Master's thesis

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

Petter Tafjord Drønnen

Detection of a person in the water from thermal images

Master's thesis in Cybernetics and Robotics
Supervisor: Annette Stahl
June 2022

NTNU
Kunnskap for en bedre verden

Petter Tafjord Drønnen

Detection of a person in the water from thermal images

Master's thesis in Cybernetics and Robotics
Supervisor: Annette Stahl
June 2022

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

NTNU
Norwegian University of
Science and Technology

NTNU

Kunnskap for en bedre verden

Detection of a person in the water from thermal images

Petter Tafjord Drønnen

Master's thesis

TTK4900

Faculty of Information Technology and Electronics

Norwegian University of Science and Technology

Norway

June 6, 2022

Supervisor: Annette Stahl

Pages / Appendix

64 / 69

Preface

This report is a master's thesis and concludes my master's degree in Cybernetics and Robotics at the Norwegian University of Technology and Science in Trondheim. The work is done in collaboration with Zebop Avalon [3] and was started in January 2022 and finished in early June 2022. The report continues the preproject work done in the subject TTK4551 completed in autumn 2021.

Now that I am finishing my master's degree, I would like to express my gratitude to all my friends and family for always having my back, and this would not be possible without you all. A special thanks to you mom, dad, Robert, and Karina.

Trondheim, June 6, 2022

Petter Tafjord Drønnen

Acknowledgement

I want to thank all the contributors to this project, and especially I would like to thank:

- My supervisor Annette Stahl at NTNU, for guidance throughout the project
- Zebop Avalon for giving me the opportunity
- Special thanks to Oscar Markovic, Max Montzka, and Kjetil Beck at Zebop Avalon for help and hardware
- Family and friends who have supported me throughout this period

A special thanks to the people in this picture that works at and with Zebop Avalon

Summary

Each year in Norway, about 100 people die from drowning accidents. Falling from land or a dock is overrepresented in the statistics, and accidents often occur when the victims are alone. This thesis aims to develop a system that uses a thermal camera, object detection, and object tracking to detect a person falling into the water to decrease the number of drowning accidents. The system uses several methods for detecting a person falling, such as detecting an object as a falling person with object detection, a line crossing check, and the relative movement speed of objects. The dataset created includes seven labels, person, car, dog, boat, truck, falling person, bus, and contains 3249 labeled thermal images. A YOLOv4-tiny model was the object detection model and scored a mean average precision of 87.23% on the validation set. Deep SORT was used as the object tracker algorithm and saved the object's history as data points for trajectory analysis. However, the Deep SORT did not reach a real-time performance, running at only nine frames per second. This leaves much potential for improvement. The trajectory analysis shows that it is easy to distinguish between relevant movement patterns, such as a person walking, falling, or climbing. The overall result is that such a system presented in this thesis could save lives in the future, but there are still many improvements to make the system even more robust.

Sammendrag

Hvert år dør rundt 100 mennesker av drukningsulykker i Norge. Fall fra land eller brygge er overrepresentert i denne statistikken, og ulykker skjer som oftest når ofrene er alene. Denne oppgaven har som mål å utvikle et system som bruker et termisk kamera, objektdeteksjon og objektsporer for å oppdage en person som faller i vannet for å redusere antall drukningsulykker. Systemet bruker flere metoder for å oppdage en person som faller. Systemet kan oppdage et objekt som faller som en fallende person, utføre en linjekryssingssjekk og se på relativ bevegelseshastighet for objekter. Datasettet som ble laget inneholder syv kategorier, person, bil, hund, båt, lastebil, fallende person, buss, og inkluderer 3249 merkede termiske bilder. En YOLOv4-tiny modell ble brukt som objektdeteksjonmodell og fikk en gjennomsnittlig presisjon på 87.23% på valideringssettet. Deep SORT ble brukt som objektsporingsalgoritmen og lagret objektets historie som datapunkter for baneanalyse. Deep SORT oppnådde imidlertid ikke en sanntidsytelse da den bare nådde ni bilder per sekund og har derfor et stort forbedringspotensial. Baneanalysen viser at det er lett å skille mellom relevante bevegelsesmønstre, som at en person går, faller eller klatrer. Det samlede resultatet er at et slikt system som er presentert i denne oppgaven kan redde liv i fremtiden, men det er fortsatt mange forbedringer for å gjøre systemet enda mer robust.

Contents

Abbreviations

FOV Field of view

SORT Simple Online and Realtime Tracking

YOLO Your Only Look Once, state-of-the-art real-time object detection

FPS Frames per second

UAV Unmanned aerial vehicle

CV Computer vision

IR Infrared

RGB Red, Green, Blue, the standard way to display colored images

ROI Region of interest

ML Machine learning

AI Artificial intelligence

DL Deep learning

CNN Convolutional Neural Network

IoU Intersection over union

mAP mean Average Precision

GPU Graphics processing unit

CPU Central Processing Unit

SDK Software Development Kit

RTSP Real-Time Streaming Protocol used to access the camera stream from Solsiden

CSV Comma Separated Values, a file format used to store data points in a file

List of Figures

List of Tables

Chapter 1

Introduction

In recent decades the price and size of electronics and cameras have been significantly reduced. When taking a closer look at the thermal camera's history, it is possible to see that in the first 30 years, a camera cost $ 50,000 [38]. Today it is possible to buy a small adapter that can be put on a mobile phone for $ 230 [19]. When more people can access better and cheaper hardware, it is easier to innovate in new and old fields. An example of this is the new field of making use of thermal cameras in self-driving cars because a thermal camera can see as good in the dark as during the day [20]. Another place it is possible to innovate is to use thermal cameras and machine learning to make automated security systems. The system can run continuously without pauses and automatically alert the proper authorities.

The Norwegian Society for Sea Rescue works to prevent drowning accidents in Norway and collects many statistics about the accidents. When looking at these statistics [64] it is possible to see that on average the last ten years, 93 people have drowned every year in Norway, with 2021 having 75 people drowning.

When categorizing the most common accidents, a fall from land or a dock is overrepresented, as seen in figure 1.1. The statistics also show that three out of four people that were over 60 years were alone when the accident occurred [75]. Due to so many being alone when the accident happens, it would be suitable to develop a system that can make it so that no one is alone. The system will give the people a chance to get the help they otherwise would not get.

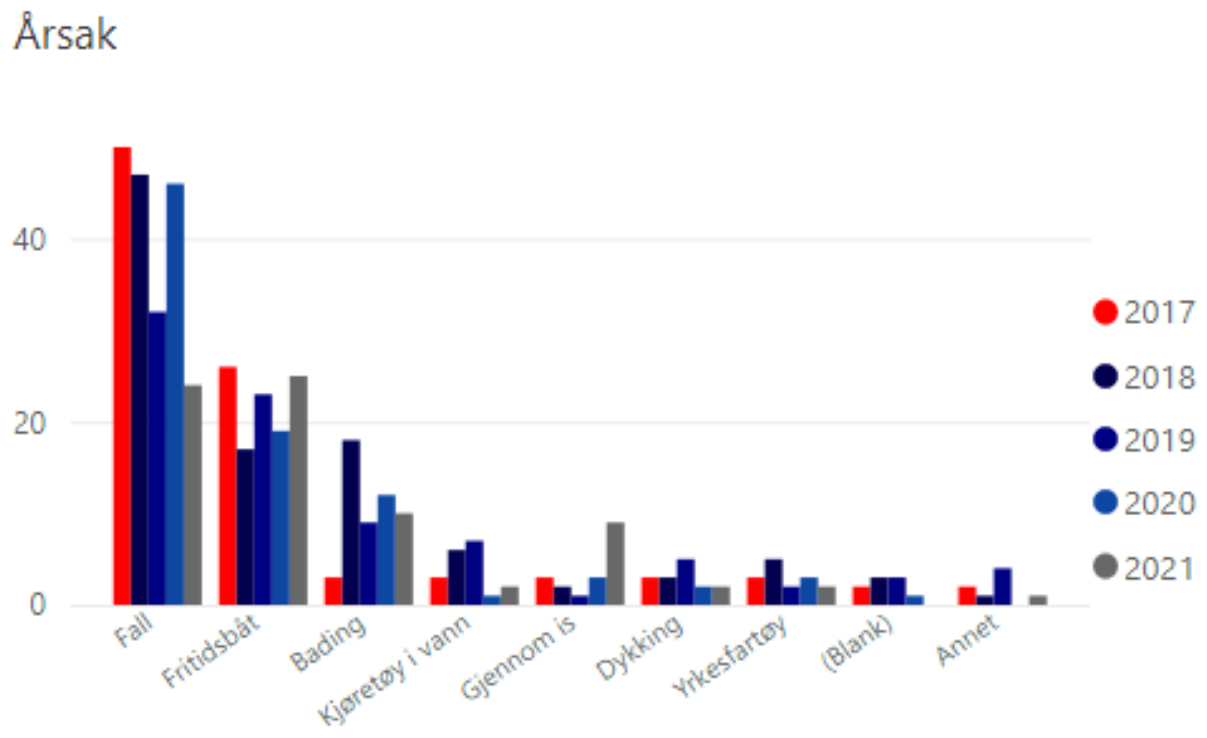


Figure 1.1: Where drowning accidents occur [64]

Nidelva in Trondheim is infamous for its almost yearly drowning accidents, as mentioned in this news article [49], and several warnings, such as in figure 1.2 are in the close vicinity to the river. Warning translated gives “12 men have drowned in the Nidelva since 2000. Half of them did not reach the age of 22.”

Figure 1.2: Warning placed at Nidelva

In November 2018, an 18-year-old boy named Odin disappeared in Trondheim. His body is still not yet located, and the hypothesis is that he fell into Nidelva. The system discussed in this thesis may have seen Odin falling into the river and could have saved his life. This thesis is done in collaboration with Zebop Avalon [3], and the Odin accident is the motivation for the company. Zebop Avalon’s motto is that nobody should drown and no accident should go unsolved.

1.1 Problem description

This study aims to develop a system that uses object detection and object tracking algorithms to implement several methods for detecting a person falling into the water. When this system works correctly, it can be used to notify rescue services about an incident and then hopefully reduce the number of drowning accidents.

The sensor used as input for the object detection and object tracking algorithms was a thermal camera. The reason for using a thermal camera is its superior performance in poor lighting conditions compared to a regular camera [79]. Lighting is essential because the system should work at any time throughout the year.

1.2 Motivation

One of the biggest benefits of using a thermal camera compared to a normal camera is its superior capture quality when it is dark or in bad weather conditions. Figure 1.3 and figure 1.4 is a comparison between the two. The pictures were taken in the evening in the wintertime of Norway. From the figures, it is possible to see that the thermal image shows features that are hard to see or completely black in the RGB image. For instance, the drainage pipe near the camera is visible on thermal but completely black on the RGB image.

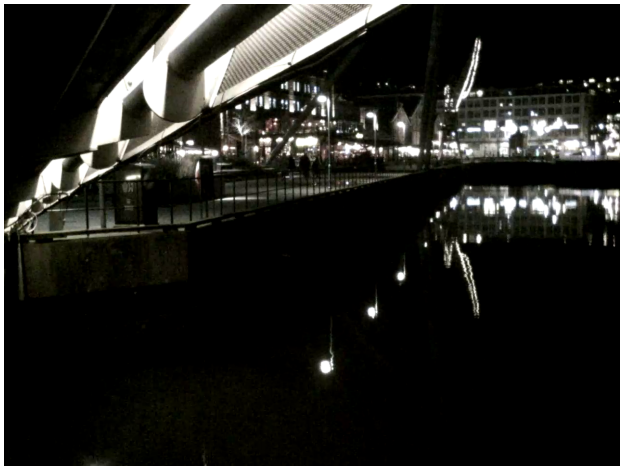


Figure 1.3: Solsiden at night seen in RGB images

Figure 1.4: Solsiden at night seen in thermal images

Since the system must function at all times with as little as possible human interaction, machine learning (ML) was needed. ML has a proven record for systems that can work independently from humans. From robots to automated cars to security systems to military use. The selected ML method was object detection and object tracking. These can detect and track objects and can be used in several different ways to give a robust system. Object tracking also allows looking at movement patterns and can be used to detect abnormal movement. Furthermore, no one uses object detection, object tracking, and thermal cameras to look for drowning accidents. Therefore, by realizing this product, society could benefit greatly.

1.3 Background

On Blomsterbrua in Trondheim, Zebop Avalon has placed their test system, including the thermal camera, as seen in figure 1.5. The figure also shows the camera's field of view (FOV) as the camera looks at Solsiden. Solsiden will be the test arena for the system. However, new test places could be necessary to test further and improve the system when the system behaves satisfactorily on Solsiden.

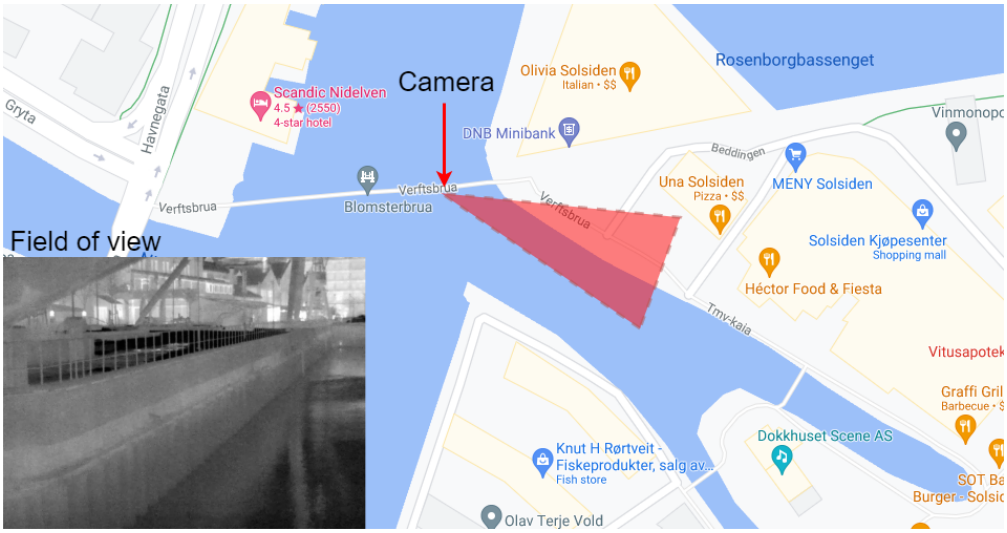


Figure 1.5: Camera placement in Trondheim

1.4 Aim and objectives

This thesis aims to make a system that can help prevent drowning accidents. Several objectives should be met to achieve this. The objectives are as follows:

- Create a new original dataset for thermal images
- Develop a system that uses object detection and object tracking to detect a person falling into the water
- Analyze the data captured by the object tracker

1.5 Contributions

The main contributions of this paper are:

- A system that can detect a person falling into the water using object detection, line detection, and movement detection
- Save and analyze Deep SORT data to recognize the movement of a person walking and falling and plot the data
- A new original dataset containing 3249 thermal images with the labels person, car, dog, boat, truck, falling person, and bus. The dataset is recorded in four different places, both indoors and outdoors, and in different weather conditions to make the data as varied as possible

1.6 Structure of the Report

The rest of the report is structured as follows:

Chapter 2 - Previous work: Relevant work done by others are presented

Chapter 3 - Theory: The theoretical background needed to understand the thesis

Chapter 4 - Method: Methods used are presented

Chapter 5 - Result: The different results are presented

Chapter 6 - Discussion: A discussion on the different results are presented

Chapter 7 - Conclusions: The report is concluded

Chapter 2

Previous work

The work done in this thesis is about both object detection, object tracker, and storing and using the data from the object tracker. This chapter presents existing work done in the respective fields and how their result may be helpful in this thesis.

2.1 Object detection and object tracking

A good object tracker is only as good as the object detection [4]. Therefore, it is essential to have a good object detector that maximizes the performance of the tracker. The following sections will present object detection and object tracking work done by others, emphasizing the performance of several different detection models when using them with an object tracker. Much work is done with object detection and tracking with an ordinary camera but not as much with thermal images. In order to reduce repetition, only the best and most relevant work is presented.

In Wojke [81] the authors took the Simple Online and Realtime Tracking (SORT) [5] algorithm that was one of the best trackers at the time and improved it. The SORT algorithm is a simple algorithm that uses a Kalman filter to predict bounding boxes and the Hungarian method to measure the bounding box overlapping, enabling tracking of objects. This method is fast but comes with many identity switches when occlusions occur, and it is here that the Deep SORT algorithm comes in. The Deep SORT takes the SORT algorithm but extends it with a pre-trained neural network that generates features from the objects. This neural network helps the algo-

rithm recognize objects when they are occluded for a time. When using Deep SORT, the identity switches were reduced by 45% compared to only using the SORT algorithm and can still be used in a real-time system. The Deep SORT algorithm developed in Wojke [81] looks like an excellent choice to use as an object tracker. It performs well both in speed and accuracy and is easy to implement. Moreover, it is popular, as mentioned in [71, 46], and therefore seeking help when problems occur is easier.

In Punn [61] the authors compared the neural networks, Faster-RCNN, single-shot detector, and You Only Look Once (YOLO) version 3 to use with Deep SORT. The task was to monitor social distancing due to COVID-19 from surveillance videos. Punn and the authors found out that YOLOv3 outperformed the other two networks, and it was the best candidate to be used with Deep SORT. The result shows that the YOLO architecture can be an outstanding choice for the network to cooperate with Deep SORT in this paper.

In Parico [57] the author's task was to make a real-time pear fruit detection and counting system. The authors used Deep SORT as the object tracker, and for the object detection, they made a thorough comparison between YOLOv4, YOLOv4-tiny, and YOLOv4-CSP. The authors went as far as testing several different network sizes of each of the three models. The overall best model was the YOLOv4, and they used the YOLOv4-512 model in their system. The -512 name means that the network resolution is 512x512. The tests also showed that the YOLOv4-tiny network outperformed the YOLOv4 model at lower network resolutions. Since the video resolution in this paper is relatively low and high speed is essential, a good model choice can be the YOLOv4-tiny model at a resolution of 416x416.

In Doan [13] the authors developed a system for detecting and counting different vehicles using YOLOv4 and Deep SORT. The author's emphasis was on performance, and they achieved 52.3 average precision at 55 frames per second (FPS) when using an image resolution of 416x416. This result makes it possible to use the YOLOv4 and Deep SORT to make a real-time system. The result achieved in Doan [13] is much better than the preproject (see appendix C) with an FPS of 31 at the exact image resolution and without Deep SORT. The different results between

Doan [13] and the preproject indicates that there is either an error in the preproject or that Doan [13] results are tough to reach.

In Chaverot [9] the authors tested the performance of object detection models only trained on RGB images against models trained on thermal images with different dataset sizes. The dataset used was the FLIR ADAS [21] dataset that contains 10228 images. They split the dataset into 7860 images for training and 1360 for validation. During testing, a standard YOLOv4 model scored a mean average precision of 69.13%, and their model trained on the full dataset scored 84.88%. Figure 2.1 shows the mean average precision of the other models. From this, it is possible to see that the performance is minimally reduced with a dataset 1/4 of the original size. Using this result indicates the minimum size of the dataset this thesis should have at $\frac{10228}{4} = 2557$ images.

	Persons	Bicycles	Cars	mAP
FLIR (Full Base)	88.12	74.96	91.57	84.88
FLIR 1/2	87.65	70.91	91.01	83.19
FLIR 1/4	86.72	71.87	90.59	84.84
FLIR 1/8	84.4	65.84	89.24	79.83
FLIR 1/16	80.52	59.76	88.04	76.10

Figure 2.1: Mean average precision of different YOLOv4 models trained on different dataset sizes. The table is taken from Chaverot [9]

2.2 Data visualization and analysis

An ideal way to display information is needed when collecting much data. It was not easy to find many theses containing relevant and good data visualization and analysis explanations. However, the following sections will explain the theses that were found and were relevant and good enough. The sections will look at how others have visualized their data and how they have used the data.

In Shaoji [70] the authors used an unmanned aerial vehicle (UAV) to capture video of three different squares. Then the authors used YOLOv3 and Deep SORT to save every midpoint of the bounding boxes to a .csv file. After this, they cleaned the data by removing outliers and then plotted it in several different ways. They used the python libraries matplotlib [43] and seaborn [69] to plot a map of all the trajectories saved. The plots they used were a heat map of the people distribution and a heat map of people's walking speed. All the maps plotted look very well and clean. However, the authors in Shaoji [70] only explain which library they used and not how they used them. This makes recreating what they have done challenging.

In Gatelli [25] they used YOLOv4 and Deep SORT to count cars and look at the movement of cars at road intersections. The authors recorded several intersections and then ran YOLOv4 and Deep SORT on the data and recorded every midpoint of the bounding boxes, similar to what Shaoji [70] did. Gatelli [25] then plotted the trajectories in a graph to look at where and how most vehicles moved. They could then automate the decisions on improving the intersections, such as installing traffic signs or widening areas. The findings in Gatelli [25] can be used to look at the standard moving patterns on Solsiden and to make plots to show this. One problem is that Gatelli [25] did not explain how they saved the data or made the plots, and therefore recreating this can be difficult. However, if done correctly, this can be used to raise the alarm if someone is deviating from the usual walking path.

In Hng [31] they developed a UAV that could fly to the front of congesting vehicles due to an accident. The authors in Hng [31] used YOLOv4 and Deep SORT to look at the relative speed of vehicles to determine if there was congestion or not. To calculate the relative speed, they used:

$$\text{Relative speed} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

This takes the coordinates from a midpoint of a bounding box and the same coordinates of a new bounding box at a later frame and calculates how many pixels it has moved. If the pixel value is lower than a set threshold, the cars are driving slow, and congestion happens. When looking for people falling into the water, the relative speed can be used to look at the speed of objects. A person falling will move much faster than a person walking on a typical path.

Chapter 3

Theory

The following chapter explains the usage of object detection and object tracking. The sections explain state-of-the-art methods in their respective fields and present the basic theory necessary to understand the methods and results later in this report.

3.1 Object tracking

The task of the object tracking algorithm is to give each object from the object detection algorithm a unique ID. The object detection sends the location of each detected object to the object tracker every video frame. The tracker should then provide the same ID to the same object and new IDs to new objects. ID matching should work as objects move around and also if they disappear and reappears in a relatively short time [11].

The advantage of using a tracker is that one can look at the movement of an object through time. Therefore, it is possible to do more data analysis, such as the relative speed of an object. The disadvantage of a tracker is that it is often very computational heavy, which can increase the model's time to make predictions. In addition, one of the biggest challenges in tracking is to match the correct objects to each other between two frames and how to handle an object that disappears and reappears again [11]. Almost all trackers have a unique way of solving this, both in complexity and speed. A popular tracker is the Deep SORT [82] tracker.

Figure 3.1 shows the tracking of two objects labeled as a person with id “1” and “2”.

Figure 3.1: Tracking of two objects

3.2 Kalman filter

Kalman filter is an optimal state estimator. It can be used to remove noise and inaccuracy on measurements or estimate system state parameters that can not be measured or observed. For instance, in object tracking, the Kalman filter can estimate the unmeasured parameter velocity of an object from the measured position of an object with high accuracy [23]. Based on [58] the Kalman filter algorithm works as follows:
Initialize the filter at:

$$\begin{aligned} \hat{\mathbf{x}}_0^- &= \mathbf{m}_{x_0} \\ \mathbf{P}_0^- &= \mathbb{C}_{x_0} \end{aligned} \tag{3.1}$$

Then:

1. Compute Kalman gain:

$$\mathbf{L}_k = \mathbf{P}_k^- \mathbf{C}^T (\mathbf{C}\mathbf{P}_k^- \mathbf{C}^T + \bar{\mathbf{R}}_\mathbf{v})^{-1} \tag{3.2}$$

2. Update estimate with measurement:

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{L}_k(\mathbf{y}_k - \mathbf{C}\hat{\mathbf{x}}_k^-) \tag{3.3}$$

3. Update error covariance matrix:

$$\mathbf{P}_k = (\mathbb{I} - \mathbf{L}_k\mathbf{C})\mathbf{P}_k^- (\mathbb{I} - \mathbf{L}_k\mathbf{C})^T + \mathbf{L}_k \bar{\mathbf{R}}_\mathbf{v} \mathbf{L}_k^T \tag{3.4}$$

4. Project ahead:

$$\begin{aligned} \mathbf{P}_{k+1}^- &= \mathbf{A}\mathbf{P}_k^- \mathbf{A}^T + \mathbf{Q}_\mathbf{w} \\ \hat{\mathbf{x}}_{k+1}^- &= \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k \end{aligned} \tag{3.5}$$

   repeat with k = k+1

Where, $\mathbf{m}_{x_0}$ = mean (expected value), $\mathbb{C}_{x_0}$ = covariance matrix, $\mathbf{L}_k$ = Kalman gain,

$\mathbf{P}_k^-$ = a priori covariance matrix, $\mathbf{C}$ = output matrix, $\bar{\mathbf{R}}_\mathbf{v}$ = noise covariance,

$\hat{\mathbf{x}}_k$ = a posteriori estimate, $\hat{\mathbf{x}}_k^-$ = a priori estimate, $\mathbf{y}_k$ = measurement,

$\mathbf{P}_k$ = a posteriori covariance matrix, $\mathbf{A}$ = transition matrix, $\mathbf{B}$ = input matrix, $\mathbf{u}_k$ = inputs, and

$\mathbf{Q}_\mathbf{w}$ = disturbance covariance

3.3 Mahalanobis distance

Commonly Euclidean distance is used to measure the distance between two points in as many dimensions as desired and is given by:

$$d(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2} \tag{3.6}$$

A problem with this calculation is that it can provide the wrong result of how close two points are when the data are correlated. To solve this problem, the Mahalanobis distance formula can be used instead. The formula works by:

1. Transform the data into uncorrelated variables
2. Scale the data making the variance equal to 1
3. Calculate the Euclidean distance.

