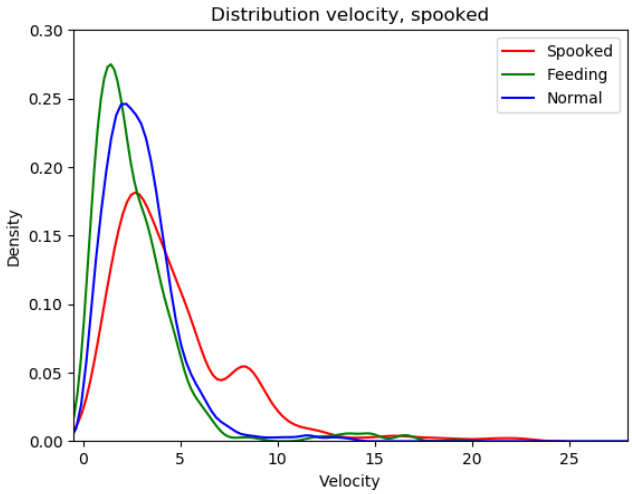
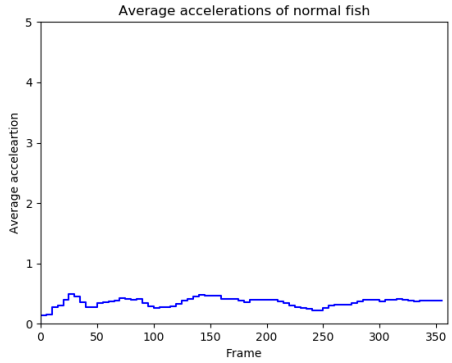
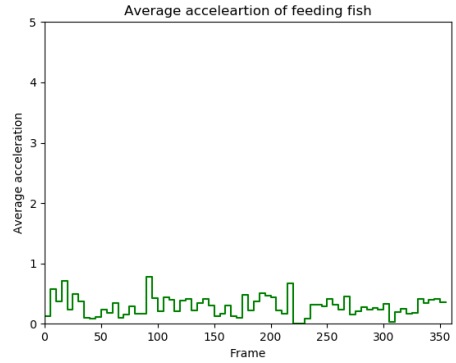


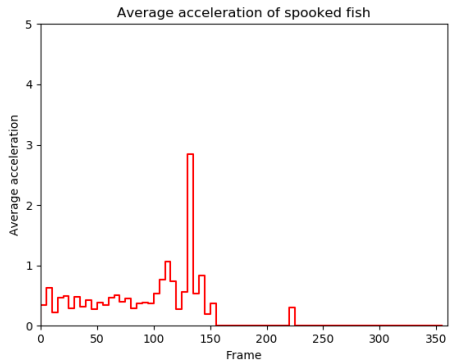
Figure 4.10: Distributions of velocities. Here we can see a clear difference between the spooked class and the two other classes.



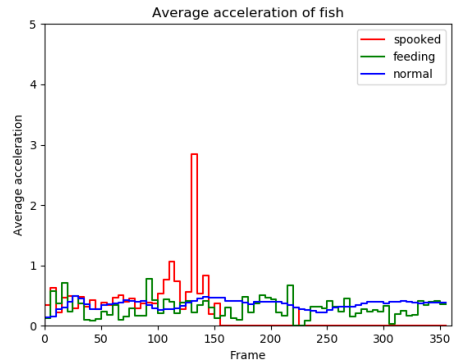
(a) Average acceleration, normal.



(b) Average acceleration, feeding.



(c) Average acceleration.



(d) Average acceleration in the same plot.

Figure 4.11: Comparison between the average acceleration over every detection the last 5 frames of each class.