This gives the Mahalanobis distance formula:

$$D = \sqrt{(x-m)^T * C^{-1} * (x-m)} \tag{3.7}$$

Where D is the distance, x is the observations, m is the mean values of independent variables, and C^{-1} is the inverse covariance matrix of independent variables [60]. However, Mahalanobis distance can be unsuitable as the only association metric in an object tracker when the motion uncertainty is high, such as tracking through occlusions. [81].

3.4 Hungarian algorithm

The Hungarian algorithm is used to solve assignment problems optimally. In object tracking, it can give objects their tracking ID. Based on [59] the algorithm works as follows:
First, a square matrix is needed. If the matrix is not square, add dummy rows or columns that have a value equal to the largest value in the whole matrix.
Then:

1. Subtract the smallest value in each row from each value in that row
2. Subtract the smallest value in each column from each value in that column
3. Draw lines over rows and columns containing zeros, with the smallest number of lines possible. If the number of lines equals the number of rows in the matrix, stop here. Otherwise, continue to step 4.
4. Find the smallest uncovered value and call it x. Subtract x from all uncovered values in the matrix and add it to any elements covered twice. Then go back to step 3.

When the algorithm is finished, select a zero in each row and column such that each only contains one zero. Remove any dummy rows or columns. The final matrix is now the ideal assignment in the original matrix.

The following theory, with some minor tweaks, is taken from chapter 3 in the preproject, found in appendix C

3.5 Computer vision

Computer vision (CV) deals with how a computer can understand digital images and videos without the help of humans. CV is a very complex problem, and much time and effort are put into this field each year. Some CV uses are object detection, video tracking, and image restoration [33].

3.5.1 Camera

In CV, the input is often captured using a camera in the form of images or video. Cameras can have different hardware, resulting in, among other things, different frame rates, resolution, and shutter speeds. In addition, the light in the shot, surrounding medium, and space can also affect the camera. Therefore it is essential to look at all these variables to get the best camera for the right task [29].

3.5.2 Thermal camera

Instead of using visible light like conventional cameras, thermal cameras use infrared (IR) radiation. These cameras typically work in the wavelength of 1 micrometer (1μ) to 16μ. These wavelengths mean that the camera captures heat instead of visible light. Therefore, the cameras have many applications ranging from military use to surveillance to data center monitoring. One of the biggest benefits of thermal cameras is better visibility in the dark [41]. There are several ways to visualize thermal images, as seen in figure 3.2 and 3.3.

Figure 3.2: Thermal with colors

Figure 3.3: Thermal in grayscale

3.5.3 Digital image

A computer typically represents a two-dimensional image with a two-dimensional array that contains all the pixels in a picture. For instance, a picture with a size of four by four pixels, for a total of 16 pixels, will have the array as shown in table 3.1. Each spot in the array can typically store 8 bits. 8 bits gives $2^8 = 256$, meaning a value from zero to 255 can be stored [44].

[[p(0,0)	p(0,1)	p(0,2)	p(0,3)]
[p(1,0)	p(1,1)	p(1,2)	p(1,3)]
[p(2,0)	p(2,1)	p(2,2)	p(2,3)]
[p(3,0)	p(3,1)	p(3,2)	p(3,3)]]

Table 3.1: Pixel placement in a picture

Storing an image is often done in grayscale or Red, Green, Blue (RGB) format. With grayscale, each pixel gets a value from zero to 255 that contains the intensity of light in that pixel. A pixel value of zero is black, 127 is gray, and 255 is white. The advantage of grayscale is the reduced size due to fewer pixels but still containing the essential information in the image. Grayscale also benefits from easier data code to process the images, as the arrays are less complex. A disadvantage is the loss of information about the colors in an image. A good use for grayscale is when the object to detect does not depend on colors. An example of this is to recognize different shapes like circles and squares [44]. An example of a 4x4 image is in table 3.2 with the pixel values shown as an image in figure 3.4.

[[0	0	0	0]
[100	100	100	100]
[200	200	200	200]
[255	255	255	255]]

Table 3.2: Grayscale array

Figure 3.4: Grayscale image

The RGB format normally contains three channels where each channel represents a color. Table 3.3 represents a 4x4 RGB image and in figure 3.5, the colors from the pixel values are shown. As before, the value in each element is usually from zero to 255. The first element in a pixel represents the intensity of red. The second element represents green and the last blue. So an pixel with value [0 0 0] = black, [255 0 0] = red, [0 255 0] = green, [0 0 255] = blue, [255 255 255] = white and any other combination gives the rest of the possible colors for a total of 16,777,216 different colors. The advantage of RGB is that there is a lot more information in the picture to use for the detection. The disadvantage is that the image gets bigger and more complex. Good uses of RGB are when the system is dependent on colors, such as making a system that can detect traffic lights [44].

[[[0 0 0]	[0 0 0]	[0 0 0]	[0 0 0]]
[[255 0 0]	[255 0 0]	[255 0 0]	[255 0 0]]
[[0 255 0]	[0 255 0]	[0 255 0]	[0 255 0]]
[[0 0 255]	[0 0 255]	[0 0 255]	[0 0 255]]]

Table 3.3: RGB array

Figure 3.5: RGB image

3.5.4 Image processing

Image processing uses an algorithm to highlight desired information in the image. These can be easy tasks such as scaling or changing an RGB picture to grayscale or vice-versa to more advanced tasks such as image enhancement and image restoration to the most advance as feature extraction with more [40].

3.5.5 Feature extraction

Feature extraction takes an image through an algorithm that highlights some exciting parts or traits in the image. Feature extraction minimizes redundant data and speeds up training, and the time it takes to make predictions. There are many methods for extracting different features, and all have their own set of uses. For example, some of the methods used in feature extraction and object recognition are edge- and corner detection [68].

- Edge detection is to find the edges in an image. Edge detection detects discontinuities in brightness in the image, and there are several methods for achieving this. In figure 3.6 an example of the Canny method can be seen [42].

Figure 3.6: Edge detection example [42]

- As in edge detection, corner detection has several methods. However, the main idea is to take a small window over every pixel and see if there is a change in the pixel values in all directions [15]. The main idea is visualised in figure 3.7.

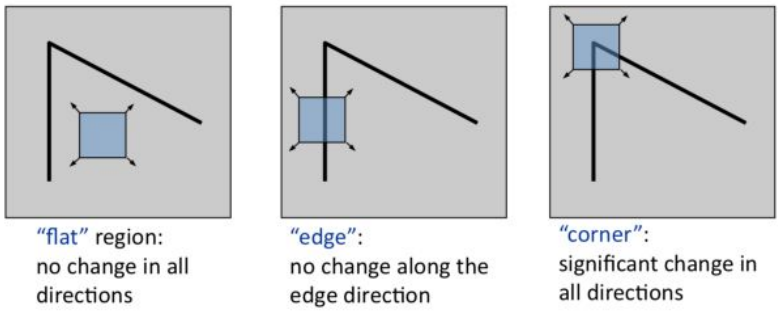


Figure 3.7: Corner detection example [15]

3.5.6 Region of interest

A region of interest (ROI) in an image is a part of the image of extra interest. Making an ROI usually means cutting away some parts of the image to reduce the size, change the aspect ratio or highlight some parts more. As a result, the bandwidth required for each image can be reduced, often achieving a higher frame rate when running detection [34].

3.6 Machine learning

Machine learning (ML) uses algorithms on data to learn more and more over time. ML can be several things, such as computer vision and recognizing the difference between a person and a dog. It could also be complex pattern recognition to help solve cancer and so on [14]. Figure 3.8 displays the difference between artificial intelligence (AI), ML, and deep learning (DL).

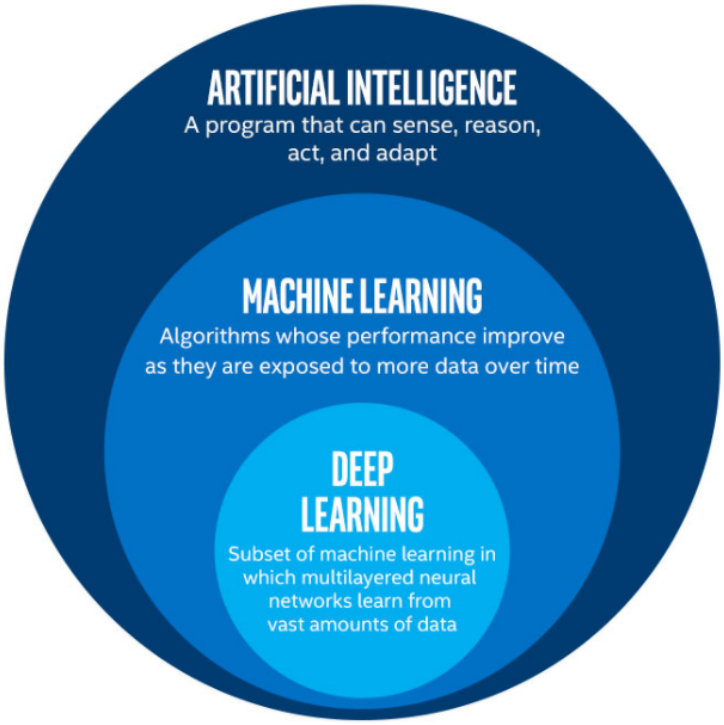


Figure 3.8: Difference between AI, ML and DL [74]

3.6.1 Training

Supervised and unsupervised are two methods used for learning in ML. Supervised learning is when the dataset is labeled and fed into the algorithms to train. The algorithm then adjusts its weights until the model performs appropriately. For example, object detection and classification often use supervised learning. Unsupervised learning is when a model gets a new dataset to analyze. When analyzing, the model finds patterns or data grouping without interaction from a human [14].

3.6.2 Neural networks

Using neural networks is a way to perform ML. Figure 3.9 shows an easy example of a neural network. A neural network consists of an input layer, one or more hidden layers, and an output layer, and neurons connect each of these layers [8]. The following sections will explain these building blocks.

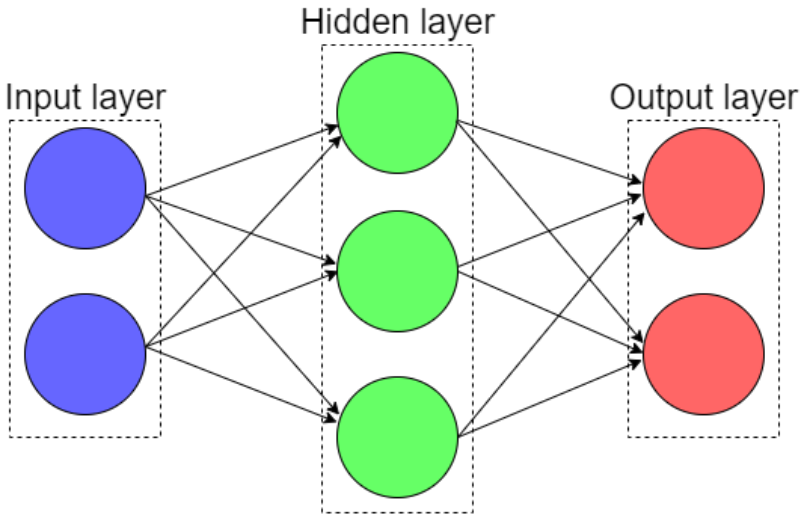


Figure 3.9: Neural network

3.6.3 Input layer

The input layer is the entering point of raw data. For instance, the input layer can be a neuron for each pixel in a picture. The data is often preprocessed to fit the total number of neurons in the input layer. For example, preprocessing can scale a picture up or down in size or make an ROI. After preprocessing, the network is ready to send the data to the hidden layer [8].

3.6.4 Hidden layer

The hidden layer does the main work of a neural network. Each hidden layer tries to learn something about the data by minimizing a cost function. An example is to think that the first hidden layer learns edge detection, the subsequent learns corner detection, and the next something else until there are no more hidden layers. The example is not exactly what is going on, but it is the main idea. The result is a complex problem broken down into more minor problems [8].

3.6.5 Output layer

The output layer returns a value for each of the neurons in the output layer. For instance, if the network displayed in figure 3.9 is an image classification between dogs and cats, the output layer will then output the confidence about each class. For example, the output can be [0.97 0.03], the model is then 97% sure the image is a dog and 3% sure it is a cat [8].

3.6.6 Neurons in deep learning

Each neuron, the colored circles in figure 3.9, are connected as shown in the figure. Each of these connections contains a weight that is a positive or negative number. When a value gets sent from one layer to the next, it gets multiplied by its weight. The neuron in the next layer then sums all the inputs from all the neurons in the previous layer.

After summing up the value, it gets to an activation function. The job of the activation function is to map the sum to the desired value. For example, using the unit step function, figure 3.10, will map all negative values to zero and all positive values to one. The activation function, in different ways, generally maps the output of a neuron in a range between zero and one. This process occurs in every neutron until the output gets to the output layer. The connection between neurons can also contain a bias. Bias is just a number that shifts the activation function left or right. [45]. Figure 3.10 shows some of the most common activation functions.

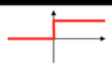
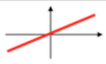
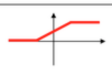
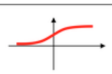
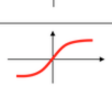
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer NN	

Figure 3.10: Activation functions [65]

3.6.7 Backpropagation

When training a model, the weights and biases need to be adjusted, which is called backpropagation. First, the network is given input and goes through the procedure explained in the previous paragraph. Then, the error at every neuron is calculated and saved often using:

$$Error = \frac{1}{2} * (prediction - actual)^2 \tag{3.8}$$

Then the backpropagation process starts using gradient descent to update the weights and biases. This process starts at the output layer and goes back towards the input layer, and repeats until the desired performance level is achieved [30]. A typical way to update the weight and bias at each neuron is using this formula:

$$\text{New weight} = \text{old weight} - (\frac{\partial \text{Error}}{\partial \text{old weight}}) \tag{3.9}$$

3.6.8 Convolutional neural network

A convolutional neural network (CNN) is a DL algorithm often used for image classification. The CNN can take an image as an input and learn its features with enough training without the need for human help [66]. Figure 3.11 shows an example of the structure of a CNN. The following sections will explain the different layers.

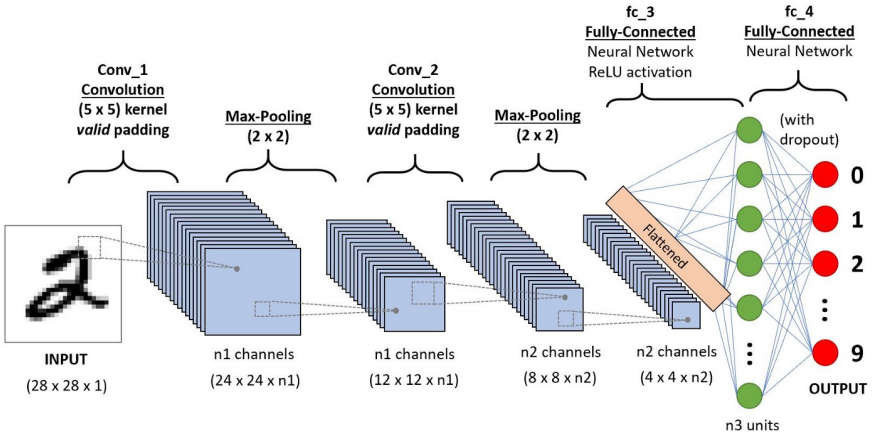


Figure 3.11: CNN for classifying handwritten digits [66]

3.6.9 Convolution layer

The convolution layer's job is to apply a filter to the input. This filter extract features such as edges and corners. Extracting features later in the layers gives more high-level features. Therefore adding a convolution layer to several places in the network makes the model more robust, as both high-level and low-level features are needed. The two methods for extracting features are the same padding and valid padding. The input is either increased in size or the same with the same padding. With valid padding, the input data is reduced [66].

3.6.10 Pooling Layer

The pooling layer reduces the size of the data and extracts dominant features. The two methods used are max pooling and average pooling. The max pooling returns the maximum value from the image portion covered by the filter. Average returns the average of these values. The most common is max pooling since this also performs a noise reduction [66]. In figure 3.12 the methods are shown. The max pooling takes some values, here four values, and returns the maximum. Average does the same but returns the average of these four values instead.

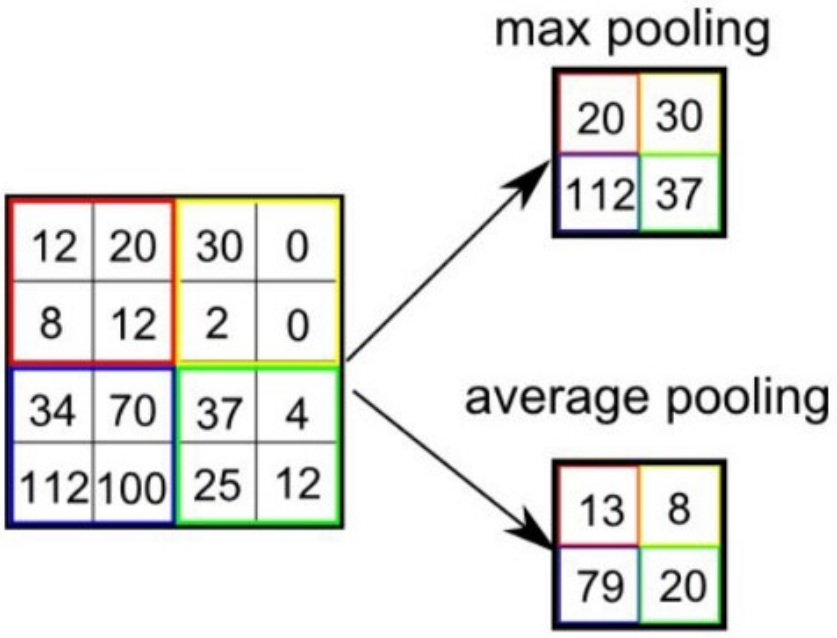


Figure 3.12: Max and average pooling [66]

3.6.11 Fully Connected Layer

The last layer in a CNN is the fully connected layer. This is often a neural network and works as described in the chapter 3.6.2. When training, the layer adjust its weights and biases with the help of backpropagation. The fully connected layer also gives the final value of the prediction [66]. As with all ML there are several ways to solve the problem. Some popular architectures are ResNet, AlexNet and YOLO [66].

3.6.12 Single stage detectors

When talking about state-of-the-art object detectors, there are mainly two types called single- and two-stage detectors [73]. This thesis uses the YOLO [6] detector, and this is a single-stage detector. Therefore the following paragraph will only explain YOLO.

YOLO works by dividing the image into X smaller grids, and for each grid, predicts bounding boxes and the confidence of each bounding box. Because the work happens in a single network, without any region proposal, it is a single-stage detector. The benefit is that it increases speed drastically but often reduces accuracy compared to two-stage detectors [72]. Since speed is one of the most important aspects of detecting a person falling, a single-stage detector was chosen. In figure 3.13 a visual representation of single- and two-stage detectors can be seen. The benefit of a two-stage detector is often the accuracy, but this comes with a speed reduction.

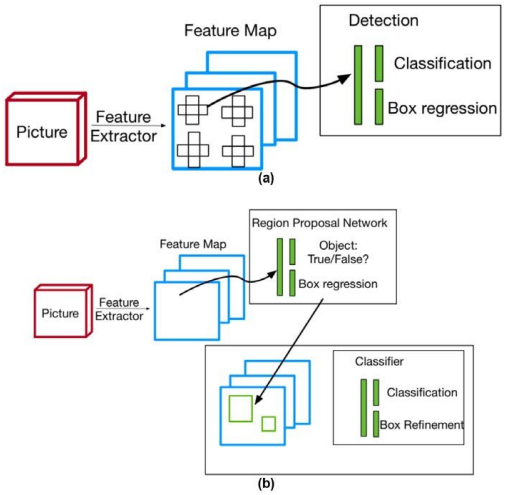


Figure 3.13: Difference in single- (a) and two-stage (b) detector [48]

3.6.13 Data augmentation

Data augmentation is to create new data by either using the existing data or creating new synthetic data. Synthetic data is data created artificially. Data augmentation with existing data can be flipping a picture horizontal or vertical, rotating the picture, changing the colors, and different scales with more. Augmentation can be done either offline or online. Offline is when data augmentation happens before training a model and is mainly used with a smaller dataset since this will increase the size of the dataset. For instance, by rotating each picture 90° , the dataset will double in size and can quickly run into space limitations when using multiple augmentations. Online augmentation is when the augmentation happens during training, like rotating a picture. The framework often has built-in support for doing this. Online augmentation is usually used for larger datasets since no new data is created [24].

3.6.14 Transfer learning

Transfer learning takes a pre-trained model that is already trained on a dataset. For instance, a model trained on the COCO dataset [10] that has over 200 000 labeled images with objects like boats, persons, dogs with many more, and take this knowledge on a new dataset like thermal images. There are several benefits from this. The pre-trained models are often trained on an extensive dataset with excellent hardware. This achieves a good starting point for a new model and usually means that a new dataset can be smaller and gain a better result [39].

3.6.15 Evaluation

When evaluating a model, it is important with a measurement that is equal for all models. In the following sections, the most standard methods will be explained, and to understand these methods, some abbreviation needs explanation:

- **True positive (TP)**: Both the real observation and prediction are positive.
- **True negative (TN)**: Both the real observation and prediction are negative.
- **False positive (FP)**: Real observation is negative, but the prediction is positive.
- **False negative (FN)**: Real observation is positive, but the prediction is negative.

3.6.16 Precision

Precision is the percentage of the predictions that are correct [32]:

$$Precision = \frac{TP}{TP + FP} \tag{3.10}$$

3.6.17 Recall

Recall calculates how good the model is at finding all the positives [32]:

$$Recall = \frac{TP}{TP + FN} \tag{3.11}$$

3.6.18 Intersection over union

Intersection over Union (IoU) is how much the prediction overlaps with the ground truth.

$$IoU = \frac{\text{Area of overlap}}{\text{Area of union}} \tag{3.12}$$

So when the IoU threshold is 0.5, the overlap needs to be over 50% to be classified as a true positive, and if the threshold is one, it needs to be a perfect overlap [32]. Figure 3.14 shows an example of IoU.

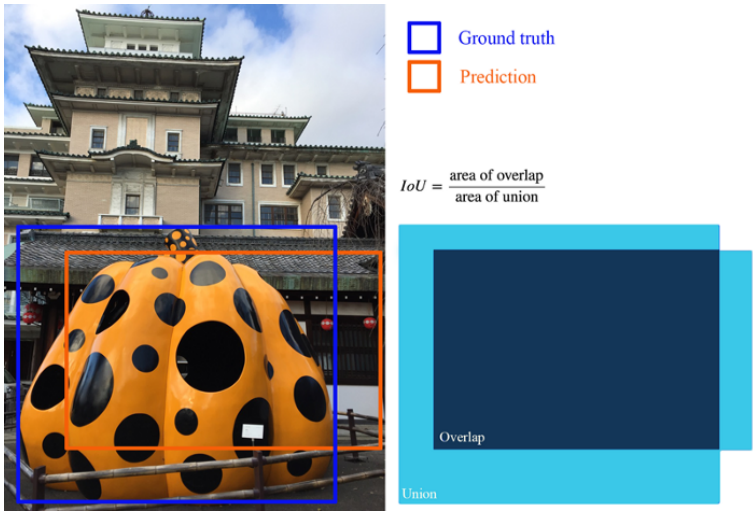


Figure 3.14: Explanation of IoU [32]

3.6.19 Mean average precision

Average precision (AP) or mean average precision (mAP) is often synonymous and is calculated by finding the area under the precision-recall curve. Sometimes the AP is calculated for each class, and then mAP is calculated by taking the average of the APs [32]. Figure 3.15 shows an example of how a precision-recall curve can look.

$$AP = \int_0^1 p(r)\,dr \tag{3.13}$$

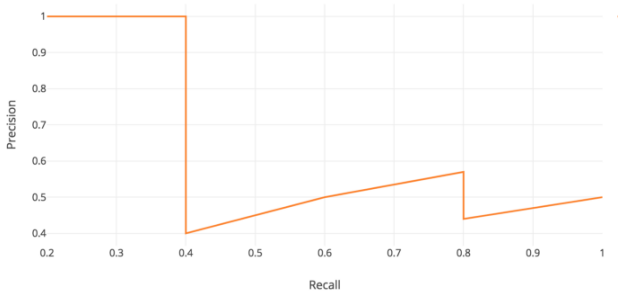


Figure 3.15: Example of precision-recall curve [32]

3.6.20 Total loss

Total loss is how well the model performs. The closer the loss is to zero, the better. When the model is training, it is this total loss it tries to reduce when changing its weights. If the total loss is very close to zero, overfitting can often occur [28].

Overfitting and underfitting

If a model's performance is terrible, then overfitting or underfitting may have caused the problem. Overfitting is when the model learns the dataset so well that the model can not generalize and performs poorly when presented with new data. Overfitting may not be easy to spot due to good performance on the training data. Underfitting is when there is insufficient data for the model to generalize. Underfitting is often easier to spot due to bad performance on the training data [7].

3.7 Programming language

Programming language is a formal language that makes it possible to write human text to a computer. The programming language's compiler makes the text into computer instruction and executes the code. Therefore, it is possible to make advanced programs and algorithms with minimal knowledge of computers [36].

3.7.1 Python

Python is an interpreted high-level, general-purpose programming language. The language focuses on code readability and is an object-oriented language. Python often gets described as a "batteries included" language for its substantial standard library and its easy access. As a result, python consistently ranks as one of the most popular programming languages [62].

3.7.2 Framework

A *framework* is a software or code that someone already has written. The code often emphasizes useability and makes it easy to manipulate for the user, and is often open-source, which means that everyone has access to the code and is allowed to change it.

There are several benefits of this. One such benefit is accelerated development. The amount of knowledge needed to succeed is lowered. The code is usually more efficient and easier to debug [27]. An example of a framework is Darknet [63].

Chapter 4

Materials and methods

This chapter starts with an explanation of the project approach. After this comes a presentation of the different materials used, such as cameras and hardware. Then a list of the different software and libraries used is presented. Next is the implementation of the dataset, the labeling, DeepStream, the convolution neural network, Deep SORT, and the different methods for detecting a person falling. Last is the implementation of how to save and plot the data.

4.1 Project approach

As mentioned, this thesis is done in collaboration with the company Zebop Avalon [3]. Zebop Avalon owns most of the equipment used and was responsible for installing the thermal camera on Solsiden. In exchange, Zebop Avalon can use the knowledge and software developed in this paper.

Bi-weekly a meeting with Zebop Avalon and the supervisor was held. Here the progress and any problems were presented and discussed. In addition, the offices of Zebop Avalon were often used as a workplace to get closer to people who could help with problems and discuss solutions.

4.2 Material

4.2.1 Camera on Solsiden

The camera mounted on Solsiden is a thermal camera from FLIR called A700. The camera has a FOV of 24° x 18°and can deliver thermal video in grayscale and color. It also films in standard RGB color at the same time. The camera delivers the thermal images at 30 frames per second FPS, with a resolution of 640 × 480 [17]. Figure 4.1 displays the camera .

Figure 4.1: Thermal camera [17]

4.2.2 Cameras for the dataset

Due to the camera on Solsiden being permanently mounted, there was a need for other cameras to collect data from other locations than Solsiden. FLIR also provided these cameras, and is called Boson 640 [16] and E96 [18]. The latter camera is a handheld camera that makes it easy to film in all locations due to its size and that it runs on a battery. The Boson 640 captures video at 60 FPS at a resolution of 640 × 512. The E96 captures video at 30 FPS at a resolution of 640 × 480. Figure 4.2 and 4.3 shows the two cameras.

Figure 4.2: Boson 640 Thermal camera [16]

Figure 4.3: E96 handheld camera [18]

4.2.3 Hardware

When finalizing the product, a FLIR Bridge [22] will be used as a hub to connect to the camera and run the software. The hardware inside this hub is based on a Jetson Nano [53]. Therefore, a Jetson Nano was used under development as this is much cheaper and offers the same experience.

The Jetson Nano is a small but powerful device with a size of only 70 mm x 45 mm, and figure 4.4 illustrates a Nano. The Nano comes with a four-core Central Processing Unit (CPU), 4GB of ram, and a Graphics processing unit (GPU) with 128 CUDA Cores [53].

Figure 4.4: Jetson Nano [53]

Since the Nano has limited performance, a more powerful computer was used for training and some of the development. This computer contains an Nvidia 1080ti GPU with 11 GB of RAM and 3584 CUDA Cores [51] and is 28 times as many cores as the Nano. The other components in the computer are 24 GB RAM at a speed of 3200 MHz. In addition, an Intel i7 7700k CPU with four cores, eight threads, and a clock speed of 4.2 GHz with a boost to 4.5 GHz [35].

4.3 Software and libraries

The listed software and libraries below have been used throughout this project.

4.3.1 Software

- **Overleaf** - A web page editor for LaTeX that gives the authors tools like real-time collaborative, spell checking and cloud-based storing [55]. Used to write the report and works as a backup system for the report.
- **Git** - A free open source distributed version control system [26]. Used to store important code.
- **Draw.io** - A free web page editor for making flowcharts, charts, and diagrams. [12]. It was used to make figures for explanations.
- **VLC** - A free open source media player with a rich set of functions [80]. It was used to save video from the thermal cameras.
- **LabelImg** - A graphical image annotation tool [78]. Used to label images for ML model training.
- **Visual studio code** - An integrated development environment (IDE) made by Microsoft that enable tools like debugging, syntax highlighting, intelligent code completion and have git implemented into it [47]. Used as the IDE to write python code.
- **DeepStream** - A software development kit from Nvidia used to develop AIs on Jetson Nanos [52]. It was used to implement optimized object detection and object tracking on a Jetson Nano.

4.3.2 Python libraries

The libraries below, combined with visual studio code [47], have been used to develop the software needed for this thesis.

- **OpenCV** - An open source computer vision library for real-time computer vision [54]. Used to read and manipulate captured video from the thermal camera.
- **Numpy** - A library used for high-speed mathematical operations and include support for large, multi-dimensional arrays and matrices [50].
- **Darknet** - An open source framework used with the YOLO object detection [63]. Used to run YOLO object detection.
- **Deep SORT** - Used with YOLO to achieve Realtime Tracking [82].
- **Pandas** - Open source data analysis and manipulation tool for python [56]. Used to save and load data points.
- **Matplotlib** - Used to display and save various plots from data points [43].

4.4 Implementation

The following sections explain the implementation of the different parts of the system. It starts with labeling the dataset. Then a look at how the YOLO model works and the training implementation. After this is the implementation of Deep SORT and detection of a person falling, and last is the implementation of data analysis with saving and plotting data. All of the source code can be found in appendix A. Unfortunately, the dataset can not be shared due to Zebop Avalon using it in their product. Due to grayscale's advantages, all images captured for training and testing use grayscale.

4.4.1 Dataset

The dataset was done in collaboration with Zebop Avalon, and contains 3249 labeled thermal pictures, with the labels person, car, dog, boat, truck, falling person, and bus. It was filmed in different locations, both indoors and outdoors, with different cameras and conditions. After the data was collected, the frames from the video were extracted for labeling. The number of extracted frames depended on the movement in the video. For example, if the video contained fast movement, more frames were extracted than videos with less movement. This was to optimize the data collected to get as useful data as possible from minimum data. Figure 4.5 shows different pictures from the dataset in different locations.

Figure 4.5: Pictures from the dataset

Labeling

The software used for labeling images is called LabelImg [78]. All objects in the label category mentioned in the last section were labeled by drawing a bounding box around each object visible in each image. The software then generates a .txt file for each image in the YOLO format [1]. This .txt file contains information about what objects are in the image and where the bounding box is. First is the label number, and then is the center's x and y coordinates of the bounding box, plus the width and height of the bounding box. All the coordinates values are normalized between 0 to 1. So if an image contains a person that is class 0 and a person falling, that is class 1. So the file for one image will look like this, without the top row, as this is just for explanation:

Class	x	y	width	height
0	0.692145	0.291546	0.114853	0.151257
1	0.141592	0.401254	0.351217	0.115789

Some examples of pictures labed can be seen in figure 4.6

Figure 4.6: Labeled images

4.4.2 DeepStream

DeepStream is a closed source software development kit (SDK) from Nvidia. An SDK is a set of development tools that allows developers to develop applications for a specific hardware platform, computer system, operating system, or similar, like the Jetson Nano. It speeds up development but does not need to be open-source, which can significantly reduce the customization wanted in a system [52].

DeepStream gives the ability to run platform-specified optimized object detection models and object tracking with minimal effort. DeepStream works by building a pipeline of predefined objects. The pipeline can be customized as wished but can only contain objects Nvidia has made available. These objects include, with more, an encoder, decoder, object detection, and object trackers. The source code for running DeepStream can be found in appendix A.1. A simplified pipeline of the pipeline used in this project can be seen in figure 4.7. A short explanation of each object:

- Decoder takes the input and makes it ready for the pipeline. It accepts a saved file or a Real-Time Streaming Protocol (RTSP) stream from a camera.
- Object detection, YOLO runs here.
- Object tracker, Deep SORT runs here.
- Analytics, make it possible to add line crossing warning, overcrowd alarm with more.
- On screen display makes it possible to draw boxes, rectangles, and text on the video.
- Encoder and sink output the pipeline's output as an RTSP stream so that it can be viewed on another computer using VLC or something similar.

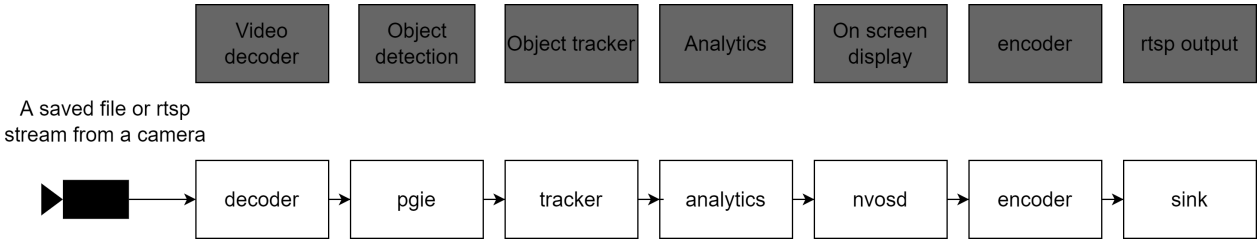


Figure 4.7: DeepStream pipeline

4.4.3 Hardware

The Jetson Nano was used to test the object detection and object tracker early on. However, due to DeepStream being a closed SDK, the development changed to the other computer that could use an open-source Deep SORT. This gave full access to Deep SORT and other system parts that would otherwise be closed by DeepStream. This was done because of the limited time of the project. It could be done on the Nano as well but would take more time due to the unfamiliarity of DeepStream.

4.4.4 Convolution neural network

Since the system should be able to run on the Nano, the YOLOv4-tiny model was used instead of the regular YOLOv4 model. The difference between the model is the size of the models. The tiny model is a smaller model with fewer layers in several places. An example of this is that the tiny model has two "YOLO head" layers, whereas the regular YOLO model has three. A smaller model makes the tiny model run much faster and therefore, able to run better on hardware with minimal computational power as the Nano has [67].

The architecture in figure 4.8 splits into three main parts. First is the backbone called CSPDarknet53, then is the neck that uses a Feature Pyramid Network (FPN), and last is the head that uses YOLO. The backbone's job is to extract features and give this to the FPN that builds a feature map from all the features. The last stage does the predictions and outputs the bounding boxes [67].

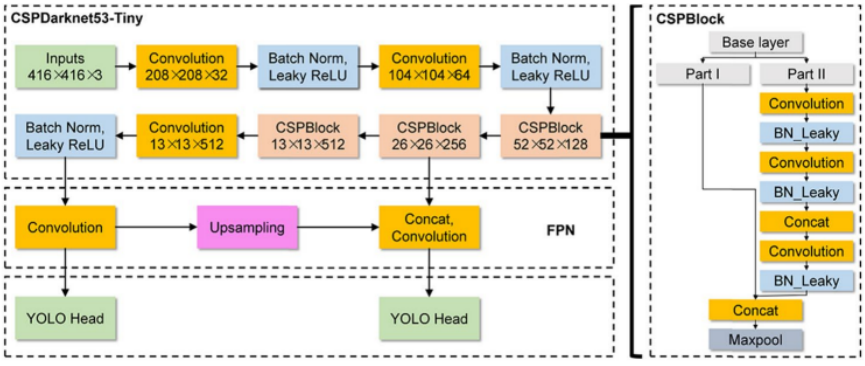


Figure 4.8: Yolov4-tiny architecture [67]

Training the YOLO model

For training the YOLO model in darknet [63], this tutorial [76] was used as inspiration. This tutorial shows everything needed to start training a model, from how to correctly save and label the dataset, change and make the correct setup files, and start the training. The dataset was split into 80% for training and 20% for validation. The online data augmentation used when training was to change saturation, exposure, and some color variation, but it did not change any angle of the images. The model was set to run for 56000 iterations with a batch size of 64 and a subdivision of 16. The batch size decides how many images are used for one iteration. The subdivision divides the batch images into smaller sizes, here $\frac{64}{16} = 4$, and then feeds four and four images to processing. When all the 64 images have been through processing, a new iteration will begin with 64 new images, and so on. During and when the training is finished, a set of checkpoints are stored. These checkpoints are called .weights files and are used to do the detection later. The weight file called XXXXXX_best.weights was used when testing the performance of the models.

4.4.5 Deep SORT

The object tracker uses the Deep SORT algorithm in this thesis because of the research done in the previous work chapter 2, and because it was possible to implement Deep SORT in DeepStream. In figure 4.9 the architecture of the Deep SORT pipeline can be seen.

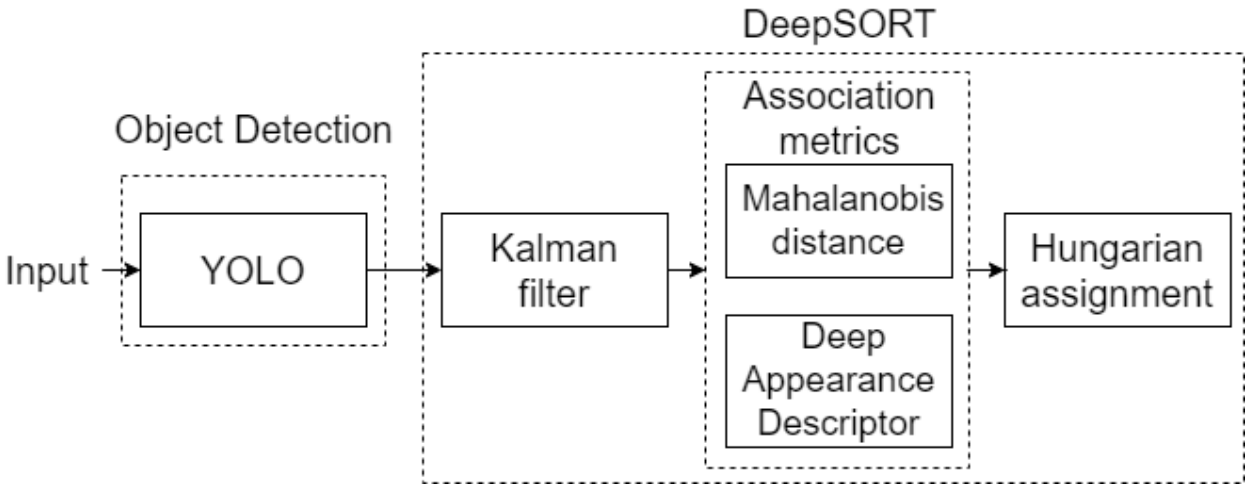


Figure 4.9: Deep SORT architecture. Figure based on [37]

The Kalman filter takes in the center of the bounding boxes, the height of the image, the aspect ratio, and their respective velocities in image coordinates. The job of the Kalman filter is to estimate where a bounding box will be in the next frame from the inputs. Then this information is sent to the association metrics block to find the same object between frames. Mahalanobis distance looks at the distance between predicted Kalman states and newly arrived measurements, and Deep Appearance Descriptor is a neural network trained to recognize features between objects. The outputs from these two methods then go to the Hungarian algorithm that assigns a tracker ID based on the bounding box overlap of predicted, and actual bounding boxes [82]. The code used to run Deep SORT in this thesis is built from the theAIGuysCode Github [77] that makes it possible to run Deep SORT with YOLOv4. This code was altered to the needs of this thesis, like saving data points and line detection, with more.

4.4.6 Detection of a person falling

Several methods for the detection of a falling person were implemented. The different methods can work together and produces a more precise system. The implementation of these methods will be explained in the following sections.

Object detection

The first method is object detection. This method is the backbone of the other methods and, therefore, the most important. The method uses the YOLO neural network to detect different objects. This method can raise the alarm when a label seen as a falling person is detected. The bounding boxes from YOLO are sent to the Deep SORT algorithm for tracking purposes in each frame.

Line detection

This method uses a predefined line, such as the red line in figure 4.10, to detect if something has crossed it. At each frame, the system draws an invisible line, shown in black in figure 4.10, through the middle of all the bounding boxes in the frame. An alarm can be made if the invisible line crosses the red line. The implementation does not care what label the object has, but this can easily be changed to be only for a person, a person falling, or both. The source code can be found in appendix A.2

Figure 4.10: Line crossing check

Adaptive line

The adaptive line calculates the mean walking trajectory every 20th second from the Deep SORT history. It then makes a line that goes through the mean trajectory and moves it 200 pixels down in the front and 50 pixels in the back due to the camera angle. The adaptive line gives a line that gets updated every 20 seconds and will fit better and better the movement pattern of people as time goes on. The source code can be found in appendix A.3

Relative movement detection

The movement detection looks at the history of the movement of an object given by the object tracker. The movement is relative because it is not a depth camera. The movement detection takes the last bounding box pixel coordinates and the second last box and calculates the distance with:

$$distance = \sqrt{(point2x - point1x)^2 + (point2y - point1y)^2} \tag{4.1}$$

If this distance is under 1, the object is standing still. If it is between a set threshold, the object is moving slowly, and last if the distance is above the threshold, the object is moving fast. Due to this being relative movement, an object that moves one meter and is far away will travel a smaller distance than an object closer to the camera. The source code can be found in appendix A.2

4.4.7 Data analysis

In the following sections, the implementation of the data analysis will be explained. First is the training of a new model, and then how data points were saved and later plotted.

Data analysis YOLO model

A new model was created to get the best possible data from Solsiden. A total of 61 images were extracted from the used fall video. Before labeling, the images were flipped horizontally to avoid the model overfitting too much on the data. An example of a picture before (left) and after (right) the flip can be seen in figure 4.11. The model learned from the best weights from the previous model and had the same setup as before. The 61 flipped images were used as training data, and the 61 original images were used for validation.

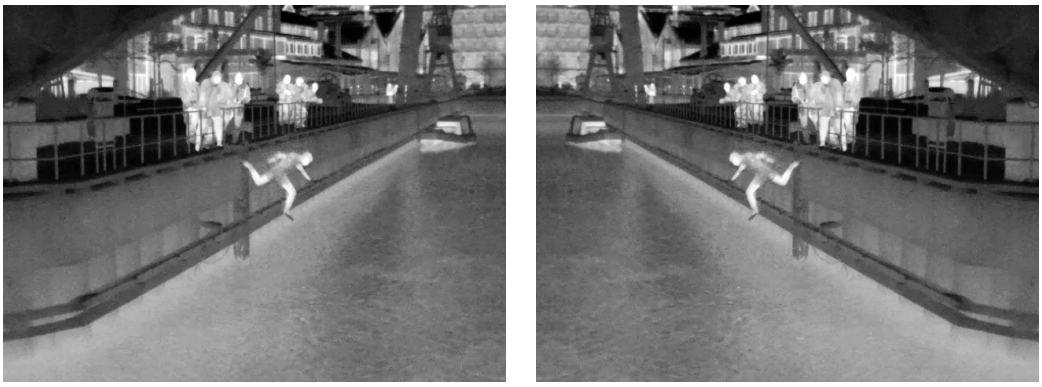


Figure 4.11: Flipped images

Saving data points

The Deep SORT [82] code was modified to store data points by making an array that contained arrays. At each frame, the x and y coordinates of each bounding box's midpoint in that frame get added to the correct array. The correct array corresponded to which object tracker ID number the bounding box had. By doing it this way, the arrays to an already known ID would expand, and if a new ID were detected, it would add a new array to the main array. After the video had ended, Pandas [56], a python library, was used to save the array to a .csv (Comma Separated Values) file.

Figure 4.12 shows an example layout of a file. The first row shows the number of each data. Rows one to six are all an array containing all the data points of that ID. The video used here contained five different tracked objects with various amounts of data points saved for each. The source code can be found in appendix A.2

	A
1	,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41
2	1,"(260, 222)","(261, 221)","(262, 221)","(264, 221)","(264, 222)","(265, 222)","(266, 224)","(267, 223)","(267, 222)",
3	2,"(28, 183)","(29, 183)","(30, 182)","(30, 182)","(30, 182)","(31, 182)","(32, 183)","(33, 184)","(36, 183)","(38, 184)"
4	3,"(485, 305)","(494, 321)","(498, 337)","(507, 358)","(512, 372)",,
5	4,"(266, 197)","(266, 198)","(266, 199)","(267, 199)","(267, 199)","(267, 199)","(268, 200)","(292, 200)","(293, 200)",
6	5,"(272, 198)","(273, 198)","(272, 197)","(272, 197)","(273, 197)","(273, 197)","(273, 197)","(274, 197)","(275, 196)",

Figure 4.12: Example of stored data points

Plot

Pandas were used to load the data from the .csv file. After the initial loading of the data, Pandas build the arrays back together to make it easier to access and iterate through the data. Then the data was given to several methods that each made a plot highlighting something interested in the data, such as plotting all the saved trajectories in the same picture. Matplotlib [43] was used to plot the different plots, and the source code of all the different plots is in appendix A.4

Chapter 5

Result

First in this chapter is a brief look at the DeepStream result. DeepStream is a short section due to the development shifted away from DeepStream. Then is a look at the result from training the model. After this, the result from the various methods for detecting a person falling is presented. Last is a look at the performance of Deep SORT and plotting relevant data from Deep SORT.

5.1 DeepStream

The following video, https://youtu.be/EfhruB82EKE, shows a running version of the DeepStream pipeline. The YOLO model running is old, so the performance is not as good as later models. figure 5.1 shows an example of the DeepStream performance

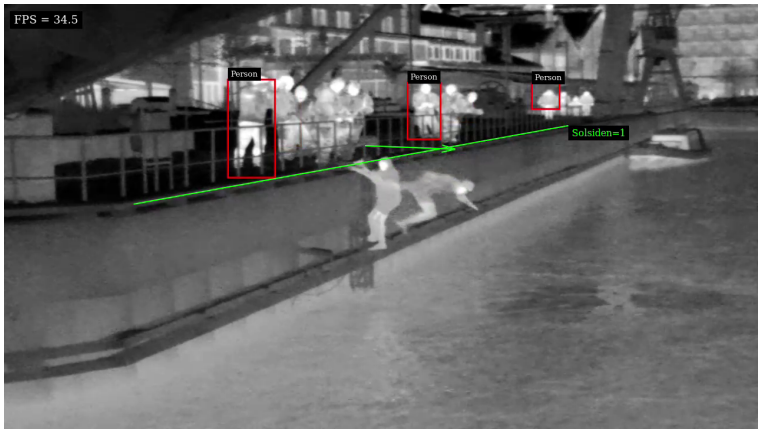


Figure 5.1: DeepStream performance

An early version of the line intersection check shows in the video, and figure 5.2. Due to having an old detection model and being very early in development, some bugs were present that were fixed later in the development when switching to the other hardware. For example, bugs such as the line check do not always detect a person crossing.

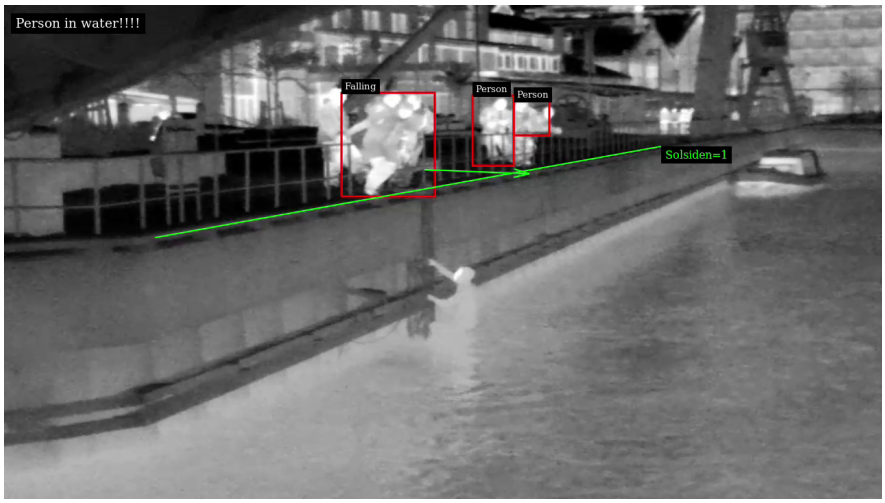


Figure 5.2: DeepStream line intersection

5.2 Object detection training

In figure 5.3 the training progress of the model from zero to 56000 iteration is shown. The training time was about nine hours, and the final mAP and loss were 39%-40% and about 0.2, respectively.