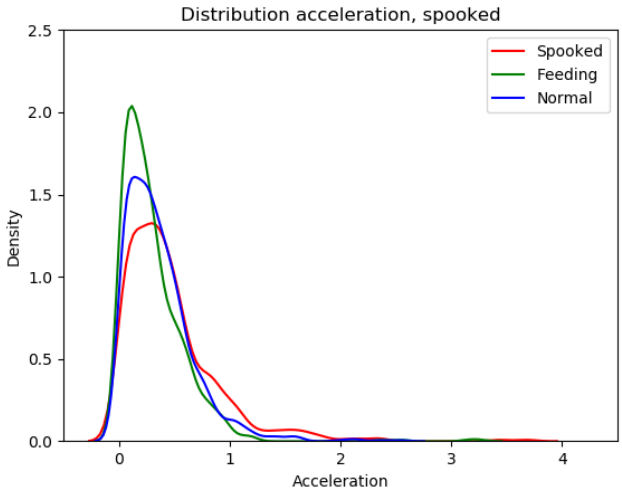
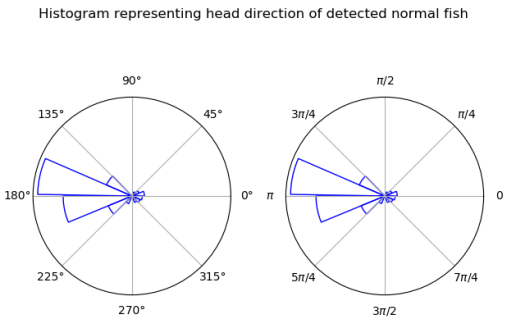
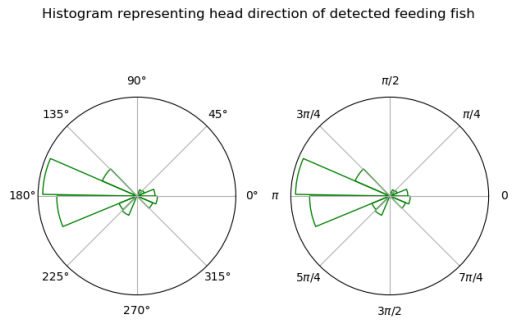


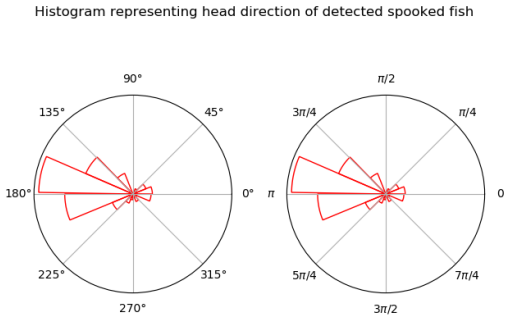
Figure 4.12: Distributions of accelerations. The difference between the spooked class and the two other classes is less distinct here, compared to the velocity distributions.



(a) Angle, normal.



(b) Angle, feeding.



(c) Angle, spooked.

Figure 4.13: A circular histogram visualizing the angles of each detected fish. The area of each bin represents how many data points are in each bin. A doubling of data points in a bin results in a doubling of the area(not the radius). As expected, we see that almost all fish are pointed towards the current.

4.4 Preliminary research in finding tail beat frequency

In addition to the statistics we have already generated, we started researching possible ways of finding the tail beat frequency of each fish. While the statistics are based on several thousands of frames, we have only carried out a small test consisting of 53 frames for this approach. In this sequence we have used the MASK R-CNN to generate masks of a fish that does approximately 4 tail beats in the duration of the sequence. To describe the tail beat frequency, we first want to isolate the tail of the fish to reduce disturbances. For this, we have used the technique of placing an ellipse over the fish. We then placed a circle with center at the tail end of the major axis of the ellipse. All the pixels within this circle is extracted and the results is seen in figure 4.14. Then, we extract the pixels which

is overlapping between the circle and the mask. This method will isolate the tail of the fish and the result is seen in figure 4.15.

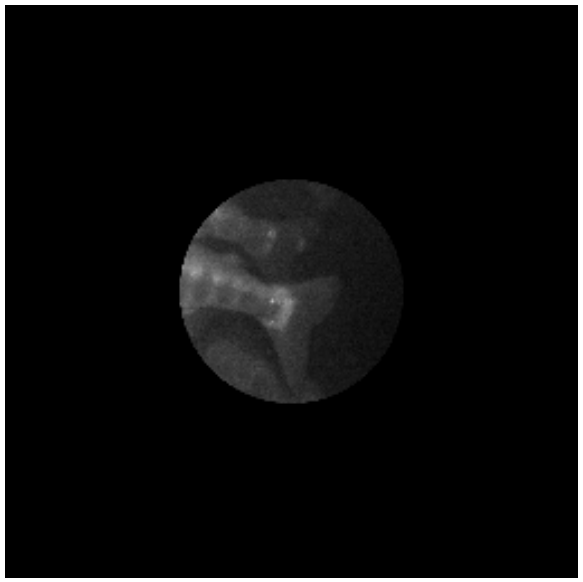


Figure 4.14: Illustration of the extracted circle around a fish tail.

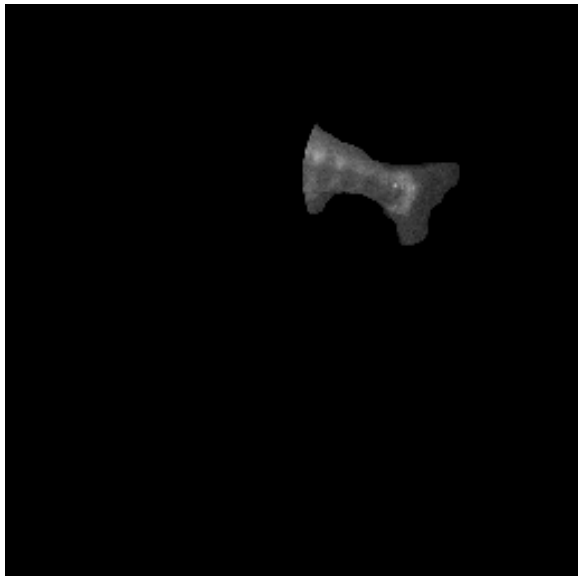


Figure 4.15: Illustration of a isolated and masked tail.