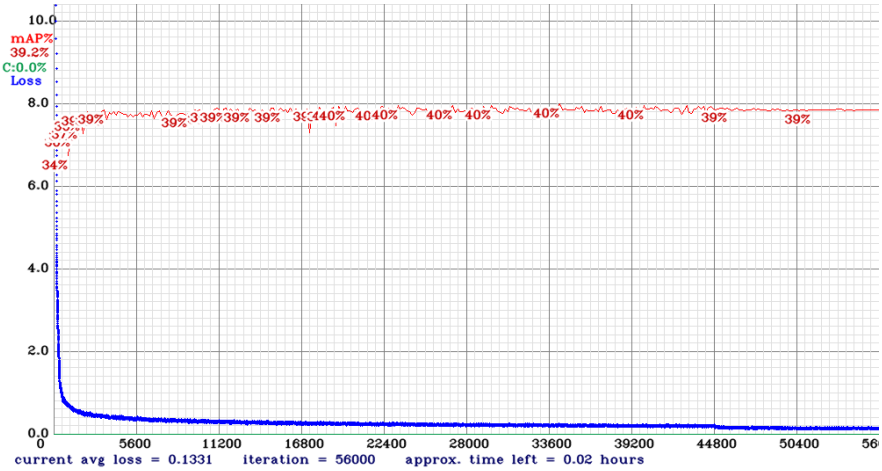


Figure 5.3: Training

The mAP mentioned last section can be somewhat misleading. As seen in figure 5.4 not all objects that was labeled was in the dataset. This is because the complete dataset contains over 70000 RGB images from the COCO dataset [10] that contains all the labeled objects. These images was removed due to time limitation as this would bring the training time from nine hours to weeks. So a more correct mAP would be $\frac{92.49\% + 100\% + 81.96\%}{3} = 91.48\%$ instead of $\frac{92.49\% + 100\% + 0\% + 0\% + 0\% + 81.96\% + 0\%}{7} = 39.2\%$ as the figure 5.3 shows.

```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
652
 detections_count = 2308, unique_truth_count = 1958
class_id = 0, name = Person, ap = 92.49%         (TP = 1705, FP = 139)
class_id = 1, name = Car, ap = 100.00%           (TP = 9, FP = 0)
class_id = 2, name = Dog, ap = 0.00%     (TP = 0, FP = 0)
class_id = 3, name = Boat, ap = 0.00%            (TP = 0, FP = 0)
class_id = 4, name = Truck, ap = 0.00%           (TP = 0, FP = 0)
class_id = 5, name = Falling Person, ap = 81.96%         (TP = 81, FP = 19)
class_id = 6, name = Bus, ap = 0.00%     (TP = 0, FP = 0)

 for conf_thresh = 0.25, precision = 0.92, recall = 0.92, F1-score = 0.92
 for conf_thresh = 0.25, TP = 1795, FP = 158, FN = 163, average IoU = 76.15 %
```

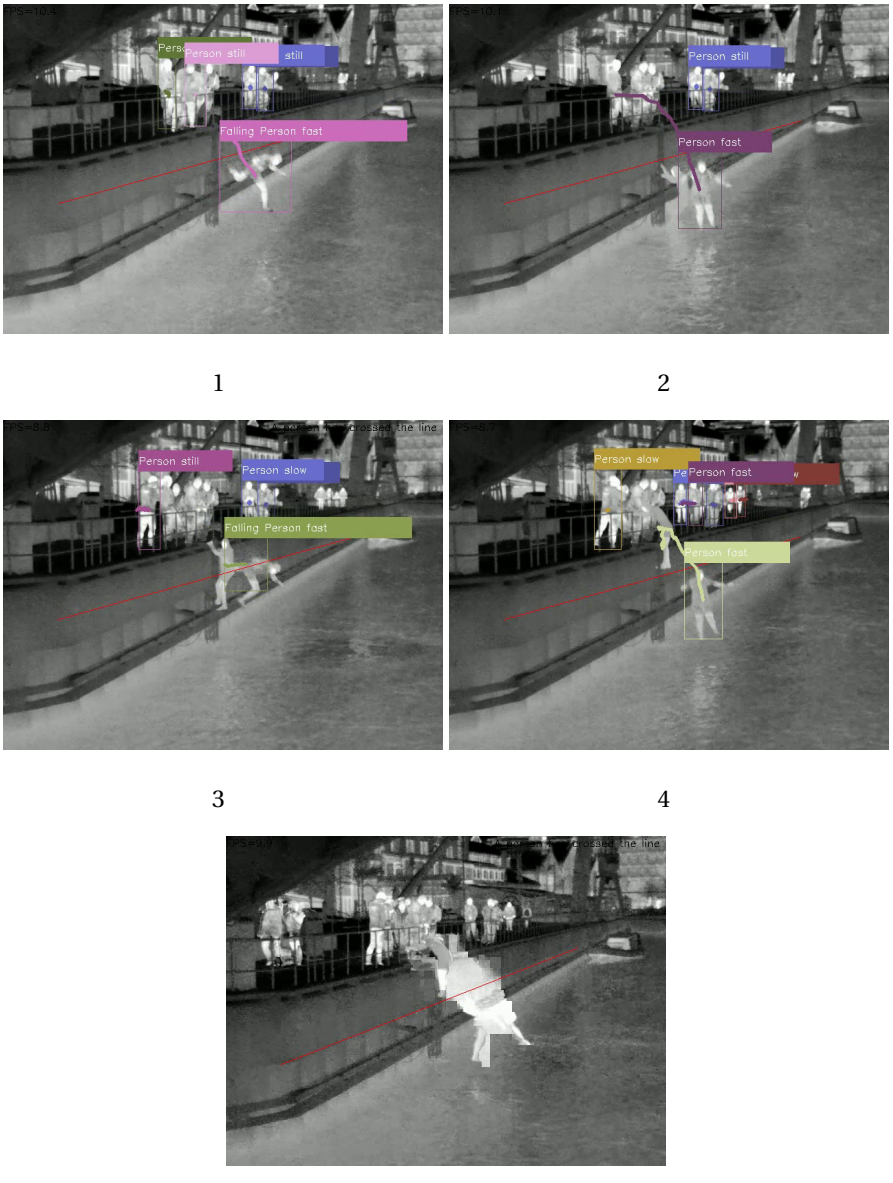
Figure 5.4: Training score

5.3 Detection of a falling person

The following sections present the results of the different methods for detecting a person falling. The following 2-minute and 27-second video was used to obtain all the results listed in the following sections https://youtu.be/KQ4kELgAeyw.

5.3.1 Object detection

The backbone of all the detection methods is object detection. Therefore, it must be as best as possible. In the video, a total of five persons are falling into the water, as shown in figure 5.5. The system detects a person falling correctly in picture 1 and 3 of the five falls in figure 5.5. In picture 2 and 4, it detects a person and contains a good history of the person falling. The system did not detect anything during the last fall.



1 2

3 4

5

Figure 5.5: Detection of a falling person

5.3.2 Line detection

As long as the object detection makes a bounding box that crosses the line, it will be detected. It only needs one frame and works 100% of the time. The line check can easily make an alarm, and in figure 5.6 an example of this can be seen with the warning saying “A person has crossed the line” in the top right corner when someone has crossed the line.

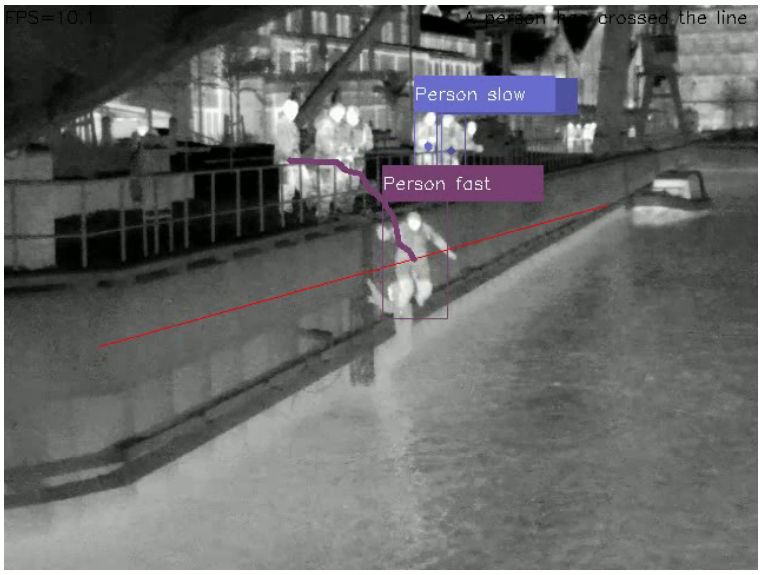


Figure 5.6: Line crossing check

5.3.3 Movement detection

In this system, the movement detection was unstable. For example, in figure 5.7, which is back to back frames, it is possible to see that both the color, therefore the ID, and label of the bounding box changed. The fall history cannot be captured correctly when this happens, and the movement detection is not working. Furthermore, the object detection sometimes only recognizes a few frames of a fall, and the movement detection will not get enough data to work. However, as seen in pictures 1, 2, and 4 in figure 5.5 sometimes it works better, and a person moving fast can be detected.

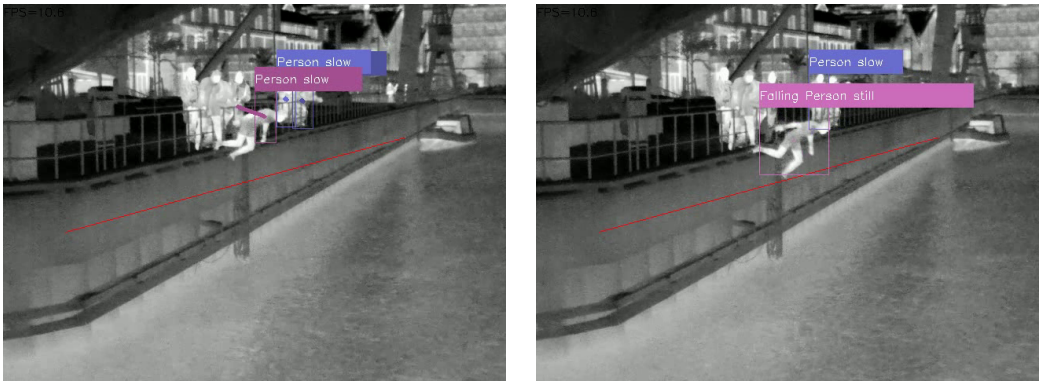


Figure 5.7: Movement detection

5.4 Deep SORT result

The Deep SORT algorithm managed to run at ≈ 9 FPS. A max-age of 20 was used to obtain the following result. The max-age tells Deep SORT how many frames an object can be missing from the frame before a new ID is made when and if the object returns. The 2-minute and 27-second video made 219 ID trajectories with different lengths. The shortest trajectory was one point, and the longest was 2782 points.

5.5 Trajectory analysis

The results from the new YOLO model will be presented in the following sections. First, how it did under the training phase, and then the data analysis results.

5.5.1 New YOLO model

The new model was transferred learned from the best weights from the previous model. Therefore it starts its training at iteration ≈ 33600 and not zero, as shown in figure 5.8. The training time was circa three hours and gave a model with 25% mAP and a loss close to zero.

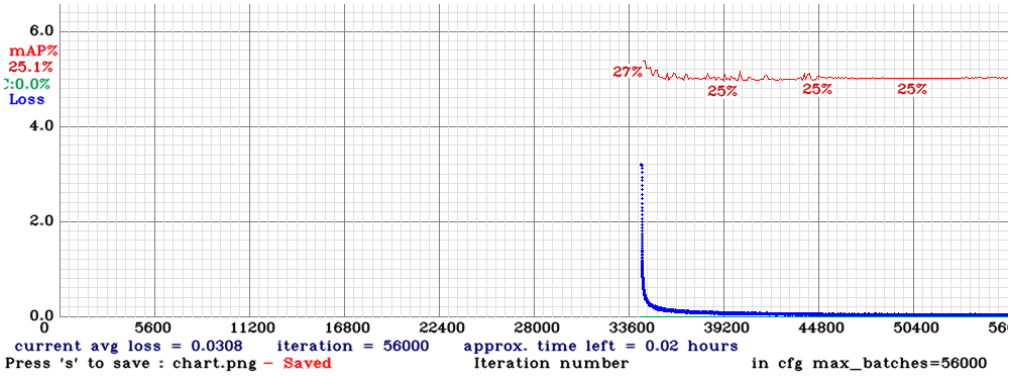


Figure 5.8: All trajectories extracted from a video

The same problem as the previous model occurs here also with not all labels being present in the dataset. So a more correct mAP, as shown in figure 5.9, is $\frac{84.34\% + 91.38\%}{2} = 87.86\%$ instead of the 25% mAP in the previous figure 5.8.

```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
36
 detections_count = 236, unique_truth_count = 255
class_id = 0, name = Person, ap = 84.34%         (TP = 195, FP = 12)
class_id = 1, name = Car, ap = 0.00%      (TP = 0, FP = 0)
class_id = 2, name = Dog, ap = 0.00%      (TP = 0, FP = 0)
class_id = 3, name = Boat, ap = 0.00%             (TP = 0, FP = 0)
class_id = 4, name = Truck, ap = 0.00%            (TP = 0, FP = 0)
class_id = 5, name = Falling Person, ap = 91.38%          (TP = 28, FP = 1)
class_id = 6, name = Bus, ap = 0.00%      (TP = 0, FP = 0)

 for conf_thresh = 0.25, precision = 0.94, recall = 0.87, F1-score = 0.91
 for conf_thresh = 0.25, TP = 223, FP = 13, FN = 32, average IoU = 73.97 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.251024, or 25.10 %
Total Detection Time: 1 Seconds
```

Figure 5.9: All trajectories extracted from a video

5.5.2 Data analysis

This video shows the performance of the new model, https://youtu.be/tMXxFmUmMcw. The data collected from this video is used to plot the figures in the following sections. The source code for all plots can be found in appendix A.4.

5.5.3 Plot of trajectories

Figure 5.10 shows all the trajectories extracted from the video. Each trajectory has its color and corresponds to one tracker ID from Deep SORT. The total number of tracker IDs was 69, with lengths from three to 4277 data points in each tracker ID.

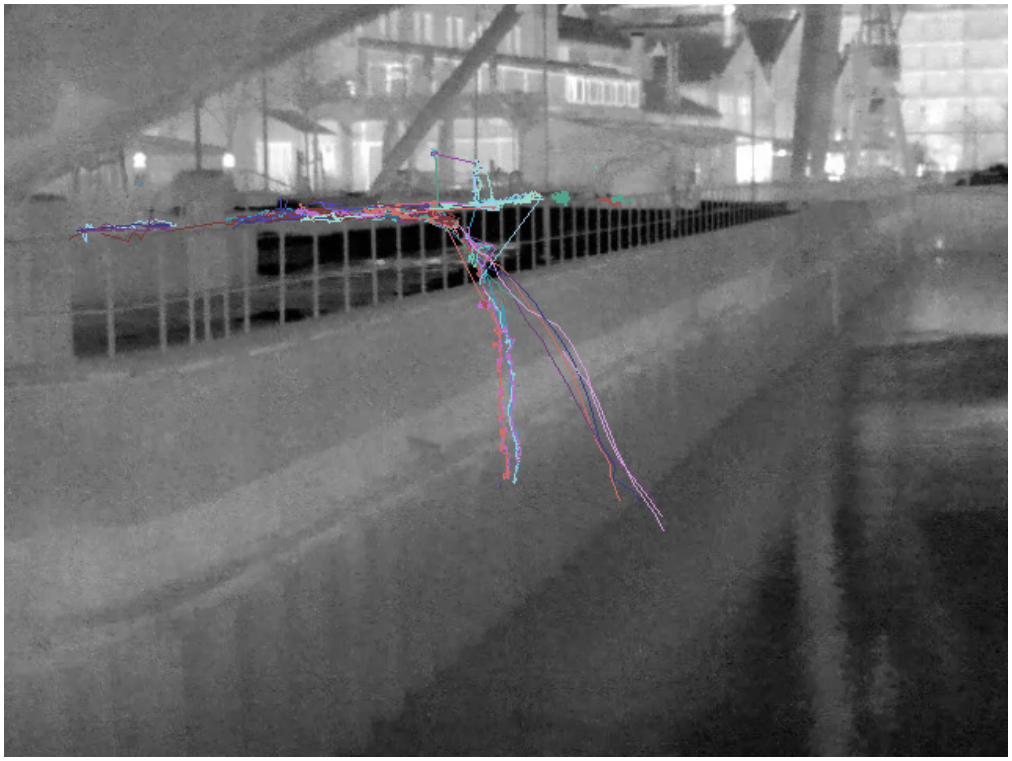
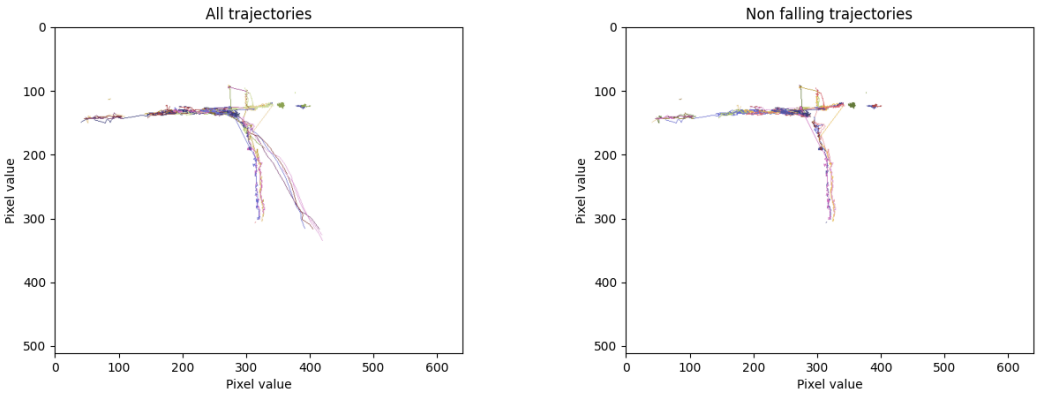


Figure 5.10: All trajectories extracted from the video

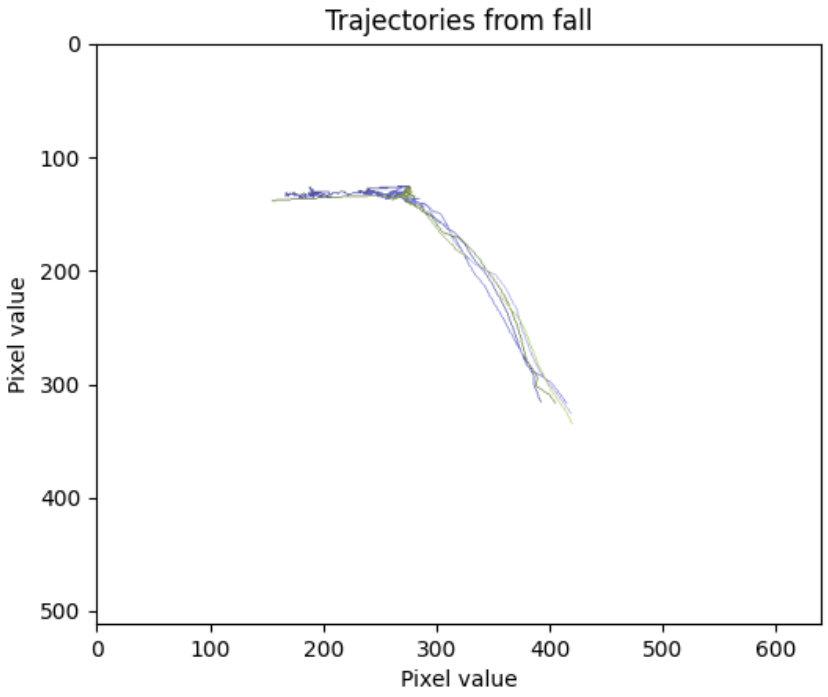
Several things were done to see if it was possible to distinguish a person falling from a person not falling from the Deep SORT data. First was dividing the data into falling or not falling trajectories as shown in figure 5.11. Plot (1) shows all trajectories. It is the same trajectories as in figure 5.10, but in a graph instead. Plot (2) shows all trajectories categorized by the software as a non falling trajectory. The vertical trajectories are the trajectories from when the people climb up

the ladder from the water. Finally, plot (3) shows all trajectories categorized by the software as falling trajectories. A total of six trajectories were categorized as this. However, it was only five falls in the recorded video. The extra fall is due to an ID switch in the last fall.



1

2



3

Figure 5.11: (1) All trajectories (2) Non fall trajectories (3) Trajectories from fall

To further analyze the movement, four trajectories from a fall and four from a non fall were divided into their X and Y values and plotted as shown in figure 5.12. The red lines are the Y values that the four non falling trajectories had, and the yellow dashed lines show the X values of a person that is not falling. The green dashed lines show the X value when falling, and the blue lines show the Y value.

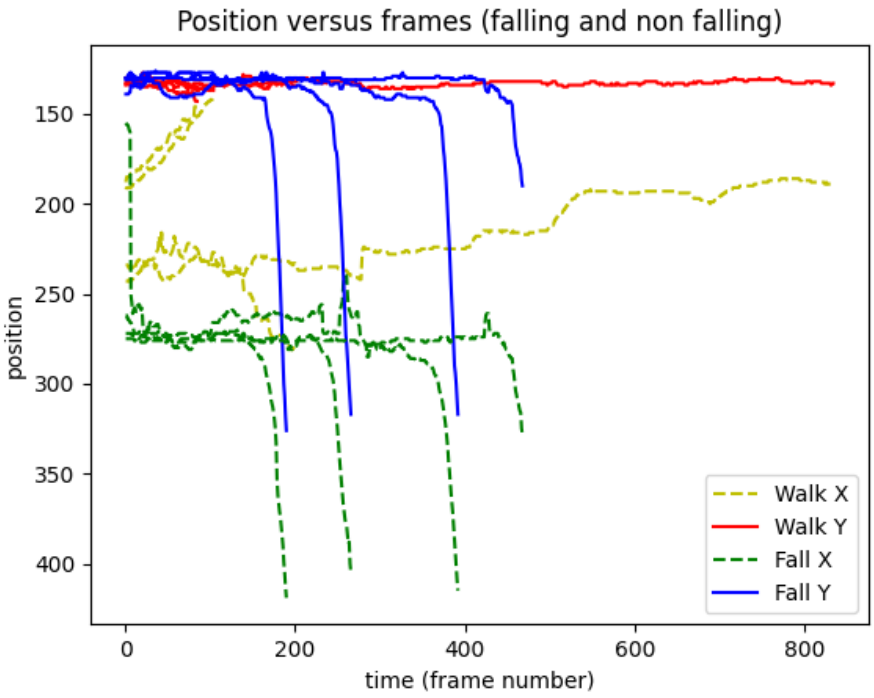


Figure 5.12: X and Y values compared between non falling and falling trajectories

One additional relevant piece of data is when a person is climbing up the ladder. Figure 5.13 shows one such trajectory. Taking this trajectory and splitting it into X and Y values gives the figure 5.14. From this figure, it is possible to see the Y value steadily growing as time goes on and the person is climbing. However, the X value stays relatively steady.

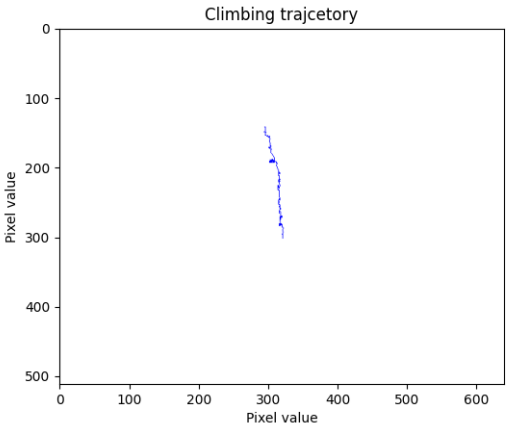


Figure 5.13: A climbing trajectory

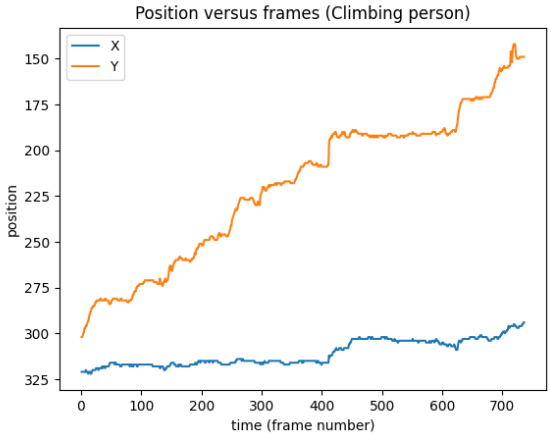


Figure 5.14: X and Y values of a person climbing

Appendix B shows more plots highlighting different movement patterns.

Chapter 6

Discussion

The following sections will discuss the different solutions and results obtained in the previous chapters.

6.1 Hardware and YOLO model

The system's limitation to run on a Nano made choosing between various detection models impossible. However, removing this limitation opens several new possibilities to improve the system's performance. The computational power of the Nano seems to limit the performance a lot. Instead of using a small computer close to the camera, the camera can stream the video to a more powerful computer. In this way, the performance will improve due to a more powerful computer, and it could be possible to change to a full-size YOLO model that will also perform better. A problem with this approach is the cost of such a system. The increased price means that it may not be economically beneficial when looking at it from a business standpoint. Another problem with this approach is the loss of frames when streaming the data. Losing frames when a person is falling means that the system can not detect a person, and in the worst case, a person falling can go undetected.

Looking at the training performance of the YOLOv4-tiny model on the validation set, the model scored 92.49% on detecting a person, 100% on detecting a car, and 81.96% on detecting a person

falling. Dropping the car label due to few test samples, only nine images, the mAP of the system is $\frac{92.49\% + 81.96\%}{2} = 87.23\%$. Using this data, the training graph, and the result, nothing points to overfitting or underfitting of the model. However, the validation data says that it was 1705 true positive (TP) and 139 false positive (FP) detection for a person. For a person falling, it was 81 TP and 19 FP. This data indicates a lack of falling person data in the dataset.

6.2 Dataset

The used dataset contains 3249 thermal images with two working labels, dropping the label car since there were so few images and the other labels since there were zero objects in the dataset. Alexey Bochkovskiy, one of the creators of the YOLOv4 models, writes on Github [2] that to improve the object detection performance, the dataset should contain at least 2000 different images for each class. When looking at the used dataset, it is possible to see that it has potential for improvement. Since the falling person class is so important, this class should contain even more pictures, possibly in the ten thousand range and not in the hundred range as it does now. Gathering and labeling so many pictures is a very time-consuming job and way out of the scope of this thesis. However, the used dataset shows good potential and is a good starting point to continue expanding the dataset.