To look for a periodic signal, we first tried simple methods as plotting a time-series of the area and mean pixel intensity of the isolated tail. These time-series can be seen in figure 4.16 and figure 4.17. Both series have their similarities, but they didn't show enough resemblance of a periodic signal for further exploration.

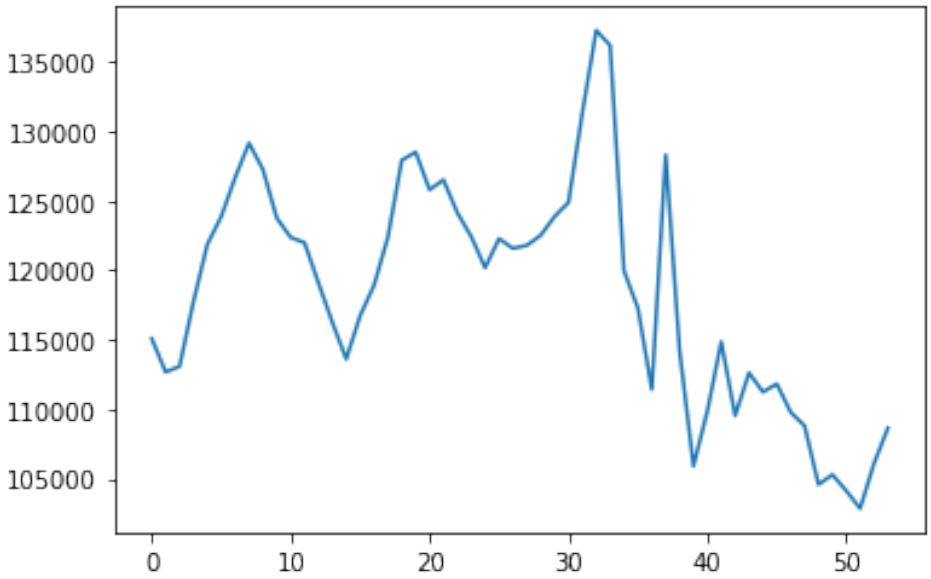


Figure 4.16: Time series of the area of the isolated tail.

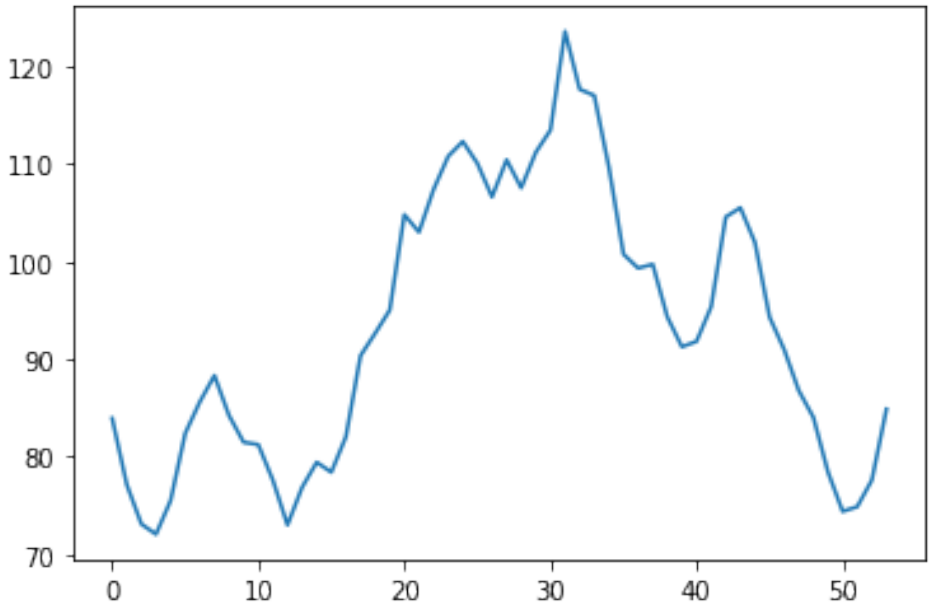


Figure 4.17: Times series of mean pixel intensity of a masked tail.

The second method we tried involves using Optical Flow. Optical Flow is the pattern

of the apparent movement of a camera or an object between two consecutive frames. It is also referred to as apparent motion of image brightness patterns in an image sequence. If we consider a pixel $P(x_0, y_0, t_0)$ in the first frame that is moved by a distance (dx, dy) in the second frame taken after dt, we end up with the equation:

$$P(x_1, y_1, t_1) = P(x_0 + dx, y_0 + dy, t_0 + dt) \tag{4.1}$$

Then, by taking the Taylor series approximation of the right-hand side, removing the common terms and dividing by dt, we get:

$$f_x u + f_y v + f_t = 0 \tag{4.2}$$

where:

$$f_x = \frac{\delta f}{\delta x}; \quad f_y = \frac{\delta f}{\delta y} \tag{4.3}$$

$$u = \frac{dx}{dt}; \quad v = \frac{dy}{dt} \tag{4.4}$$

f_x and f_y are the image gradients, and f_t is the gradient along the time dimension. To find u and v, there exists several methods. Here, we used Dense Optical Flow, which is based on Gunner Farneback's algorithm explained in the paper *Two-Frame Motion Estimation Based on Polynomial Expansion*[7].

We center the tail in each image to remove the motion of the fish to focus on how the brightness moves as the tail beats. The output of the Dense Optical Flow algorithm is a 2-channel array with Optical Flow vectors (u, v). We convert to HSV color space, where H (Hue) represents the direction, V (Value) represents the magnitude and S (Saturation) is set to 255. This is then converted to the RGB color space to produce an Optical Flow image as seen in figure 4.18. To look for a periodic pattern, some experimenting was done with the generated flow vectors. By our findings, converting to gray-scale and computing the mean pixel intensity over each frame results in a promising periodic time-series. We then filter the signal using a low-pass filter to generate a filtered time-series as seen in figure 4.19. We compute the fast Fourier transform(FFT) of the time-series and extract the largest peak in the FFT to receive the strongest periodicity. By multiplying this frequency(0.0755) by the total number of frames(53) we believe to have found the number of tail beats(4.001) in the video sequence. An animation of the time-series and tail beats is attached to this thesis. In the animation we can see a clear connection between each cycle and the tail beats. It is important to note that this has only been tested on one video sequence. Further testing has to be conducted to verify if this is a an approach that can be used or not.

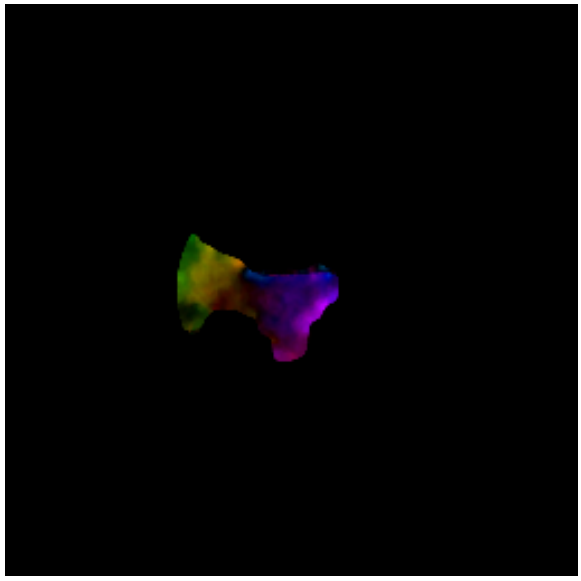


Figure 4.18: Optical Flow image of a masked tail.