6.3 Detection of a falling person

In the video used for testing, five people were falling. The system detected a fall in at least one frame in four of the five falls, and the last fall was probably not detected due to some corrupt frames. This result shows that the system would have raised the alarm correctly in all the tests and could have saved this person's life if the falls were accidental and they needed help.

6.3.1 Object detection

Since object detection is the backbone of all the detection methods, it is the most crucial aspect to use time and effort to improve. Improving the dataset and using better hardware and longer time to train on the dataset will improve the object detection. Better object detection will also

improve all the other methods, and the system will be more robust. Regardless, the object detection shows good potential for the system and works as proof that it is worth investing time and money in the project to make a system that can save countless lives.

6.3.2 Line detection

The line detection method for detecting a falling person works very well. As long as the object detection detects an object at the line check, it works 100% of the time, and raising the alarm can easily be done. The line check now works as long as an object crosses it. This can be changed to be a specific object. Combining the line check and Deep SORT makes it possible to look at the direction of an object crossing the line. Directions make it possible to remove false positives, such as when a person is climbing up the ladder and is not falling. Also, using the adaptive line, the system can learn and adapt to people's movement patterns over time. This gives a system that is automated and is robust to changes in movement patterns and locations.

6.3.3 Movement detection

The movement detection has massive potential if everything works correctly. It can be extended to look at the expected movement patterns and make an alarm if someone deviates too much from the expectations. This will also make the line detection obsolete as the movement detection will work better and without a defined strict line. The movement detection presented in this thesis looks at the absolute relative movement of objects. However, splitting the relative movement into X-direction movement and direction Y-movement may be favorable, as a fall will significantly change Y movement.

The result shows that the movement detection is unstable. The detection and tracker are not working well enough to gather the necessary history or a bad history of objects. The uncertainty makes it difficult to raise the alarm correctly. A possible solution is to combine the movement detection and object detection only to raise the alarm if the object moving fast is also a falling person or a person moving above a set threshold in only Y-direction. Doing this will also compliment the object detection making it more precise.

6.4 Deep SORT

The result from the Deep SORT algorithm is a bit difficult to validate due to a challenging video scenario with much occlusion. For example, one tricky part is to know if the ID switching was incorrect or due to the object reaching its max-age parameter. However, there are things to try to import the Deep SORT. The original Deep SORT uses a neural network trained to extract and recognize people in RGB images. A way to improve the Deep SORT in thesis would be to train a new network trained on thermal images. This would mean a new dataset for this purpose. How much this would improve Deep SORT is hard to tell due to the uncertainty in how well the network can get and recognize features in thermal images.

Deep SORT gives the system much valuable information that can help in making the system even more automated. For example, the information about regular movement patterns can be beneficial for making a system that can learn and adapt over time. This will lift the performance and the overall usability of the system. However, the high computational power needed to run Deep SORT and if the information gathered is wrong or lacking, it can be beneficial to drop Deep SORT to redirect the computational power elsewhere. For instance, for the object detection to gain a higher FPS. It can also be reasonable to look at other trackers that are not so dependent on features since there are fewer features in thermal images than in RGB images. The FPS reach in this thesis strongly contradicts the result achieved in Doan [13] and can point to some error in the implementation of the Deep SORT, or maybe the used Github [77] code has some faulty code. This discrepancy should be looked into because the system, as of now, can not run as a real-time system.

6.5 Trajectory analysis

When looking at the plot of all trajectories, it is easy to see where the majority of movement is. The majority of the movement goes along the pier, either from the left or right side. This will be the mean trajectory and can work to raise the alarm if an object is deviating too much.

Comparing a person falling and a non falling person shows a distinct difference in movement. If a person is walking, the Y value gets a very consistent value as the movement when walking is almost only in the X-direction. The X value steadily increases or decreases and depends on whether the object walks towards the camera or away. When a person falls, the X and Y value increases very fast. The rapid change in the values is similar in all four falling trajectories. This difference makes it easy to distinguish between a person falling and walking and can easily make for an alarm that looks for such movement.

A person climbing up a ladder also gives a very distinct trajectory movement. The movement is steady in X value, as the ladder is vertical. However, the Y value steadily decreases as the person moves upwards on the ladder. The movement is different both from a person falling and a person walking. Using all this information, it would be possible to make a system that can distinguish between all three relevant movements in this thesis. The movement can be split into allowed movement patterns if a person is walking or climbing and into patterns that will raise the alarm, such as falling or some unknown pattern.

A possible challenge is that the movement patterns may vary if the camera angle and position change. Due to this, the system needs unique adjustments at each location. Therefore, implementing such a system can be challenging, time-consuming, and expensive.

6.5.1 New YOLO model

When looking at the training result and other data such as the mAP on $\frac{84.34\% + 91.38\%}{2} = 87.86\%$ for the new model, it is nothing that directly points to overfitting or underfitting. The system should work as best as possible in new places without seeing the place before. Therefore, it seems not fair to use the model to look at object detection performance. However, the new model can be used to show a proof of concept on the trajectory analysis part. It even gives a more accurate trajectory as it detects the objects better.

Chapter 7

Conclusion and future work

7.1 Conclusion

This thesis shows that it is possible to develop a system that uses several methods to detect a person falling into the water. The methods use different approaches and can be used on their own or complement each other to make a more robust system. The object detection works with an mAP of 87.23% on the validation set. A system that combines object detection and the line check will work every time as long as an object is detected at the line crossing. The Deep SORT algorithm is not running in real-time and has a lot of improvement potential with gathering correct data. However, improving Deep SORT can give the system much helpful information that can make a system that can automatically adapt to changes in behavior or locations.

The trajectory analysis shows that it is easy to distinguish the movement pattern of a person falling, walking, or climbing. A system can interpret the analysis to look at typical movement patterns and make an alarm when someone deviates too much from a set path. As more trajectories are recorded and analyzed, such a system can also learn over time.

The overall result is that the system can detect a person falling into the water. However, the system needs further improvement and optimization, such as increasing the dataset size. Nevertheless, investing time and resources into this system could make it so that no accidents will go unseen and save countless lives.

7.2 Further work

Some improvements can be implemented to increase the system's reliability and automation. The following list shows suggestions for further development and improvement of the system.

- Improve the dataset, focusing on increasing the number of images containing a person falling
- Improve the speed of the system to make it real-time. Either by fixing or changing the tracker or dropping the tracker and focusing only on object detection
- Make the system more reliable to detect more people and reduce false positives. It can be done by using more powerful hardware, training a more complex detection model, or training a model longer and on a better dataset

Bibliography

[1] AlexeyAB. Yolo format, 07 2018. URL https://github.com/AlexeyAB/Yolo_mark/issues/60. accessed 11.03.22.

[2] AlexeyAB. How to improve object detection, 01 2022. URL https://github.com/AlexeyAB/darknet#how-to-improve-object-detection. accessed 24.05.22.

[3] Zebop Avalon. Zebop avalon, 09 2021. URL https://www.zebopavalon.com/. accessed 12.09.21.

[4] Nilesh Barla. The complete guide to object tracking, 05 2022. URL https://www.v7labs.com/blog/object-tracking-guide. accessed 22.05.22.

[5] Alex Bewley, ZongYuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. *CoRR*, abs/1602.00763, 2016. URL http://arxiv.org/abs/1602.00763.

[6] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020. URL https://arxiv.org/abs/2004.10934.

[7] Jason Brownlee. Overfitting and underfitting with machine learning algorithms, 08 2019. URL https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/. accessed 15.09.21.

[8] cdeterman. what is a 'layer' in a neural network, 02 2016. URL https://stackoverflow.com/a/35347548. accessed 27.10.21.

[9] Maxence Chaverot, Maxime Carré, Michel Jourlin, Abdelaziz Bensrhair, and Richard Grisel.

Object detection on thermal images: Performance of yolov4 trained on small datasets, 10 2021. URL https://www.esann.org/sites/default/files/proceedings/2021/ES2021-130.pdf. accessed 15.01.22.

[10] COCO. Coco dataset, 09 2021. URL https://cocodataset.org/#home. accessed 15.09.21.

[11] Anushka Dhiman. Object tracking using deepsort in tensorflow 2, 10 2020. URL https://medium.com/analytics-vidhya/object-tracking-using-deepsort-in-tensorflow-2-ec013a2eeb4f. accessed 06.03.22.

[12] diagrams. draw.io, 09 2021. URL https://www.diagrams.net/. accessed 12.09.21.

[13] Thanh-Nghi Doan and Minh-Tuyen Truong. Real-time vehicle detection and counting based on yolo and deepsort, 2020. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9287483.

[14] IBM Cloud Education. Machine learning, 07 2020. URL https://www.ibm.com/cloud/learn/machine-learning#toc-machine-le-K7VszOk6. accessed 08.10.21.

[15] Maël Fabien. Local features, detection, description and matching, 03 2019. URL https://maelfabien.github.io/computervision/cv_4/#. accessed 30.09.21.

[16] FLIR. Boson 640, 11 2021. URL https://www.flir.com/products/boson/?model=20640A018. accessed 03.11.21.

[17] FLIR. Flir a400/a700, 10 2021. URL https://www.flir.com/products/a400-a700-science-kits/?model=85902-0202. accessed 06.10.21.

[18] FLIR. Flir e96, 01 2022. URL https://www.flir.eu/products/e96/8. accessed 28.01.22.

[19] FLIR. Flir one gen 3, 05 2022. URL https://www.flir.com/products/flir-one-gen-3/?vertical=condition20monitoring&segment=solutions. accessed 16.05.21.

[20] FLIR. Build safer, smarter cars with thermal cameras, 05 2022. URL https://www.flir.com/oem/adas/. accessed 11.05.22.

[21] FLIR. Free teledyne flir thermal dataset for algorithm training, 01 2022. URL https://www.flir.com/oem/adas/adas-dataset-form/. accessed 15.01.22.

[22] FLIR. Flir bridge, 01 2022. URL https://www.flir.eu/products/bridge/?model=T131369. accessed 28.01.22.

[23] William Franklin. Kalman filter explained simply, 12 2020. URL https://thekalmanfilter.com/kalman-filter-explained-simply/. accessed 04.06.22.

[24] Arun Gandhi. Data augmentation how to use deep learning when you have limited data part 2, 05 2021. URL https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/. accessed 30.09.21.

[25] Luiz Gatelli, Gabriel Gosmann, Felipe Fitarelli, Guilherme Huth, Anderson A. Schwertner, Ricardo de Azambuja, and Valner J. Brusamarello. Counting, classifying and tracking vehicles routes at road intersections with yolov4 and deepsort, 2021. URL https://ieeexplore.ieee.org/document/9557244.

[26] git. git, 09 2021. URL https://git-scm.com/. accessed 12.09.21.

[27] Ekta Goel. Software framework vs library, 09 2020. URL https://www.geeksforgeeks.org/software-framework-vs-library/. accessed 09.09.21.

[28] Google. Descending into ml: Training and loss, 02 2020. URL https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss. accessed 25.09.21.

[29] Tom Harris. How cameras work, 09 2021. URL https://electronics.howstuffworks.com/camera.htm. accessed 02.09.21.

[30] HMKCODE. Backpropagation step by step, 11 2019. URL https://hmkcode.com/ai/backpropagation-step-by-step/. accessed 09.10.21.

[31] Tan Jie Hng, Eric Lim Weilie, Chong Shao Wei, and Sutthiphong Srigrarom. Relative velocity model to locate traffic accident with aerial cameras and yolov4, 2021. URL https://ieeexplore.ieee.org/document/9611963.

[32] Jonathan Hui. map (mean average precision) for object detection, 03 2018. URL https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173. accessed 25.09.21.

[33] IBM. What is computer vision?, 09 2021. URL https://www.ibm.com/topics/computer-vision. accessed 02.09.21.

[34] STEMMER IMAGING. Region of interest (roi), 09 2021. URL https://www.stemmer-imaging.com/en-pl/knowledge-base/region-of-interest-roi/. accessed 08.09.21.

[35] Intel. Intel® core™ i7-7700k processor, 01 2022. URL https://www.intel.com/content/www/us/en/products/sku/97129/intel-core-i77700k-processor-8m-cache-up-to-4-50-ghz/specifications.html?wapkw=7700k. accessed 21.01.2022.

[36] JavaTpoint. Programming language, 09 2021. URL https://www.javatpoint.com/programming-language. accessed 02.09.21.

[37] Ritesh Kanjee. Deepsort — deep learning applied to object tracking, 08 2020. URL https://medium.com/augmented-startups/deepsort-deep-learning-applied-to-object-tracking-924f59f99104. accessed 14.03.22.

[38] Ed Kochanek. Are we in the golden age of thermal imaging?, 05 2022. URL https://irinfo.org/10-01-2015-kochanek/. accessed 16.05.21.

[39] Simeon Kostadinov. What is deep transfer learning and why is it becoming so popular?, 11 2019. URL https://towardsdatascience.com/what-is-deep-transfer-learning-and-why-is-it-becoming-so-popular-91acdcc2717a. accessed 15.09.21.

[40] Nishant Kumar. Digital image processing basics, 07 2021. URL https://www.geeksforgeeks.org/digital-image-processing-basics/. accessed 29.09.21.

[41] Lynred. Infrared technology and thermal cameras: How they work, 09 2021. URL https://www.lynred.com/blog/infrared-technology-and-thermal-cameras-how-they-work. accessed 08.09.21.

[42] MathWorks. Edge detection methods for finding object boundaries in images, 09 2021. URL https://www.mathworks.com/discovery/edge-detection.html. accessed 30.09.21.

[43] Matplotlib. Matplotlib: Visualization with python, 03 2022. URL https://matplotlib.org/. accessed 04.03.22.

[44] Maximinusjoshus. Understanding the concept of channels in an image, 04 2021. URL https://medium.com/featurepreneur/understanding-the-concept-of-channels-in-an-image-6d59d4dafaa9. accessed 29.09.21.

[45] Nick McCullum. Deep learning neural networks explained in plain english, 06 2020. URL https://www.freecodecamp.org/news/deep-learning-neural-networks-explained-in-plain-english/. accessed 27.10.21.

[46] Vidushi Meel. What is object tracking? – an introduction, 03 2022. URL https://viso.ai/deep-learning/object-tracking/. accessed 22.03.22.

[47] Microsoft. Visual studio code, 02 2022. URL https://code.visualstudio.com/. accessed 02.02.22.

[48] Vanessa Ndonhong, Anqi Bao, and Olivier Germain. Wellbore schematics to structured data using artificial intelligence tools, 04 2019. URL https://www.researchgate.net/publication/308320592_Fast_Single_Shot_Detection_and_Pose_Estimation. accessed 12.10.21.

[49] NRK. Henter opp døde mennesker fra nidelva hvert eneste år, 11 2019. URL https://www.nrk.no/trondelag/henter-opp-dode-mennesker-fra-nidelva-hvert-eneste-ar-1.13911303. accessed 02.09.21.

[50] NumPy. What is numpy?, 06 2021. URL https://numpy.org/doc/stable/user/whatisnumpy.html. accessed 12.09.21.

[51] Nvidia. 1080ti, 09 2021. URL https://www.nvidia.com/en-gb/geforce/graphics-cards/geforce-gtx-1080-ti/specifications/. accessed 11.09.21.

[52] Nvidia. Deepstream sdk, 01 2022. URL https://developer.nvidia.com/deepstream-sdk. accessed 28.01.22.

[53] Nvidia. Jetson nano developer kit, 01 2022. URL https://developer.nvidia.com/embedded/jetson-nano-developer-kit. accessed 21.01.2022.

[54] opencv. opencv, 09 2021. URL https://opencv.org/. accessed 12.09.21.

[55] Overleaf. About us, 09 2021. URL https://www.overleaf.com/about. accessed 12.09.21.

[56] Pandas. Pandas, 03 2022. URL https://pandas.pydata.org/. accessed 04.03.22.

[57] Addie Ira Borja Parico and Tofael Ahamed. Real time pear fruit detection and counting using yolov4 models and deep sort. *Sensors*, 21(14), 2021. ISSN 1424-8220. doi: 10.3390/s21144803. URL https://www.mdpi.com/1424-8220/21/14/4803.

[58] Kristian Fjelde Pedersen, Anders Vatland, Christian Peter Bech Aschehoug, and Rune Nordmo. Ttk4115: Linear system theory, 07 2021. URL https://www.wikipendium.no/TTK4115_Linear_System_Theory#kalman-filtering. accessed 04.06.22.

[59] Laura Pennington. Using the hungarian algorithm to solve assignment problems, 11 2021. URL https://study.com/academy/lesson/using-the-hungarian-algorithm-to-solve-assignment-problems.html. accessed 04.06.22.

[60] Selva Prabhakaran. Mahalanobis distance – understanding the math with examples (python), 04 2019. URL https://www.machinelearningplus.com/statistics/mahalanobis-distance/. accessed 04.06.22.

[61] Narinder Singh Punn, Sanjay Kumar Sonbhadra, and Sonali Agarwal. Monitoring COVID-19 social distancing with person detection and tracking via fine-tuned YOLO v3 and deepsort techniques. *CoRR*, abs/2005.01385, 2020. URL https://arxiv.org/abs/2005.01385.

[62] Python. Python, 09 2021. URL https://www.python.org/about/. accessed 02.09.21.

[63] Joseph Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016. accessed 12.09.21.

[64] Redningsselskapets. Redningsselskapets drukningsstatistikk, 05 2022. URL https://rs.no/drukning/. accessed 19.05.21.

[65] riptutorial. Activation functions, 10 2021. URL https://riptutorial.com/machine-learning/example/31624/activation-functions. accessed 09.10.21.

[66] Sumit Saha. A comprehensive guide to convolutional neural networks, 12 2018. URL

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. accessed 09.10.21.

[67] Sergio Saponara, Abdussalam Elhanashi, and Zheng Qinghe. Developing a real-time social distancing detection system based on yolov4-tiny and bird-eye view for covid-19. *Journal of Real-Time Image Processing*, pages 1–13, 02 2022. doi: 10.1007/s11554-022-01203-5. accessed 12.03.22.

[68] ScienceDirect. Feature extraction, 09 2021. URL https://www.sciencedirect.com/topics/engineering/feature-extraction. accessed 30.09.21.

[69] Seaborn. seaborn: statistical data visualization, 03 2022. URL https://seaborn.pydata.org/. accessed 24.03.22.

[70] Wu. Shaoji. Approach to auto-recognition of human trajectory in squares using machine learning-based methods - an application of the yolo-v3 and the deepsort algorithm, 2021. URL http://papers.cumincad.org/data/works/att/ecaade2021_021.pdf. accessed 23.03.22.

[71] Aditya Singh. Top 5 object tracking methods, 11 2021. URL https://medium.com/augmented-startups/top-5-object-tracking-methods-92f1643f8435. accessed 22.03.22.

[72] Petru Soviany and Radu Tudor Ionescu. Optimizing the trade-off between single-stage and two-stage object detectors using image difficulty prediction, 08 2018. URL http://arxiv.org/abs/1803.08707. accessed 12.10.21.

[73] Petru Soviany and Radu Tudor Ionescu. Optimizing the trade-off between single-stage and two-stage object detectors using image difficulty prediction, 03 2022. URL https://arxiv.org/abs/1803.08707. accessed 04.03.22.

[74] Suman. Artificial intelligence, 10 2019. URL https://ai.stackexchange.com/questions/15859/is-machine-learning-required-for-deep-learning. accessed 08.10.21.

[75] Kristin Sørdal. Derfor drukner flest eldre, 07 2021. URL https://www.vi.no/helse/derfor-drukner-flest-eldre/74019234. accessed 19.05.21.

[76] techzizou. Train a custom yolov4 object detector on windows, 08 2021. URL https://techzizou.com/train-a-custom-yolov4-object-detector-on-windows/. accessed 06.02.21.

[77] theAIGuysCode. yolov4-deepsort, 08 2021. URL https://github.com/theAIGuysCode/yolov4-deepsort. accessed 31.03.22.

[78] Tzutalin. Labelimg, 07 2021. URL https://github.com/tzutalin/labelImg. accessed 12.09.21.

[79] LYNRED USA. Visible vs. thermal detection: Advantages and disadvantages, 02 2022. URL www.lynred-usa.com/homepage/about-us/blog/visible-vs-thermal-detection-advantages-and-disadvantages.html. accessed 15.02.22.

[80] videolan. Vlc features, 09 2021. URL https://www.videolan.org/vlc/features.html. accessed 12.09.21.

[81] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. *CoRR*, abs/1703.07402, 2017. URL http://arxiv.org/abs/1703.07402.

[82] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric, 03 2017. URL https://github.com/nwojke/deep_sort. accessed 04.03.22.

Appendices

Appendix A

Source code

https://github.com/dr0nn1/masterThesis

A.1 DeepStream

https://github.com/dr0nn1/masterThesis/tree/main/DeepStream

A.2 deepSort

https://github.com/dr0nn1/masterThesis/blob/main/deepSORT/object_tracker.py

A.3 Adaptive line and line intersection

https://github.com/dr0nn1/masterThesis/tree/main/deepSORT/tools

A.4 Data analyse

https://github.com/dr0nn1/masterThesis/blob/main/deepSORT/csvReader.py

Appendix B

Data analyse pictures

All the plots that were made but not used in the report, due to not being relevant or good enough.

All the source code for the plots is in appendix A.4

Figure B.1 shows the mean trajectory calculated from the data.

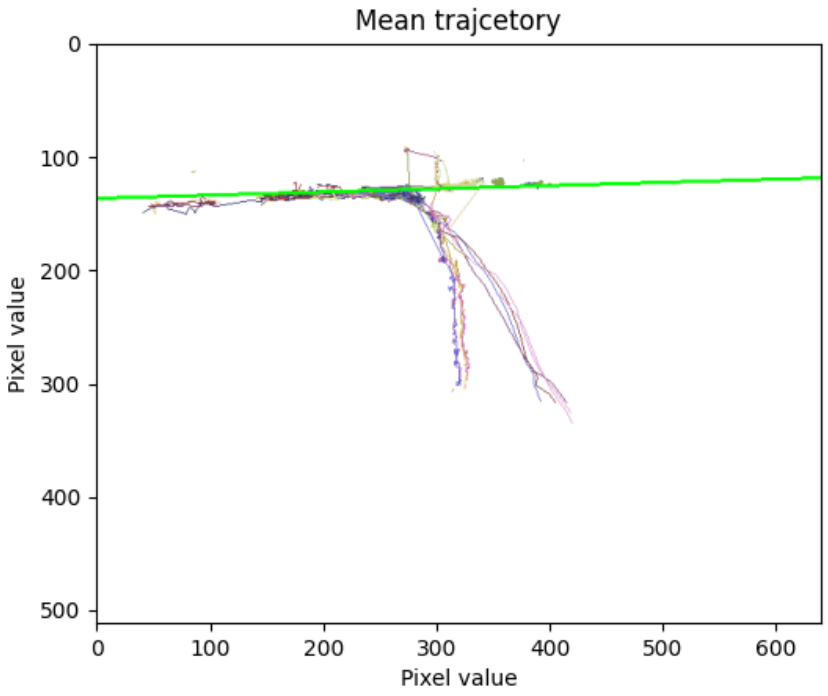


Figure B.1: Mean trajectory

Figure B.2 shows the heat map of the the movement. The brighter the more (X,Y) coordinates are in that spot.

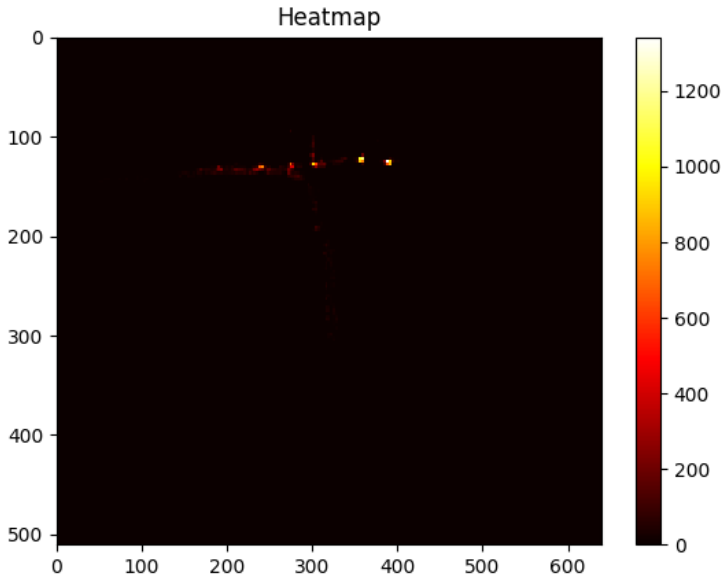


Figure B.2: Heat map

Figure B.3 shows number of times a X and Y coordinate is present in the data. Left side is X and right side is Y.

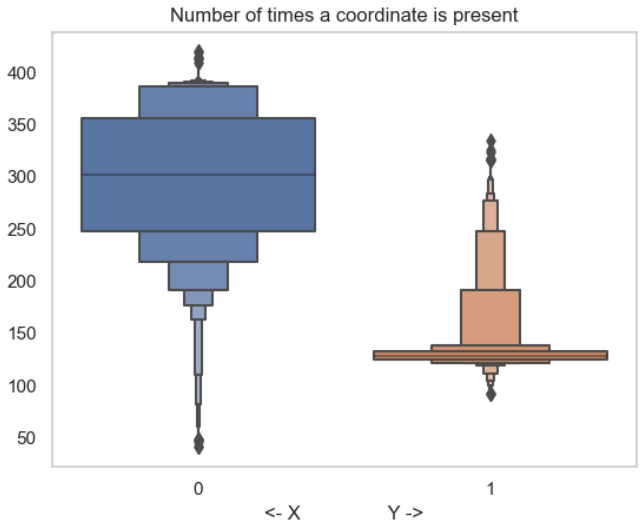


Figure B.3: Number of times a coordinate is present

Figure B.4 shows a heat map of distance between two points and their location in the image. The brighter the bigger distance.

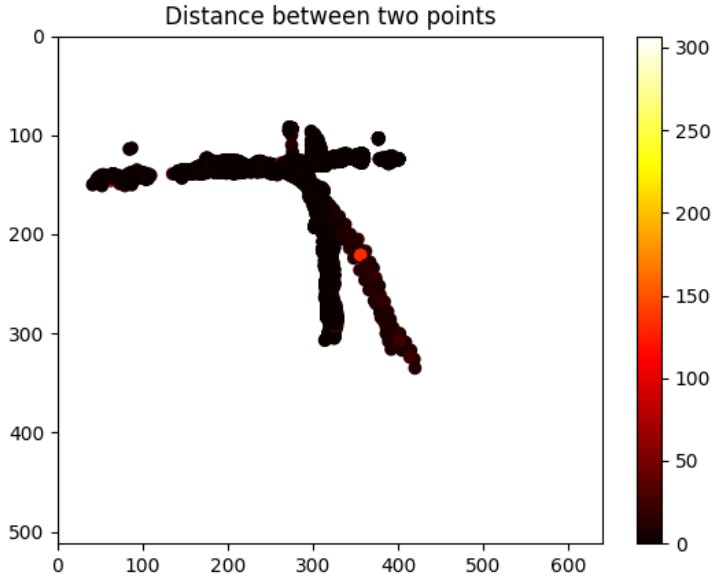


Figure B.4: Distance between points heat map

Figure B.5 shows number of times a distance between two points occur.

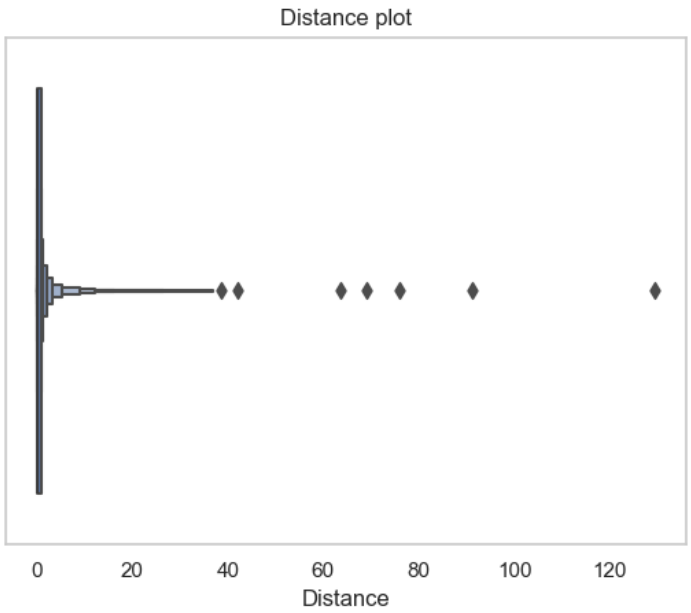


Figure B.5: Number of times a distance occur

Appendix C

Preproject

NTNU

Kunnskap for en bedre verden

Detection of a person in water from thermal images

Petter Drønnen

TTK4551

Faculty of Information Technology and Electronics

Norwegian University of Science and Technology

Norway

December 20, 2021

Supervisor 1: Annette Stahl

Pages / Appendix

53 / 0

Preface

The intriguing part of this project was to build a system from the ground up that both could be used in a commercial product and also help prevent drownings. If the system is used correct in the future it can help save lives, and minimize unnecessary accidents of people drowning.

This report was written on Gløshaugen campus - NTNU in Trondheim for NTNU in the subject TTK4551 7.5 SP. The project started late august 2021 and finished in late December 2021. The project is an introduction to the master thesis that will be written next semester. The report is written by a student taking the two year master course Cybernetics and Robotics.

Trondheim, December 20, 2021

Petter Drønnen

Acknowledgement

I would like to thank all the contributors to this project, and especially I would like to thank:

- My supervisor Annette Stahl at NTNU for guidance throughout the project
- All the great people working at Zebop Avalon for giving me the opportunity, guidance and hardware.
- Family and friends who have supported me throughout this period.
- Fellow students for good discussions about the project.

People working with and at Zebop Avalon posing for the thermal camera on Solsiden

Summary

In this thesis a new dataset for thermal images is presented containing person, person falling in the water and person in the water. The dataset is extracted from 13 videos and contains a total of 2265 images. From this several models were trained, and compared to both each other and their untrained version. The frameworks used in this paper were Tensorflow and Darknet. The models compared were Faster-RCNN and Single Shot Detector from Tensorflow, and YOLO and YOLO tiny from Darknet.

The results proved that a model trained on thermal images performed better than a model only trained on RGB images. However the training also introduced more errors. This can be a indication of overfitting. The overall best performing model was the YOLO model. The YOLO tiny model achieved 100% FPS boost compared to the YOLO model without to much compromising the performance, and can therefore be a good choice if there are hardware limitations.

With this mention there are still many improvements to be made such as improving the dataset.

Contents

Abbreviations

FOV Field of view

ML Machine learning

AI Artificial intelligence

CV Computer vision

IR Infrared

CNN Convolutional Neural Network

IoU Intersection over union

TF TensoFlow machine learning software from google

YOLO Your Only Look Once, state-of-the-art real-time object detection.

SSD Single Shot Detector

GPU Graphics processing unit

CPU Central Processing Unit

FPS Frames per second

mAP mean average precision

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem description

This study aims to use thermal cameras with different machine learning methods to detect a person falling in to the water. This will later be used to automatically tell the rescue service about an incident and hopefully reduce the number of drowning accidents. The figure 1.1 shows a person falling in the water that should be detected by the system.

Figure 1.1: A person falling into the water

A thermal camera was chosen due to its performance compared to normal cameras when it comes to environments with low visibility or poor lightning. This can be particular important when it comes to potentially hazardous remote areas such as a harbour at night.

1.2 Motivation

Each year in Norway about 100 people drowns under different circumstances [59], and each year one or more of these people drowns in Trondheim [38]. Through Trondheim, as seen in figure 1.2, a river called Nidelva runs. The arrows shows the direction the river flows. This river have strong under water currents, and it's here most of these accidents happens in Trondheim. So why not try to use technology to save some of these people?

Figure 1.2: Overview

1.3 Background

In this video [4] the fire and rescue service in Trondheim shows how difficult it's to swim in Nidelva due to the currents. The Norwegian Society for Sea Rescue [46] also want the government to make the same zero death vision for drowning accidents as it has for deaths in the traffic. The figure 1.3 shows the camera placement and field of view (FOV). The thermal camera is placed on "Blomsterbrua", a bridge in Trondheim, and is looking towards Solsiden.

Figure 1.3: Camera placement

1.4 Aim and objectives

The aim for this thesis is to lay a good foundation for the following master thesis. Therefore several milestones should be achieved. This will be used as arguments for the choices taken in the master thesis. The following milestones should be achieved:

- Create a new original dataset for thermal images
- Develop a system to detected the following classes: "people", "people falling" and "people in the water"
- Testing of different machine learning models and framework for detecting these behaviors

1.5 Approach and contributions

In this project the approach will be as the following. First the data will be collected. This collecting of data will happen throughout the project when it seems necessary. This can be different locations, different scenarios etc. After the first data is collected it needs to be preprocessed and labeled. When this is ready several models will be tested against each other to see which model fits best the requirements. This will lay the ground work for which models that should proceed to the master thesis.

The main contributions of this paper are a) comparing several different convolutional neural networks pretrained on RGB images and models trained on thermal images. Also the different frameworks are compared. b) a new original dataset for thermal images with persons, persons falling and persons in the water labeled.

1.6 Structure of the Report

The rest of the report is structured as follows:

Chapter 2 - Previous work: Gives an introduction to work done by others in this field

Chapter 3 - Theory: Gives an introduction to the theoretical background needed

Chapter 4 - Method: Methods used to perform the different test are presented

Chapter 5 - Result: The different results are presented

Chapter 6 - Discussion: A discussion on the different results are presented

Chapter 7 - Conclusions: The report is concluded

Chapter 2

Previous work

2.1 Object detection

The state of object detection has improved a lot in the last years with the help of methods such as convolutional neural network. Much time and research are still invested in object detection, and today's performance with model architecture such as Faster R-CNN, YOLO, EfficientDet and SSD are historical good when both comparing them on speed and accuracy. Object detection on thermal images is a sub branch of the normal way, and has not so much time and research put into it. However in the recent years several promising area of application has been researched such as surveillance and the use in autonomous cars.

When comparing models used in object detection the two most common methods are speed on the detection and mean average precision (mAP) often on the COCO dataset. mAP is just a measure for how good a model is, the higher the better from 0-100, and the COCO dataset [9] is a dataset with over 200 000 labeled images with 80 object categories.

In Zhang [63] the authors used Multi-block Local Binary Pattern to achieve real-time and robust objects classification in diverse camera viewing angles. This report was written before deep learning become widespread. Local Binary Pattern is a simple method that goes through each pixel in an image and looks at its neighbors with a threshold and consider the result as a binary number. The result in the paper was real-time performance with an accuracy of around 80% on a test dataset.

In Sreenu [53] the authors present a review of how others have used video surveillance to do for instance abnormal behavior detection or crowd analysis. The report has its main emphasis on deep learning methods, and convolutional neural networks was the most used method such as the YOLO algorithm.

In Bochkovskiy [3] the authors took the YOLOv3 model and improved it in to the YOLOv4 model. The result was 65.7% mAP with a intersection over union (IoU) at 50% at a speed of ~65 FPS with a Tesla V100 GPU on the COCO dataset. This result is a 10% mAP improvement over the YOLOv3 with 12% higher FPS.

In Tan [55] the authors developed several models called EfficientDet. The models ranges from EfficientDet-D0 to EfficientDet-D7 where D0 is smallest and D7 is the biggest. The D7 achieved state of the art performance with a mAP of 52.2% on the COCO dataset. The models also often performed better and were smaller than models in the same size region.

In Brunner [6] the authors used an unmanned ground vehicle that was equipped with both a normal camera and a thermal camera. The main goal was to perform visual-SLAM. The best method presented in this paper was combining both normal and thermal before running state-of-the-art visual-SLAM on the data. This resulted in better performance especially in low-visibility conditions.

2.2 Thermal camera object detection

Thermal cameras can have several benefits compared to normal cameras. The main benefit is its improved visibility at nights due to looking at heat radiation rather than light. However due to this difference some problems can occur when trying to detect persons. In the following section some papers that uses thermal cameras for different proposes are listed. In figure 2.1 and 2.2 a side by side comparison of normal and thermal in the dark can be seen.

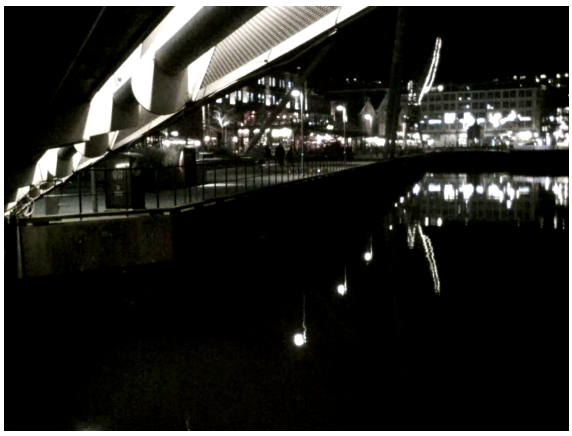


Figure 2.1: Solsiden with a normal camera

Figure 2.2: Solsiden in thermal

In Munir [36] the authors presented a method for fusing normal images and thermal images to transfer the low-level features from normal images to thermal. The method uses a generative adversarial network to fuse the images together, and the authors saw an improvement over detecting only on the thermal images.

In Rodin [48] the authors used a fixed wing Unmanned Aerial Systems equipped with a thermal camera to conduct Search And Rescue at the ocean. A Gaussian Mixture Model (GMM) was first used on the images in order to discriminate foreground objects from the background and then given to a CNN to detect either boats, pallets, human or buoys. The final accuracy achieved on the test dataset was 92.5%.

In Schedl [50] the authors used a drone with a thermal camera to try find persons in the forest when performing search and rescue operations. The authors did see an improvement when combining multi-perspective recordings before classification compared to single images. The CNN used was YOLO and achieved 96% accuracy on their test dataset.

In Ippalapally [24] the authors trained two different MobileNet models, MobileNet V1 and MobileNet V2, and tested it on thermal images. The MobileNet architecture is one of the smallest and fastest. The result the authors got on their dataset was an average precision of 33.2 and 35.1 with IoU at 0.50 for V1 and V2 respectively.

In Kristo [28] the authors tested how good several models could work with detecting persons in thermal images. They also tested under different circumstances like fog and bad weather. In one of the tests several models were compared where faster R-CNN scored the highest with 98,86% mAP on their dataset with 7,097 frames per second (FPS). YOLOv3 scored about one percent lower at 97,93% mAP, but had 27,472 FPS.

Chapter 3

Theory

3.1 Computer vision

Computer vision (CV) deals with how a computer can understand digital images and videos without the help of humans. This is a very complex problem and much time and effort is put into this field each year. Some examples of what CV can be used for is object detection, video tracking and image restoration [22].

3.1.1 Camera

In CV a camera is often used as the input as either an image or video. Camera can also have different hardware that will result different frame rates, resolution and shutter speed with more. The light in the shoot, surrounding medium and space also effect the camera. Therefore all this effects needs to be accounted for to see what field the camera is best suited to work in [19].

3.1.2 Thermal camera

Instead of using visible light like conventional cameras the thermal cameras uses infrared (IR) radiation. These cameras normally works in the wavelength of 1 micrometre (1μ) to 16μ. This means that the camera see heat instead of light. Therefore the cameras have a large set of applications ranging from military use to surveillance to data center monitoring. This is because the

thermal cameras often works better in some conditions like in the dark [31]. There are several ways too visualise thermal images, and two of these can be seen in figures 3.1 and 3.2 below.

Figure 3.1: Thermal with colors

Figure 3.2: Thermal in grayscale

3.1.3 Digital image

A computer normally represent a two dimension image with an two dimension array that contains all the pixels in a picture. For instance a picture that has a height of 4 pixel and width of 4 pixel, for a total of 16 pixel, have their own spot in the array as shown in table 3.1. Each spot can normally store 8 bits. 8 bits gives $2^8 = 256$, meaning a value from zero to 255 can be stored [33].

[[p(0,0)	p(0,1)	p(0,2)	p(0,3)]
[p(1,0)	p(1,1)	p(1,2)	p(1,3)]
[p(2,0)	p(2,1)	p(2,2)	p(2,3)]
[p(3,0)	p(3,1)	p(3,2)	p(3,3)]]

Table 3.1: Pixel placement in picture

Usually images are either stored in grayscale format or red green blue (RGB) format. With grayscale each pixel gets its own value from zero to 255 that contain the intensity of light in that pixel. This means a value of zero is black, 127 is gray and 255 is white [33]. An example of a 4x4 image can be seen in table 3.2

[[0	0	0	0]
[100	100	100	100]
[200	200	200	200]
[255	255	255	255]]

Table 3.2: Grayscale array

The RGB format normally contains three channels were each channel representing a color as seen in table 3.3, this is a 4x4 image. As before the value in each element is usually from zero to 255. The first element in a pixel represent the intensity of red. The second element represent green and the last blue. So an pixel with value [0 0 0] = black, [255 0 0] = red, [0 255 0] = green, [0 0 255] = blue, [255 255 255] = white and any other combination gives the rest of the possible colors for a total of 16,777,216 different colors [33].

[[[0 0 0]	[0 0 0]	[0 0 0]	[0 0 0]]
[[100 100 100]	[100 100 100]	[100 100 100]	[100 100 100]]
[[200 200 200]	[200 200 200]	[200 200 200]	[200 200 200]]
[[255 255 255]	[255 255 255]	[255 255 255]	[255 255 255]]]

Table 3.3: RGB array

3.1.4 Image processing

Image processing is to use a algorithm on a image to do some work on the image. This can be easy task such as scaling or changing a RGB picture to grayscale or vice-versa to more advance such as image enhancement and image restoration to the most advance as feature extraction with more [29].

3.1.5 Feature extraction

Feature extraction is to take an image through an algorithm that highlight some interesting parts or traits in the image. The reason that this is used is to minimize redundant data and speed up training and the time it takes to make a predictions. There are many methods for extracting different features, and all have there own set of uses. Some of the methods used in feature extraction and object recognition are edge- and corner detection [51].

- Edge detection is to find the edges in an image. This is done by detecting discontinuities in brightness in the image, and there are several methods for achieving this. In figure 3.3 an example of the Canny method can be seen [32].

Figure 3.3: Edge detection example [32]

- Corner detection, as in edge detection, has several methods that can be used. However the main idea is to take a small windows over every pixel and see if there are a change in the pixel values in all directions [12]. The main idea is visualised in figure 3.4.

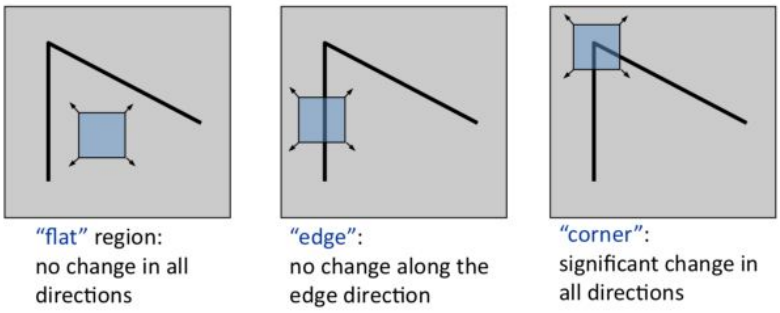


Figure 3.4: Corner detection example [12]

3.1.6 Region of interest

A region of interest (ROI) in a image is a part of the image that is of extra interest. This usually means to cut away some part of the image to either reduce the size, change aspect ratio or highlight some parts more. Therefore the bandwidth required for each image can be reduced and a higher frame rate can often be achieved [23].

3.2 Machine learning

Machine learning (ML) is to use algorithms on data to learn more and more over time. This can be several things such as computer vision and recognizing the different between a person and a dog. It can be complex pattern recognition to help solve cancer and so on [11]. The difference between artificial intelligence (AI), ML and deep learning (DL) can be seen in the figure 3.5.

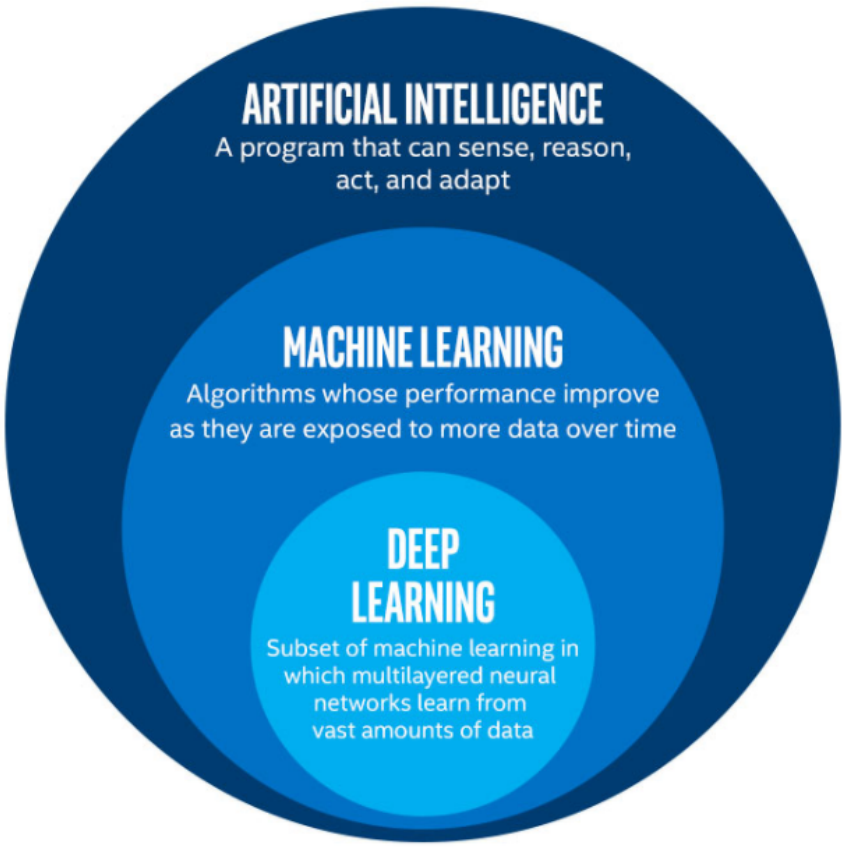


Figure 3.5: Difference between AI, ML and DL [54]

3.2.1 Training

Supervised and unsupervised are two methods used for learning in ML. Supervised learning is when the dataset is labeled and fed into the algorithms to train. The algorithm then adjust its weights until the model perform appropriately. This is often used in problems such as object detection and classification. Unsupervised learning is when a model is given a new dataset to analyze. When analysing the model finds patterns or data grouping without interaction from a human [11].

3.2.2 Neural networks

Using neural network is a way to perform ML. The way neural networks are build up is often compared with the human brain. An easy example of a neural network can be seen in the figure 3.6 below. A neural network consist of an input layer, one or more hidden layers and a output layer. Each of this layers are connected to each other with the help of neurons [8]. In the following sections the building blocks will be explained.

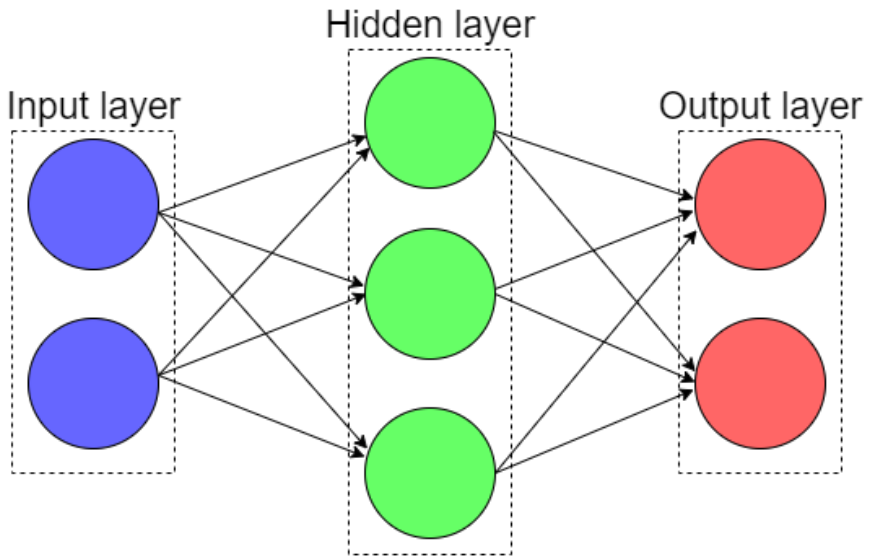


Figure 3.6: Neural network

3.2.3 Input layer

The input layer is where the raw data is first entered. This can for instance be a neuron for each pixel value in a picture. The data is often preprocessed to fit the total number of neurons in the input layer. This can be to scale a picture up or down in size. When this is done the network is ready to send the data to the hidden layer [8].

3.2.4 Hidden layer

It's in the hidden layer the work is done in a neural network. Here each layer is trying to learn something about the data by minimizing a cost function. The easiest way to think about this is that for example the first hidden layer learn edge detection, the next corner detection and so on. This is not exactly what is going on, but it's the main idea. The results is therefore a complex problem broken down to smaller problems [8].

3.2.5 Output layer

The output layer returns a value for each of the neurons that is in the output layer. For instance if the network displayed in figure 3.6 is a image classification between dogs and cats, the output layer will then output the confident about each class. The output can be [0.97 0.03], the model is then 97% sure the image is a dog and 3% sure it's a cat [8].

3.2.6 Neurons

Each neuron, the colored circles in figure 3.6, are connected to each other as shown in the same figure. Each of this connection contains a weight that is a positive or negative number. When a value is sent from a layer to the next layer the value is multiplied with its weight. The neuron in the next layer then sum all the inputs from all the neurons in the previous layer.

The value from this sum is then sent to an activation function, some of the most common are shown in figure 3.7. The connection between neurons can also contain an bias. This is just a number that shift the activation function either left or right. The activation function is used to keep the output of a neuron in a range normally between zero and one. This process occur in every neutron until an output is given in the output layer [34].

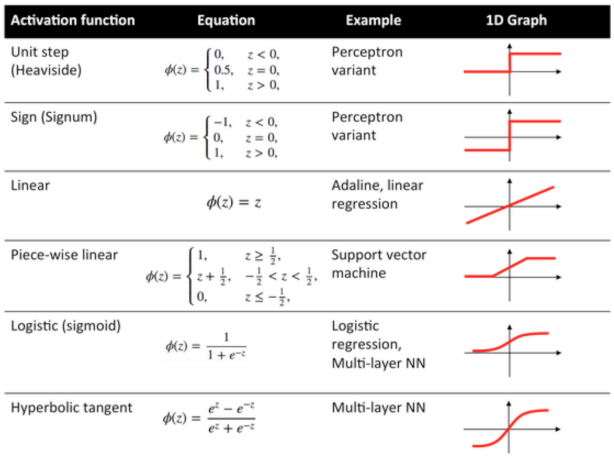


Figure 3.7: Activation functions [47]

3.2.7 Backpropagation

When training a model the weights and biases needs to be adjusted, and this processes is called backpropagation. First the network is given an input and goes through the procedure explained in the previous paragraph. The error at every point is calculated and saved often using:

$$Error = \frac{1}{2} * (prediction - actual)^2 \tag{3.1}$$

Then the backpropagation processes starts using gradient descent to update the weights and biases. This processes starts at the output layer and goes backwards towards the input layer, and repeats until a desired performance level is achieved [20]. A normal way to update the weight and bias at each neuron is using this formula:

$$\text{New weight} = \text{old weight} - (\frac{\partial \text{Error}}{\partial \text{old weight}}) \tag{3.2}$$

3.2.8 Convolutional neural network

Convolutional neural network (CNN) is a deep learning algorithm often used for image classification. The CNN can take a image as a input and learn it features with enough training without the need of human help [49]. The figure 3.8 shows an example of the structure of a CNN. The different layers will be explained in the next sections.