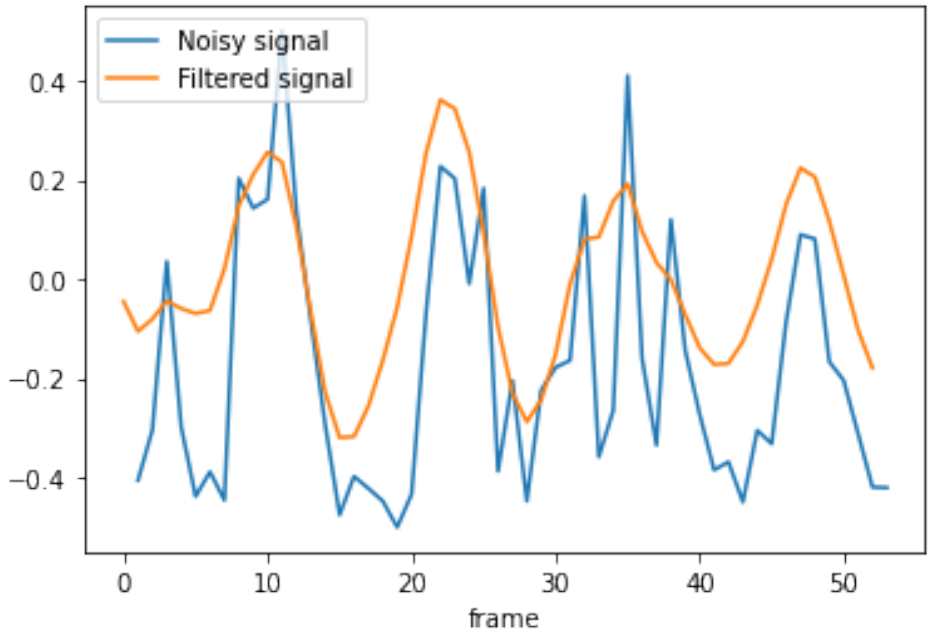


Figure 4.19: Time-series of mean pixel intensity when using Optical Flow and converting to gray-scale.

4.5 Overall performance discussion and future work

As we have seen, by looking at the statistics from our setup, we are able to clearly distinguish between the spooked class and the two others. Finding a difference between the feeding class and normal class is a more challenging task, which confirms the hypothesis we made when we captured the videos. Even when having an overview of the whole tank when filming the videos, a change in behaviour when the feed was dropped was hard to spot. We believe the reason behind this is because the salmon are fed so consistently. When we were filming, they were fed approximately every 2 minutes. If the feeding had happened once or twice a day, it might have been easier to spot a difference. For now, the normal situation represents the 2 minutes between the feeding. The fish might behave differently if the fish goes longer periods without feed. As a result, the definition normal- and feeding behaviour could be changed. An alternative is to merge the two classes or define the normal class as a period where the fish is not fed for a certain amount of time (for example at night).

4.5.1 Sources of error

There are certain elements in our approach that can cause wrong calculations in the generated statistics. Mostly, it comes down to the Mask R-CNN. Most of the calculations we use in our approach rely on the masks created by the Mask R-CNN. Usually, the problem which occurs is masks that are not complete(e.g missing part of the tail) or masks that contain elements which are not part of the fish as we saw in figure 4.2. This can cause the resulting bounding box to be wrongly placed on the fish. The exported area is also affected by this. As a consequence, all the underlying calculations can or will be affected.
We also have a source of error when it comes to the 2D calculations. For fish that swims perpendicular to the camera the approach works, but when fish moves towards or away from the camera, we loose/ignore a component and are not able to accurately describe the actual movement. It also uses the assumption that the salmon are equal in size, which might not always be the case.
In some situations is is also possible that the the tracker makes a mistake. If two fish are swimming close or on top of each other, there is a possibility for the tracker to mistake one fish for the other one. This doesn't necessarily affect the resulting shoal statistics, but rather individual statistics.

4.5.2 Future work

The largest bottleneck of our network is the Mask R-CNN. It is not able run in real-time, and could still see improvements. Since the tracker relies on the segmentation from this network, it should be the priority when improving the complete setup. To increase the performance the network we make the following suggestions:

1. Increase the dataset by a large margin. Mask R-CNN is a powerful network capable of handling large datasets. Usually, the rule the more the better applies here.

2. Improve the quality of the dataset. The performance of deep learning networks are dependent on the quality of the data they are training on. With a better camera setup, especially regarding the illumination, we could achieve better focus in the images and have sharper edges between the fish and the background, which makes it easier to segment the fish. Another problem for our setup was that the fish would not stay in the captured frame for a long time. Looking through the distribution of how long the fish is detected for in figure 4.20, we can see that most fish are not tracked for more than $\sim$20 frames, which is around a second(we use 18 FPS). Therefore, a wider camera angle could be tried to enable tracking over longer distances.

3. Use as stereo setup when capturing videos. Capturing 3D images could provide us with more information regarding the movements of the fish in the image. In this thesis we only consider the 2D plane, but when using 3D we could both extend and improve our calculations.

4. Exploring the possibilities of a dual stream architecture. This means that we could train two parallel networks, for example one that is trained on thresholded images and one that is trained on normal NIR-images. The final output will be a weighted sum of the output of each stream.

5. Use another network architecture. In the recent year, several new architectures which achieves state of the art performance in instance segmentation have surfaced. Experimenting with a new architecture could prove beneficial. State of the art methods can be found here [25].

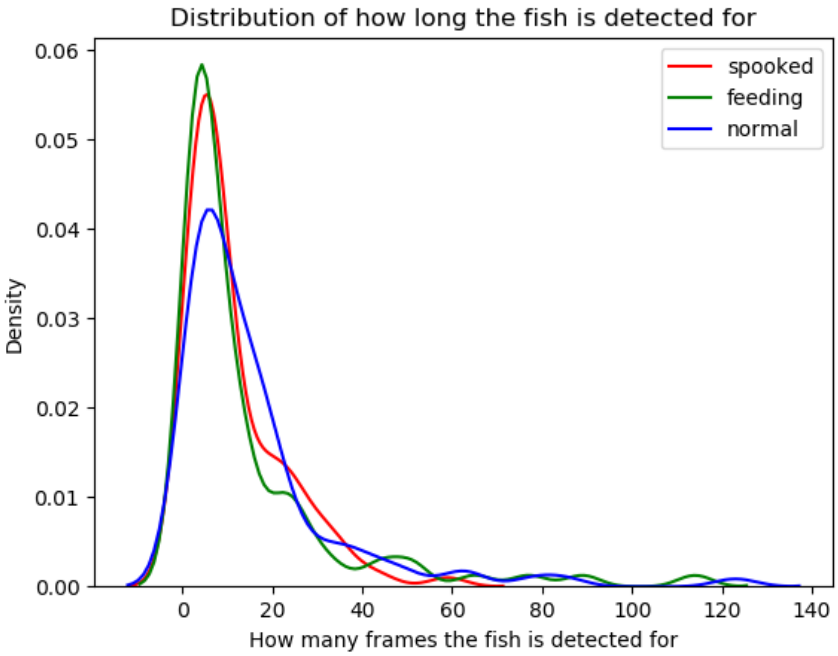


Figure 4.20: Distribution of how long the fish is detected for.

For a large scale test of the setup, we could use multiple cameras to cover a larger part of the tank. Then, behaviour classes could be tested over a longer period of time. An example is where the fish is sick. If one tank contained sick fish, the data generated from this tank could be compared to the data generated from a normal tank.

As for now, we have only used visualization of one variable at a time to look for deviance in the statistics. However, it is not always one variable alone that allows us to find an anomaly, but rather a combination of variables. The goal of this thesis is not to produce statistics that will be manually inspected through visualisation, but rather create data which can be used in automatic approaches to detect anomalies in such data.

There are many approaches of finding anomalies in data series. Examples include simple statistical methods as traversing the mean over a time-series, but also machine learning methods as clustering-based anomaly detection and support vector machine-based anomaly detection. Another example is to use auto-encoders. These neural networks try to reproduce the input data. They have as many input nodes as output nodes, while the hidden layers have fewer nodes. This way, the hidden layers will extract useful information from the input to effectively reproduce it at the output layers. A well trained auto-encoder

reproduces the input with minimal error and learns to reconstruct data that follows a certain format. It is trained on normal data and learns how the features will look for this data. When an anomaly is fed through the model, it fails to reproduce it and ends up with a large error term. A successful implementation could, in the future, result in a fully automated approach of detecting anomalies in the fish tank.
An architecture where not the statistics, but the original videos are fed to network could also be tried. This field is known as Action Recognition. However, this method will only classify each behaviour and not provide any statistics from the tank.

Chapter 5

Conclusion

In this thesis, we present an approach for classifying different behavior characteristics through deep learning and multiple object tracking. Through manual inspection of the generated tank statistics, we found the method able to distinguish one class from two other classes. A summary of the goals we presented in the beginning of the thesis is a follows:

1. **Create a network that can successfully segment salmon in NIR images**:
   The Mask R-CNN we used, segments salmon to a satisfactory level, when filtering the the segmentation based on confidence score. Future improvements can greatly increase the performance of our method.

2. **Create a tracker that can successfully track salmon in a video stream**:
   The modified SORT algorithm we use, is successful in tracking the segmented salmon from the Mask R-CNN.

3. **Create data based on the tracking which includes velocity, acceleration, and swimming/moving direction**:
   We are able to generate statistics on velocity, acceleration and moving direction using the 2D calculations we presented in chapter 3.4.

4. **Test the program on new data to check if we are able to distinguish between the data generated from three different behaviour classes: feeding, normal and spooked.**:
   We have tested our approach on three classes, where we can distinguish the spooked class from the two other classes.

The proposed approach is only tested on a small dataset and further improvements must be made before carrying out a large scaled test. Successfully improving the method, could result in practical industrial relevance given we are able to provide valuable information about the state in the tank.

Bibliography

[1] Abdulla, W., 2017. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. https://github.com/matterport/Mask_RCNN.

[2] abewley, 2020. Sort. https://github.com/abewley/sort.

[3] Bewley, A., Ge, Z., Ott, L., Ramos, F., Upcroft, B., 2016. Simple online and realtime tracking, in: 2016 IEEE International Conference on Image Processing (ICIP), pp. 3464–3468. doi:10.1109/ICIP.2016.7533003.

[4] Brochu, F., 2019. Increasing shape bias in imagenet-trained networks using transfer learning and domain-adversarial methods .