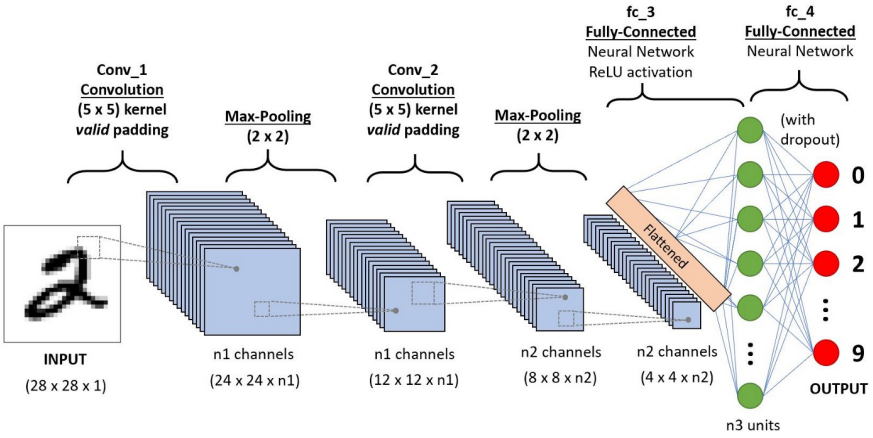


Figure 3.8: CNN for classifying handwritten digits [49]

3.2.9 Convolution layer

The convolution layer job is to apply a filter on the input. This filter extract features such as edges and corners. The later the layers are in the network the more high level features are extracted. Therefore adding convolution layer several places in the network makes the model more robust. The two methods for extracting features are same padding and valid padding. With same padding the input is either increased in size or the same. With valid padding the input data is reduced [49].

3.2.10 Pooling Layer

The pooling layer is used to reduce the size of the data and also extracting dominant features. The two methods used are max pooling and average pooling. The max pooling returns the maximum value from the portion of the image covered by the filter. Average returns the average of these values. The most common is max pooling since this also performs a noise reduction [49].

In the figure 3.9 the methods are shown. The max pooling takes a number of values, here used four values, and returns the maximum. Average does the same, but returns the average of these four values instead.

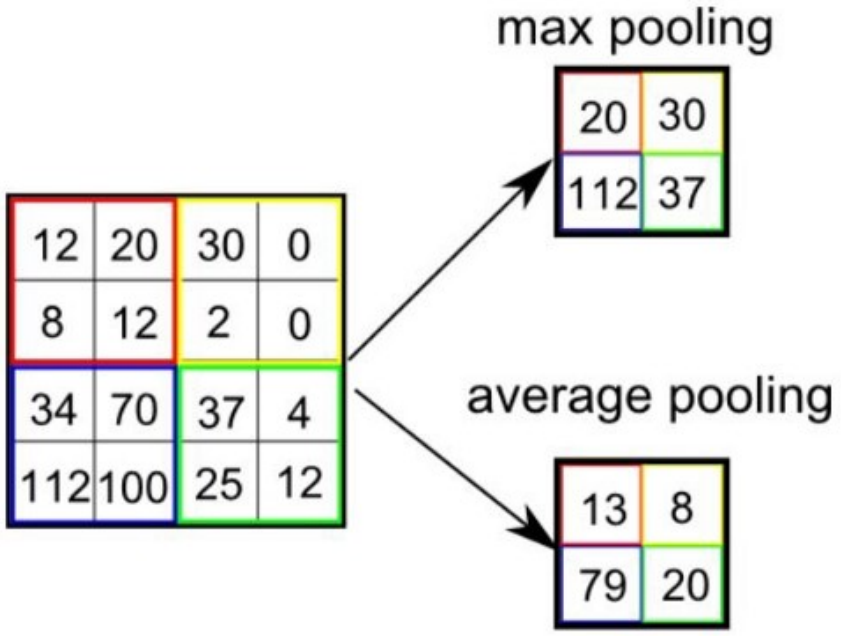


Figure 3.9: Max and average pooling [49]

3.2.11 Fully Connected Layer

The last layer in a CNN is the fully connected layer. This is often a neural network and works as described in the chapter 3.2.2. When training, the layer adjust its weights and biases with the help of backpropagation. The final output is also given here [49]. As with all ML there are several ways to solve the problem. Some popular architecture are ResNet, AlexNet and YOLO [49].

3.2.12 Single- and two-stage detectors

When talking about state-of-the-art object detectors single- and two-stage detectors are most often mention. The two-stage first uses a Region Proposal Network to extract region of interest from the image. This regions are then sent further in the network where object detection is done. The benefits of a two-stage detector it its accuracy.

A single-stage detector does all this in one stage. This increases speed drastically, but often reduces accuracy [52]. In the figure 3.10 a visual representation of the methods can be seen.

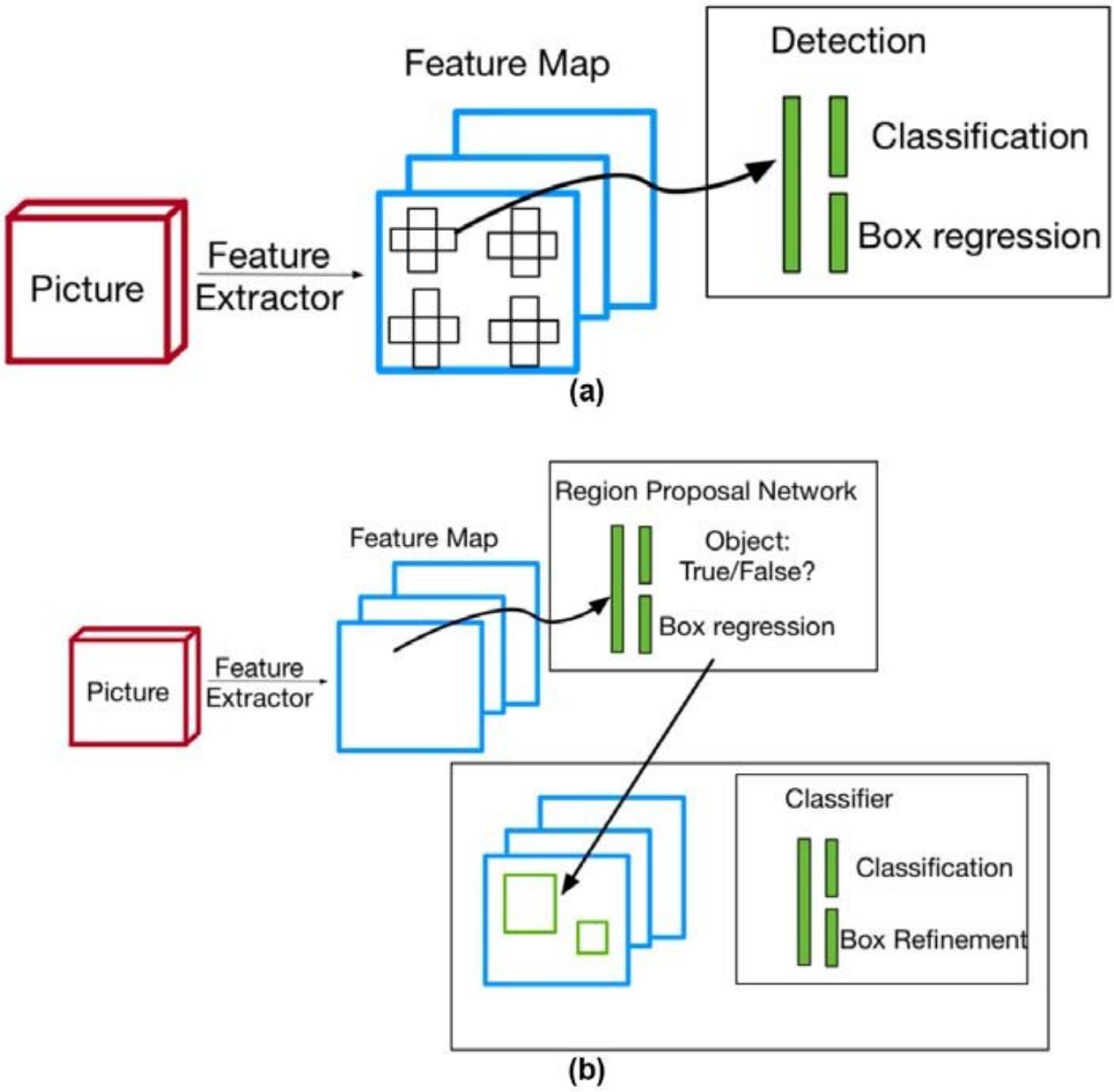


Figure 3.10: Difference in single- (a) and two-stage (b) detector [37]

3.2.13 Data augmentation

Data augmentation is to create new data by either using the existing data or create new synthetic data. Synthetic data is data created artificial. When using existing data it can be techniques such as flipping a picture horizontal or vertical, rotating the picture, change the colors,

different scales with more. This can be done either offline or online. Offline is when data augmentation happens before training a model, and is most used with a smaller dataset since this will increase the size of the dataset. For instance if each picture is rotated 90° the dataset will double in size, and can than fast run in to space limitations when multiple augmentations are used. Online augmentation is when the augmentation happens during training, and is often done by the framework. This is often used for larger dataset since no new data is created [14].

3.2.14 Transfer learning

Transfer learning is to take a pretrained model that is trained on a dataset, for instance the COCO dataset [9] that has over 200 000 labeled images with objects like boats, persons, dogs with many more, and take this knowledge on a new dataset like thermal images. There is several benefits from this. The pretrained models are often trained on a very large dataset with very good hardware. This means a good starting point for a new model can be achieved easy. It often also means that the new dataset can be smaller and achieve a better result [27].

3.2.15 Evaluation

When evaluating a model it's important with a measurement that is equal for all models. In the following sections the most standard methods will be explained. For understanding these methods some abbreviation needs explanation:

- **True positive (TP)**: Both the real observation and prediction is positive.
- **True negative (TN)**: Both the real observation and prediction is negative.
- **False positive (FP)**: Real observation is negative, but prediction is positive.
- **False negative (FN)**: Real observation is positive, but prediction is negative.

3.2.16 Precision

Precision is the percentage of the predictions that are correct [21]:

$$Precision = \frac{TP}{TP + FP} \tag{3.3}$$

3.2.17 Recall

Recall calculates how good the model is to find all the positives [21]:

$$Recall = \frac{TP}{TP + FN} \tag{3.4}$$

3.2.18 Intersection over union

Intersection over union (IoU) is how much the prediction overlaps with the ground truth.

$$IoU = \frac{\text{Area of overlap}}{\text{Area of union}} \tag{3.5}$$

So when the IoU threshold is 0.5 the overlap needs to be over 50% to be classified as a true positive, and if the threshold is one it need to be a perfect overlap [21]. An example of IoU is shown in the figure 3.11 below.

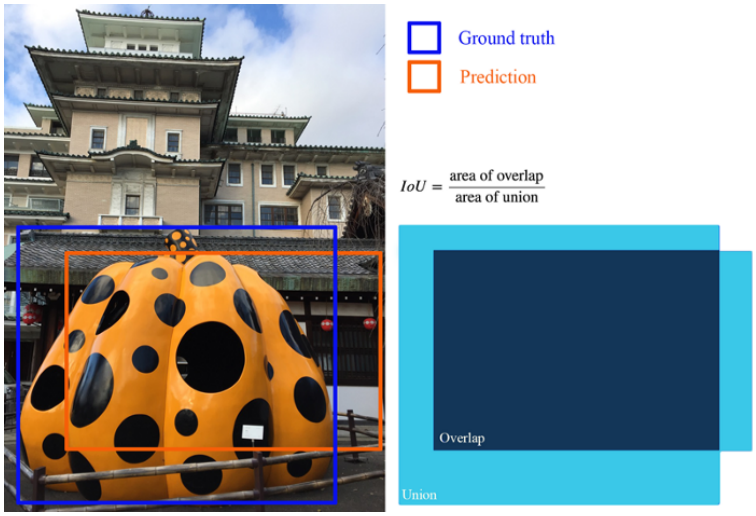


Figure 3.11: Explanation of IoU [21]

3.2.19 Mean average precision

Average precision (AP) or mean average precision (mAP) are often synonymous with each other, and is calculated by finding the area under the Precision-recall curve. Sometimes the AP is calculated for each class, and then mAP is calculated by taking the average of the APs [21]. Below in the figure 3.12 an example of how a precision-recall curve can look is shown.

$$AP = \int_0^1 p(r)\,dr \tag{3.6}$$

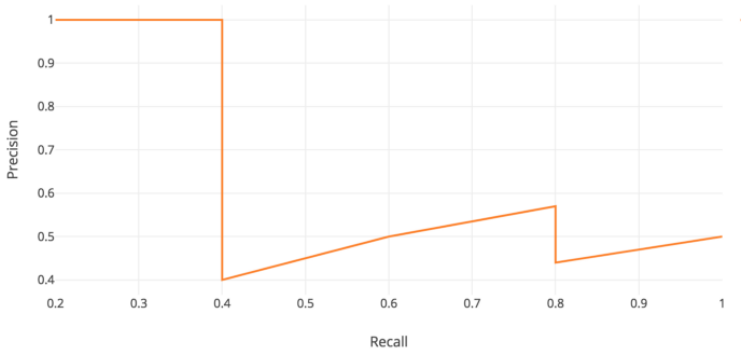


Figure 3.12: Example of precision-recall curve [21]

3.2.20 Total loss

Total loss is how good the model performs, the closer to zero the better. When the model is training it's this total loss it tries to reduce when changing its weights. If the total loss is very close to zero, overfitting can often occur [17]. Overfitting will be explained in the next section.

Overfitting and underfitting

If a models performance is bad than overfitting or underfitting can be the cause. Overfitting is when the models learns the dataset so good that when new data is presented the model can not generalize and therefore performs bad. This can be difficult to spot due to good performance on the training data. Underfitting is when there is not enough data for the model to be generalize. This is often easier to spot due to bad performance on the training data [5].

3.3 Programming language

Programming language is a formal language that makes it possible to write human text to a computer. The compiler of the programming language makes this text into computer instruction and execute the code. Therefore it's possible to make advance programs and algorithms with minimal knowledge of computers [25].

3.3.1 Python

Python is an interpreted high-level general-purpose programming language. The language focuses on code readability and is a object-oriented language. Python is often described as a "batteries included" language for its huge standard library and its easy access. Python consistently ranks as one of the most popular programming languages [43].

3.3.2 Framework

A framework is software or code that is already written. The code is written in a such way that it's easy to manipulated for the user, and is often open source. Meaning that everyone have access to the code and are allowed to change it.

There are several benefits of this. One such benefit is accelerated development. The amount of knowledge need to succeed is lowered. The code is usually more efficient and easier to debug [16]. An example of a framework is TensorFlow [18].

Chapter 4

Materials and methods

4.1 Project approach

As mention before this is a project done as a preproject before the master thesis next semester. This project is also done in collaboration with the company Zebop Avalon [2]. This means they will help with hardware and knowledge, but in return the software developed in project can be used by them.

Every other Monday a meeting with the supervisor and at least one member of Zebop Avalon was held. In this meeting the progress of the project was presented and if any problem had occurred these was discussed. Every Wednesday the offices at Zebop Avalon was used as a workplace. This was to get closer to people that could help and to discus problems.

4.2 Materials

4.2.1 Camera

The camera used is a thermal camera from FLIR called Boson 640. The camera can deliver thermal video in both grayscale and color, and it also films in normal RGB color at the same time. The camera delivers the thermal images at 60 frames per second, with a resolution of 640 × 512 [13]. The camera can be seen in the 4.1 figure below.

Figure 4.1: Thermal camera [13]

4.2.2 Hardware

The computer used for training and testing the different models is equipped with a Nvidia 1080ti Graphics processing unit (GPU). This GPU have 11 GB of RAM and 3584 CUDA Cores that can be used for ML [40]. The rest of the computer contains 24 GB RAM at a speed of 3200 MHz. A Intel i7 7700k Central Processing Unit (CPU) with 4 cores, 8 threads and clock speed of 4.2 GHz with a boost to 4.5 GHz.

4.3 Software and libraries

The listed software and libraries below has been used throughout this project.

4.3.1 Software

- **Overleaf** - A web page editor for LaTeX that gives the authors tools like real time collaborative, spell checking and cloud-based storing [42].
- **Git** - A free open source distributed version control system [15].
- **Draw.io** - A free web page editor for making flowcharts, charts and diagrams. [10].
- **VLC** - A free open source media player with a rich set of functions [61].
- **LabelImg** - A graphical image annotation tool [60].
- **Visual studio code** - An integrated development environment made by Microsoft that enable tools like debugging, syntax highlighting, intelligent code completion and have git implemented into it [35].

4.3.2 Python libraries

The libraries below has been used in combinantion with visual studio code.

- **OpenCV** - An open source computer vision library for real-time computer vision [41].
- **Numpy** - a library used for very fast mathematical operations, and include support for large, multi-dimensional arrays and matrices [39].
- **TensorFlow** - An open source framework used for ML developed by Google [18]
- **Darknet** - An open source framework used with the YOLO object detection [44].

4.4 Testing

There were several tests conducted during the project. The next sections will explain the different test approaches.

4.4.1 Convolution neural network testing

When testing several CNN a standard way to measure the performace was needed. The following evaluation metrics were used when CNNs were compared:

- mAP
- Inference time
- FPS
- Performance on a standard dataset for testing

4.4.2 Framework testing

Both TensorFlow and Darknet were tested and compared. Both of these have a lighter version of itself called TensorFlow Lite [57] and YOLO tiny [45] that is made for mobiles and devices with small computing power. When comparing these framework the biggest factors to compare are:

- Compatibility
- Speed
- Performance of CNN

4.5 Implementation

4.5.1 Data

Since the camera on Solsiden was not mounted until late November, a different place was needed to collect relevant data. The place chosen was Grilstad in Trondheim. This location has a similar layout to Solsiden with a fence and a two to three meter drop to the water. The data was collected during one sessions with different movements, number of people in the frame, people falling and in the water. This was to get a wide variety of data that could be useful for training.

4.5.2 Extracting frames

The camera captures a video with 60 frames per second (FPS). This means that in one second 60 images are taken. Therefore a video lasting one minute gives a maximum of 60 frames * 60 seconds = 3600 images. Taking this into account the number of frames extracted from each video was optimized for getting the best images from minimum data. When the video contain little action like walking, every 20. frame was extracted. When the video contain more rapid moved like a person falling every fifth frame was extracted. The first scenario gives a new frame every $\frac{20}{60} = 0,33s \approx 333ms$ and the second scenario $\frac{5}{60} = 0,083s \approx 83,33ms$. The software used for extracting frames was VLC [61].

4.5.3 Dataset

There was several thousand frames extracted from the videos, but only the frames containing one or more persons, a person falling or a person in the water could me used as this is the labels used. After removing the pictures that could not be used, the total dataset contain 2265 images. These images were manual labeled with every person, person falling and person in the water.

4.5.4 Labeling

The software used for labeling was LabelImg [60], and an example of labeling a picture with a person falling can be seen in the figure 4.2. The software saved the data in the PASCAL VOC format that is used in the TF models, and was later transformed with software to the YOLO format so that both types of framework could be used.

Figure 4.2: Labeling

4.5.5 Convolution neural network

Faster-RCNN

Faster-RCNN is a two-stage detector, and the architecture can be seen in the figure 4.3 below. In the figure the VGG16 network is used to extract features. This can be change to several different networks [7].

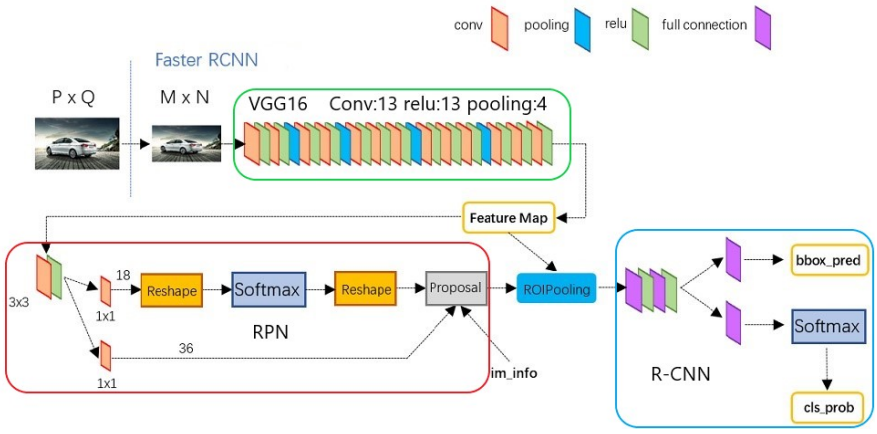


Figure 4.3: Faster-RCNN [7]

SSD

Single Shot Detector is a single-stage detector, and the architecture can be seen in the figure 4.4 below. Also here the VGG16 network can be change to several different networks [30].

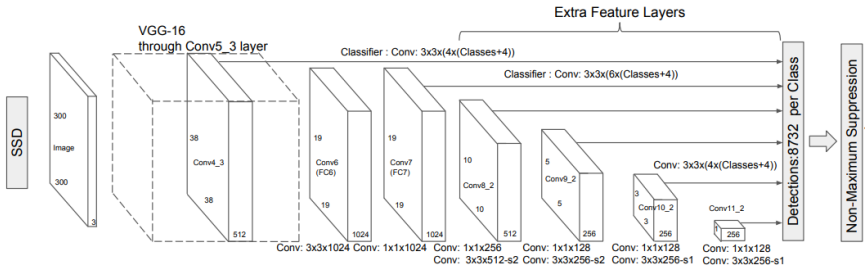


Figure 4.4: SSD [30]

YOLO

You only look once (YOLO) is a single-stage detector, and the architecture can be seen in the figure 4.5 below.

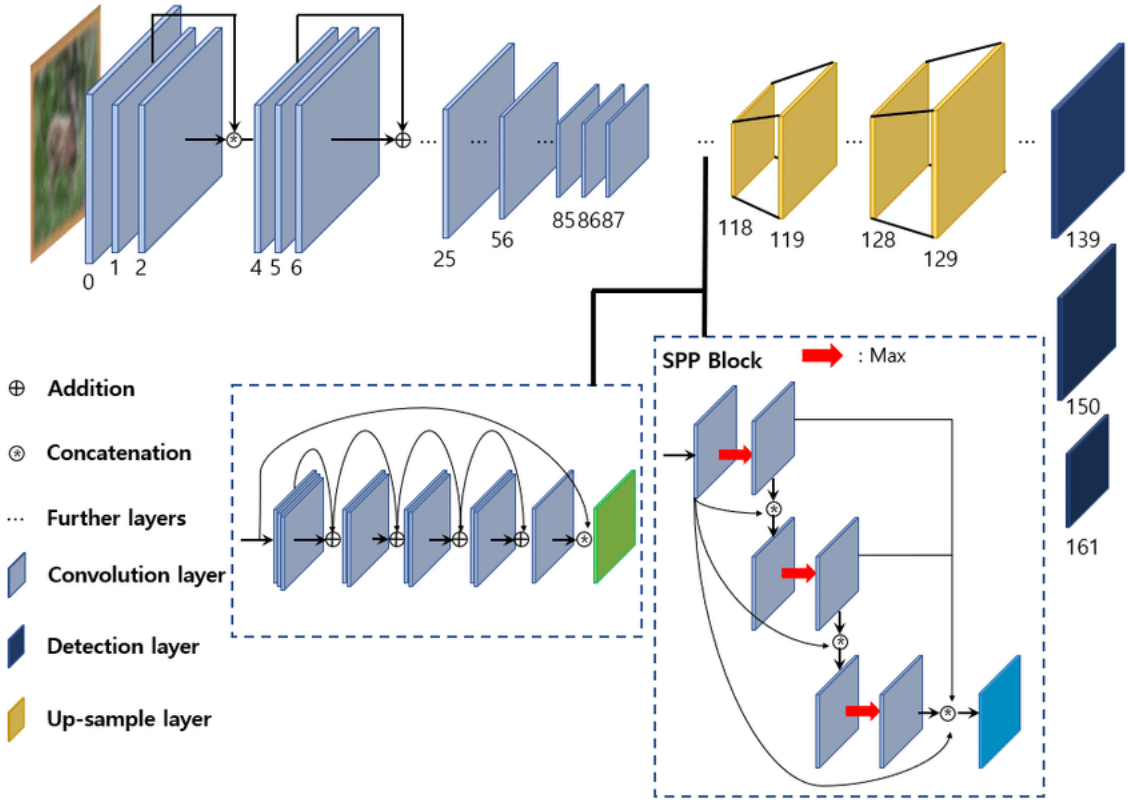


Figure 4.5: YOLO [26]

4.5.6 Pipeline test

Three models were trained in the first test. Two in TensorFlow and one in darknet. TensorFlow have many pretrained models that can be found on this site [58], and from here the two models trained were SSD ResNet50 V1 FPN 640x640 (RetinaNet50) and Faster R-CNN ResNet101 V1 640x640. From this one can see that they uses a ResNet50 and ResNet101 respectively instead of a VGG16 network as shown in figure 4.3 and 4.4. In darknet a YOLOv4 model was trained. The pretrained model was found on this website [1], and is called YOLOv4.conv.137 and consist of 137 convolutional layers.