[5] Brooks, J., 2019. COCO Annotator. https://github.com/jsbroks/coco-annotator/.

[6] Brown, R.G., Hwang, P.Y.C., 1997. Introduction to random signals and applied kalman filtering: with MATLAB exercises and solutions; 3rd ed. Wiley, New York, NY. URL: https://cds.cern.ch/record/680442.

[7] Farnebäck, G., 2003. Two-frame motion estimation based on polynomial expansion, in: Proceedings of the 13th Scandinavian Conference on Image Analysis, Gothenburg, Sweden. pp. 363–370.

[8] Føre, M., Frank, K., Norton, T., Svendsen, E., Alfredsen, J.A., Dempster, T., Eguiraun, H., Watson, W., Stahl, A., Sunde, L.M., Schellewald, C., Skøien, K.R., Alver, M.O., Berckmans, D., 2018. Precision fish farming: A new framework to improve production in aquaculture. Biosystems Engineering 173, 176 – 193. URL: http://www.sciencedirect.com/science/article/pii/S1537511017304488, doi:https://doi.org/10.1016/j.biosystemseng.2017.10.014. advances in the Engineering of Sensor-based Monitoring and Management Systems for Precision Livestock Farming.

[9] Girshick, R., Donahue, J., Darrell, T., Malik, J., 2013a. Rich feature hierarchies for accurate object detection and semantic segmentation. arXiv:1311.2524.

[10] Girshick, R.B., 2015. Fast R-CNN. CoRR abs/1504.08083. URL: http://arxiv.org/abs/1504.08083, arXiv:1504.08083.

[11] Girshick, R.B., Donahue, J., Darrell, T., Malik, J., 2013b. Rich feature hierarchies for accurate object detection and semantic segmentation. CoRR abs/1311.2524. URL: http://arxiv.org/abs/1311.2524, arXiv:1311.2524.

[12] He, K., Gkioxari, G., Dollár, P., Girshick, R.B., 2017. Mask R-CNN. CoRR abs/1703.06870. URL: http://arxiv.org/abs/1703.06870, arXiv:1703.06870.

[13] He, K., Zhang, X., Ren, S., Sun, J., 2015. Deep residual learning for image recognition. arXiv:1512.03385.

[14] Heras, J., 2020. Clodsa. https://github.com/joheras/CLoDSA.

[15] Hosteland, L.T.S., 2017. Nytt konsept skal øke menneskets evne til å overvåke oppdrettsfisk. https://www.kyst.no/article/nytt-konsept-skal-oeke-menneskets-evne-til-aa-overvaake-oppdrettsfi

[16] Kalman, R.E., 1960. A new approach to linear filtering and prediction problems. Transactions of the ASME–Journal of Basic Engineering 82, 35–45.

[17] Kelly, A., 2018. Mask r-cnn training and inference. https://github.com/akTwelve/tutorials/blob/master/mask_rcnn/MaskRCNN_TrainAndInference.ipynb.

[18] Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks, in: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (Eds.), Advances in Neural Information Processing Systems 25. Curran Associates, Inc., pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[19] Leal-Taixé, L., Milan, A., Reid, I., Roth, S., Schindler, K., 2015. Motchallenge 2015: Towards a benchmark for multi-target tracking. arXiv:1504.01942.

[20] Lecun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 2278–2324.

[21] Lin, T., Maire, M., Belongie, S.J., Bourdev, L.D., Girshick, R.B., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L., 2014. Microsoft COCO: common objects in context. CoRR abs/1405.0312. URL: http://arxiv.org/abs/1405.0312, arXiv:1405.0312.

[22] Moutarde, F., . Tutorial deep learning. http://perso.mines-paristech.fr/fabien.moutarde/ES_MachineLearning/TP_convNets/convnet-notebook.html.

[23] Mulchrone, K.F., Choudhury, K.R., 2004. Fitting an ellipse to an arbitrary shape: implications for strain analysis. Journal of Structural Geology 26, 143 – 153. URL: http://www.sciencedirect.com/science/article/pii/S0191814103000932, doi:https://doi.org/10.1016/S0191-8141(03)00093-2.

[24] Otsu, N., 1979. A threshold selection method from gray-level histograms. IEEE Transactions on Systems, Man, and Cybernetics 9, 62–66.

[25] Papers-With-Code, 2020. Instance segmentation on coco test-dev. URL: https://paperswithcode.com/sota/instance-segmentation-on-coco.

[26] Ren, S., He, K., Girshick, R., Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. arXiv:1506.01497.

[27] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L., 2015. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) 115, 211–252. doi:10.1007/s11263-015-0816-y.

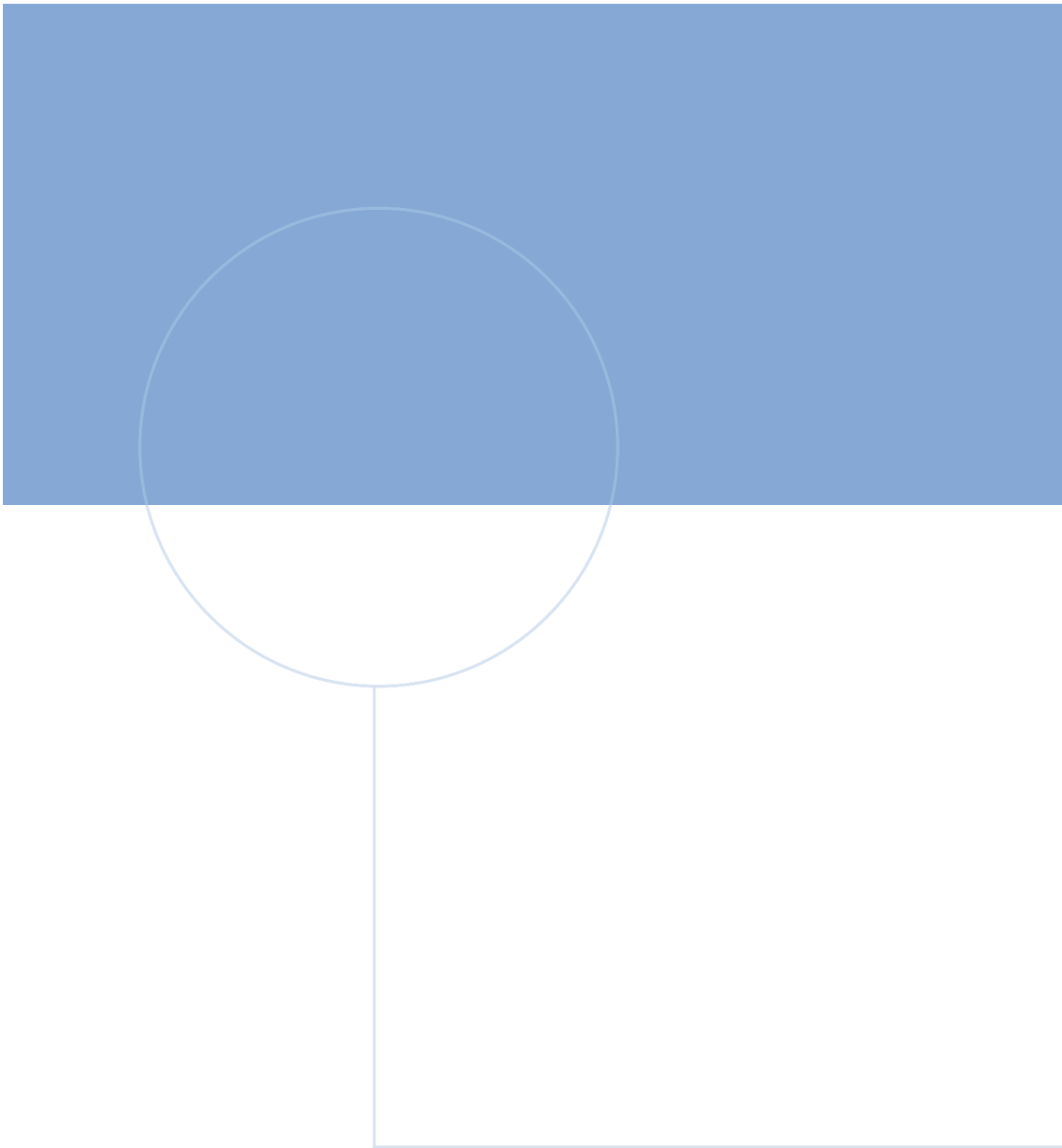
[28] Sagstad, S., 2019. Collecting and preparing an image dataset of salmon smolt for use in deep learning based behavioral analysis and classification. Project report in TTK4550. Department of Information Security and Communication Technology, NTNU – Norwegian University of Science and Technology.

[29] Simard, P.Y., Steinkraus, D., Platt, J.C., 2003. Best practices for convolutional neural networks applied to visual document analysis, in: Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings., pp. 958–963.

[30] Simonyan, K., Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv 1409.1556 .

[31] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2014. Going deeper with convolutions. arXiv:1409.4842.

[32] Uijlings, J., van de Sande, K., Gevers, T., Smeulders, A., 2013. Selective search for object recognition. International Journal of Computer Vision URL: http://www.huppelen.nl/publications/selectiveSearchDraft.pdf, doi:10.1007/s11263-013-0620-5.



NTNU

Kunnskap for en bedre verden