When making the pipeline for training and testing using TF this tutorial [62] was used for help. The same is true for darknet with this tutorial [56]. The dataset used consisted of 2265 images that was split in to 80% training and 20% test sets. The TF models used online data augmentation with random horizontal flip. The YOLO model didn't use any form of data augmentation.

Since Tensorflow and darknet uses some different dependencies some problems can occur. Performing this first test with the pipeline should eliminate this kind of problems, both now and in the future. This will lower the time it takes to train new models in the future.

4.5.7 Stock versus trained models

Since all the data was used in the pipeline test and this can lead to uncertainties when it comes to the result some new models where trained from scratch to exclude this uncertainties. The models used in this test were the same as in the last test, but some changes to the dataset were made. One video containing a person falling into the water was left out from both the training and test datasets. This was 233 images so the new dataset consisted of 2032 images. This dataset was then spilt in to 80% training and 20% test sets as before.

When leaving a video out of both the training and test sets it's easier to get a more correct result due to the models has never seen the data before. Doing it this way the stock and trained models gets a more fair comparison.

4.5.8 Test on new data

The third test consisted of the models trained in the second test and their stock versions. A new YOLO tiny model was also trained for this test using the same dataset as the second test. A new standard validation dataset was also made consisting of eight pictures from the recording left out in the second test, and includes person, person falling and person in water images. It also contains twelve new, never before seen, pictures from the camera at Solsiden. In the pictures from Solsiden only people on land is present. The test was manually conducted where each picture was tested and the result written down.

Due to the data in the dataset being quite similar it's important to test how good the model is to generalize with data that is different and has never seen before. This test will focus on how good the model can generalize the data from Grilstad when performing on data from Solsiden.

Chapter 5

Result

5.1 Pipeline test

In this test the main focus was on the pipeline performance rather than the model performance, this will be talked about in the next section 5.2. From the table 5.1 below the most important results from the first test are listed. Due to the models having slightly different configurations the training time varies from four to seven hours. The models were trained at night without supervision, and all finished without any problems the first time training.

Name	Training time	Recall	mAP @ IoU 0.50	mAP @ IoU 0.75	mAP	Inference time
Faster-RCNN	ca. 4 hours	0.6095	0.9944	0.8151	0.6621	ca. 144 ms
SSD	ca. 7 hours	0.5562	0.9827	0.6648	0.6077	ca. 97 ms
YOLO	ca. 6 hours	0.98	0.9945	0.6439	-	ca. 24 ms

Table 5.1: Training score test 1

In the figures 5.1 and 5.2 below the training process of the TF models can be observed. The models trained for a total of 25 000 epochs, and had a steady fall in loss through this period. The Faster-RCNN loss for the training ended at a value of 0.008, and the test dataset ended on 0.057. The loss for the SSD model ended at a value of 0.18, and the loss for the test dataset ended on 0.32.

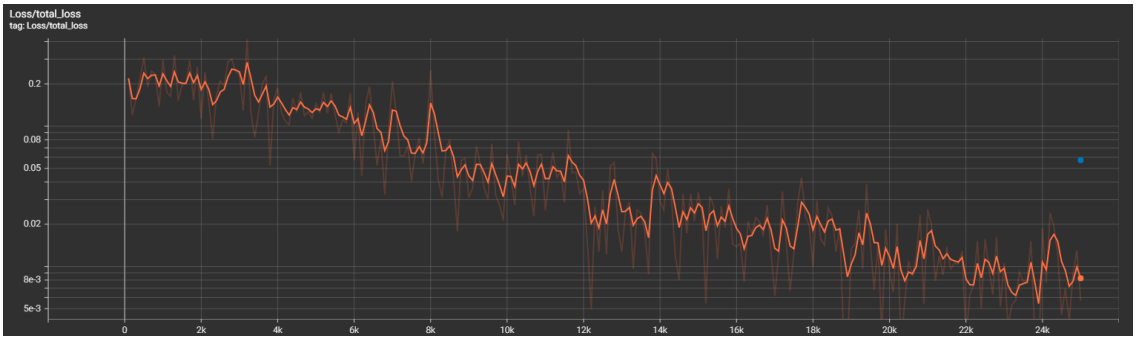


Figure 5.1: Faster-RCNN training

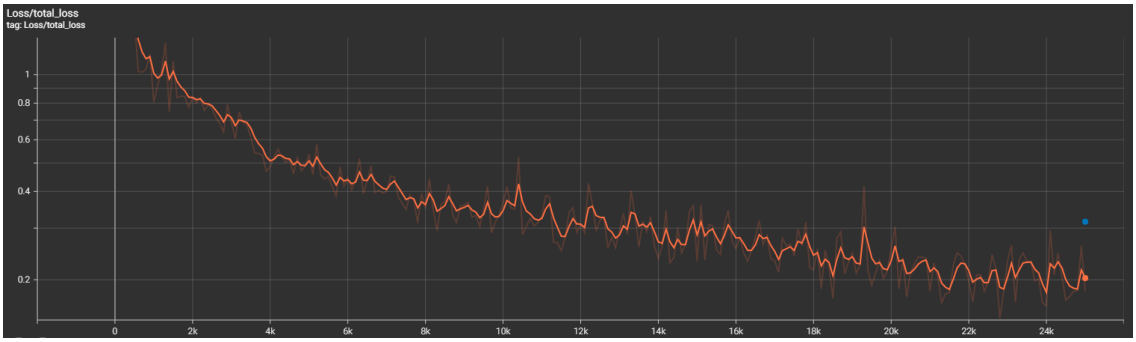


Figure 5.2: SSD training

In the figure 5.3 below one can see the training loss versus the mAP of the YOLO model during its training period. The model trained for a total of 6000 epochs, and had a fast decline in loss at the beginning before stabilizing relativity fast on a low loss results. From the figure it can be seen that the highest mAP achieved was 99% and a loss of about 0.2.

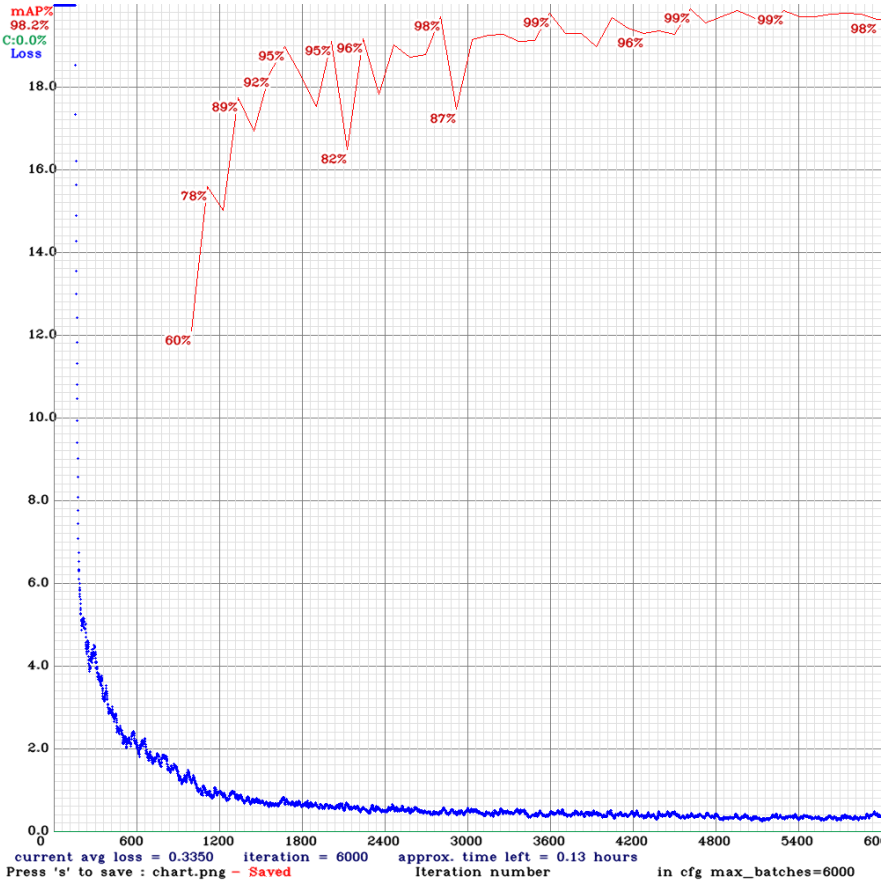


Figure 5.3: YOLO training

5.2 Stock versus trained models

In the table 5.2 the results from the second test are listed. From this table one can see that the Faster-RCNN has the highest mAP with 0.75 IoU and the second highest 0.5 IoU, but also has the highest inference time at 144 ms. The YOLO model has the highest mAP with 0.50 IoU and the lowest inference time with 24 ms. The YOLO also outperforms the two other models when it comes to the recall with a 0.98 out of 1.

Name	Training time	Recall	mAP @ IoU 0.50	mAP @ IoU 0.75	mAP	Inference time
Faster-RCNN	ca. 4 hours	0.6047	0.9922	0.8098	0.6644	ca. 144 ms
SSD	ca. 7 hours	0.5876	0.9789	0.7564	0.6586	ca. 97 ms
YOLO	ca. 6 hours	0.98	0.9925	0.6424	-	ca. 24 ms

Table 5.2: Training score test 2

In the figure 5.4 the loss at each epoch during training for the second test is shown. The most important thing to look at when it comes to performance is the loss at the end. For the Faster-RCNN the training loss ended on 0.0054 and the evaluate ended on 0.055, this is a 10.19 times difference.

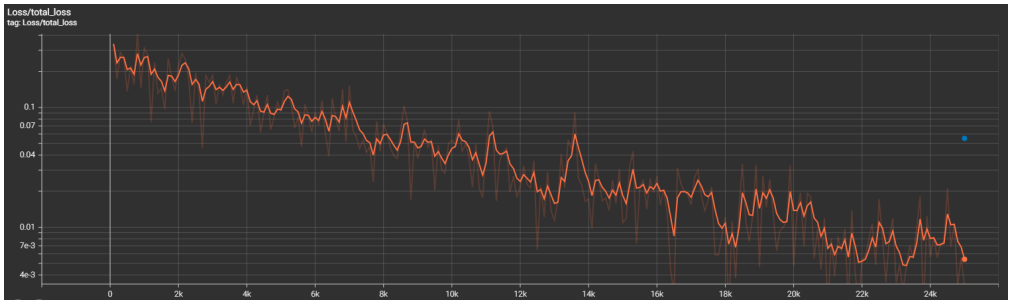


Figure 5.4: Faster-RCNN training

In the figure 5.5 the loss during training for the SSD model is shown. The same benchmark applies here with the training loss ending on 0.2 and the evaluate ending on 0.3242, a 1.62 times difference.

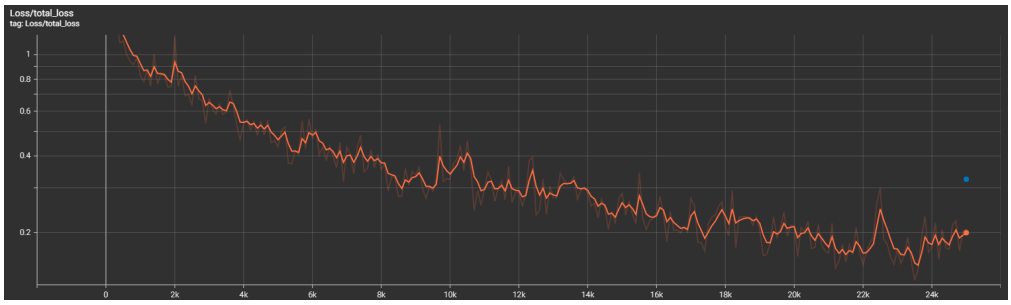


Figure 5.5: SSD training

Darknet presents the training evolution different than TF, here the figure 5.6 represent the training loss versus the mAP of the YOLO model during its training period. From the figure it can be seen that the highest mAP achieved was 99% and ending with a loss of about 0.3.

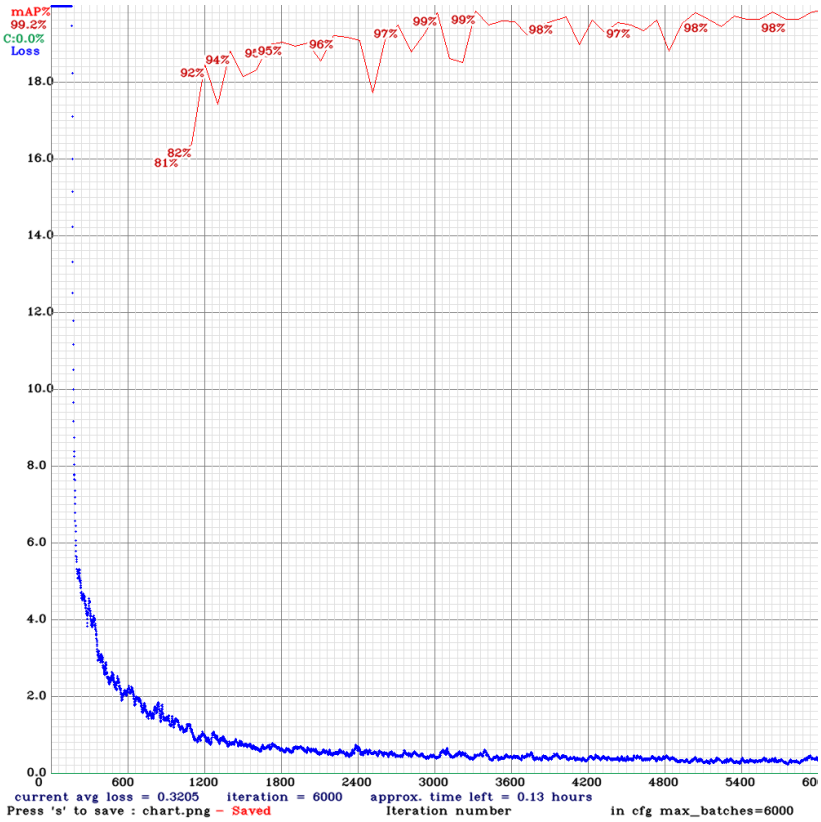


Figure 5.6: YOLO training

In the table 5.3 below four frames from the video are shown. The red boxes shows the ground truth of the predictions, and the boxes with labels show the model prediction. From the table one can see that all the stock models are struggling to predict any results. The threshold for showing a prediction was 50%. The best performing model was the Faster-RCNN with a 100% correctness, and the YOLO on second place with one missed prediction.

Stock Faster-RCNN				
Faster-RCNN				
Stock SSD				
SSD				
Stock YOLO				
YOLO				

Table 5.3: Test examples. The red boxes are the ground truth

To show the complete result a video was made that shows the stock and trained models running parallel on the whole video for easier comparison. The result can be seen in the following video https://youtu.be/Bwc606nS7Hw. The FPS on the stock YOLO model was about 21 and 31 FPS with the trained model. The stock Faster-RCNN had ca. 6 in FPS versus the trained with about 7 FPS. The SSD models had about 4 FPS with the stock model and 10 FPS on the trained model. All stock models suffer from lower FPS than their trained versions, and the reason for this is that the stock models have 80 labels to check for instead of three labels on the trained models.

5.3 Test on new data

In the figure 5.7 the loss versus the mAP for the YOLO tiny model during its training period can be seen. The loss ended on ca. 0.1 with the highest mAP on 96%, and the training took about one hour. In this video, https://youtu.be/D4SaMiFgGJ0, the model running on the video left out from the dataset can be seen. The average FPS on the whole video was 65.

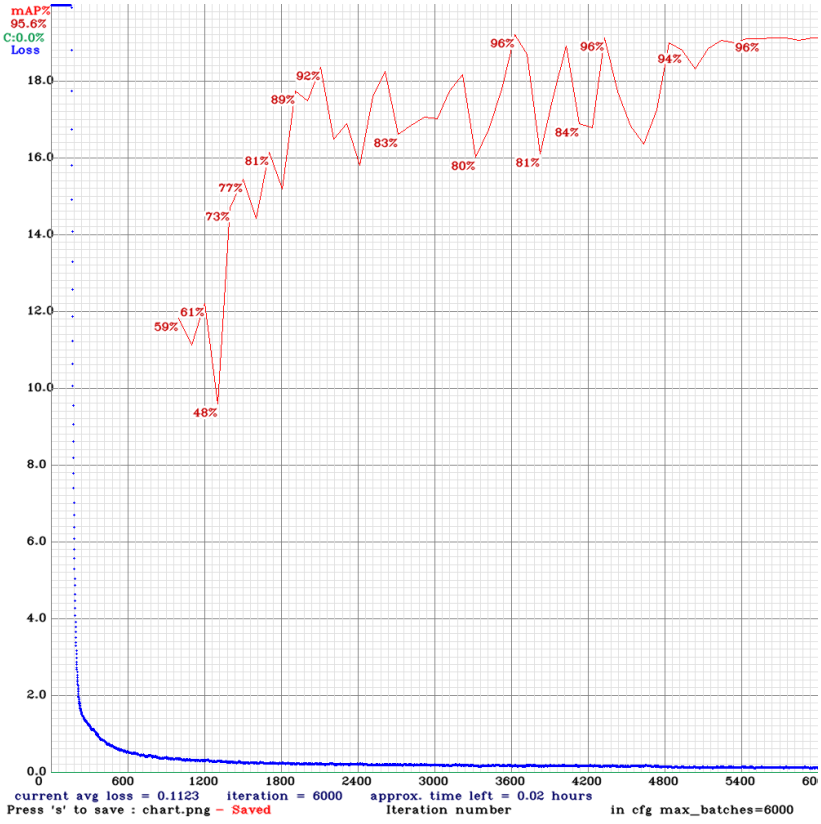


Figure 5.7: YOLO tiny training

In the first part of this test eight new pictures from the left out dataset was used. The result was equivalent to that of the second test where the trained models outperforms the stock models. In table 5.4 the performance of the models on the eight images used are listed.

Name	TP	FP	FN	Precision	Recall
Stock Faster R-CNN	1	0	13	1	1/14
Faster R-CNN	14	0	0	1	1
Stock SSD	0	0	14	N/A	N/A
SSD	7	0	7	1	1/2
Stock YOLO	2	0	12	1	1/7
YOLO	14	0	0	1	1
YOLO tiny	12	0	2	1	6/7

Table 5.4: Evaluation of images from recording

In the second part the models were tested on new images. In table 5.5 the performance on the twelve images extracted from Solsiden are listed. Both the stock and trained SSD model did not do a single detection. The result was done without a strict defined ground truth so result may vary a little, but gives a overall impression.

Name	TP	FP	FN	Precision	Recall
Stock Faster R-CNN	25	2	23	25/27	25/48
Faster R-CNN	47	20	5	47/67	47/52
Stock SSD	-	-	-	N/A	N/A
SSD	-	-	-	N/A	N/A
Stock YOLO	58	3	6	58/61	29/32
YOLO	71	25	0	71/96	1
YOLO tiny	56	8	7	7/8	8/9

Table 5.5: Evaluation of images from Solsiden

Two pictures from part one and two, a totalt of four, was used to give a visual presentation of the result. In the table 5.6 below the images from this test are shown, and easily compared. One thing to notice is the false positives in Faster-RCNN, YOLO and YOLO Tiny models when performing on the Solsiden images.

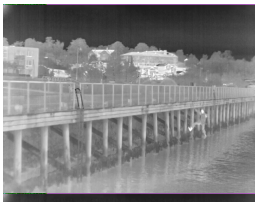
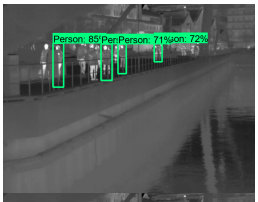
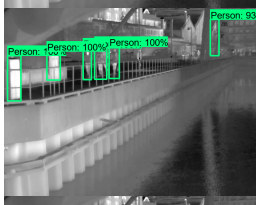
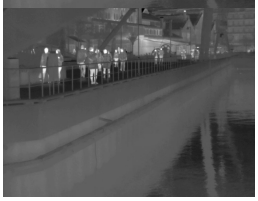
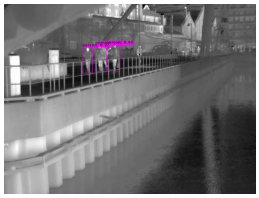
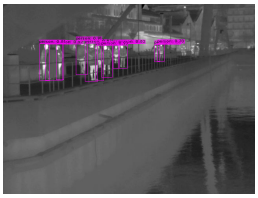
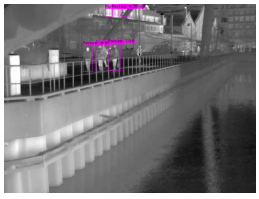
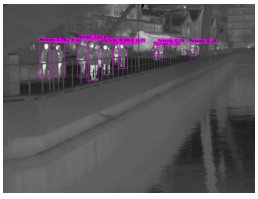
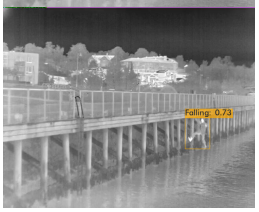
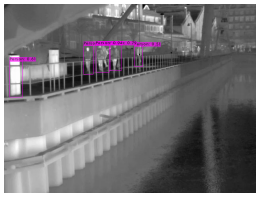
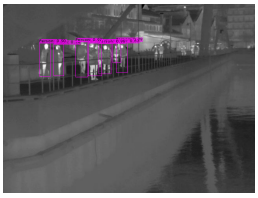
Stock Faster-RCNN				
Faster-RCNN				
Stock SSD				
SSD				
Stock YOLO				
YOLO				
YOLO Tiny				

Table 5.6: Test examples

Chapter 6

Discussion

6.1 Dataset

The dataset was only recorded during one session with nice weather.Therefore the dataset lack images with external noise such as rain, snow and fog that can often occur in Trondheim. The dataset also only includes seven videos of a person falling. Taking into account that a person uses about 640ms to fall two meters and every fifth frame is extracted at a interval of 83,33ms. This gives us about $\frac{640}{83,33}$ = 7 images per video and a total of 49 pictures of a person falling. When comparing this to the total dataset $\frac{49}{2265} * 100$ = 2.1%. Given there are three different labels this number should be closer to 33.33%. However the dataset can give an indication on which model performs the best and fastest with some few remarks.

6.2 Test results

6.2.1 Pipeline test

Since all the data was used in this test the performance is difficult to evaluate. Therefore the test focused more on testing the pipeline and making it ready for the next test. The result from this was that the remaining test was faster to perform, and when a new dataset was made the training process could start in a few minutes. Fixing all the dependencies in this test also made

it a lot easier to avoid errors in the remaining tests. One thing to point out is how low the loss was in the Faster-RCNN model. This is often correlated to overfitting. A problem on the TF models and probably the YOLO one too is the big deviation on the loss for the train and evaluate datasets. This is often called generalization gap, and the smaller the better and is often a overfitting problem. A better dataset can help minimizing this error.

6.2.2 Stock versus trained models

In this test the same indication of overfitting can be seen from the training data, with the worst being the Faster-RCNN with a ten times difference between the training and evaluating. It's also possible to see on the video that for instance the faster-RCNN model almost frame perfect changes label from person to person falling. This can be a very good model, but can easily also be overfitting. From the video it's also possible to see that the trained models have improved a lot when it comes to detection of people, and all the stock models performs very bad. The biggest take away from this test is how much faster the YOLO model is compared to the other models, and performance is close to the best which is the faster-RCNN model.

6.2.3 Test on new data

The YOLO tiny model performed surprising well, and had a 100% increase in FPS over the normal YOLO model that was already much faster then the other models. Detection wise the model also performed very good. When evaluating the two small test datasets it can be seen that the training helps the models improve their performance. However the training also cause more false positive, i.e errors in the dataset from Solsiden, indicating overfitting.

The performance validation for the dataset on Solsiden was conducted without a strict ground truth. However there are still some points to be made out from this. The stock models performed much better on the Solsiden test then the original dataset. Probably the biggest reason for this is that the images from Solsiden more close resembles the RGB images used for training, in the form of shapes and contrast. The stock YOLO model performed so well that it could be used in a real system, and makes a great foundation for a trained model.

6.3 Model

As mention the time it takes for a person to fall into the water is very short, and therefore speed of the models are incredible important. As stated before it takes about 600 ms to fall two meters, and to be sure a person is captured when falling at least two frames should be taken in this time. This gives a new frame every 300 ms, and a FPS of $\frac{1}{0.3} \approx 4$. This is the bare minimum, and a FPS that is two or three times this is desirable. Here the two YOLO models stands out both in speed and performance. Since this test was running on a powerful computer, and that may not be true in the future, the YOLO tiny model may be the only viable option to get enough frames to detect a fall.

6.4 Framework

Since there are no real benchmark for framework the comparison will be based on personal experience. The hardest part of both framework was the installation and getting the right dependencies. After that the training procedure felt the same in terms of difficult. Both framework also seems to have the same compatibility, and with theirs smaller version limited hardware should be no problem. TF have more models in their model zoo and seems more popular compared to darknet. A quick google search on tensorflow gives about 60 300 000 results versus darknet at 11 900 000 results, even with darknet other meaning included. However due to the performance and the speed of the YOLO models it's hard not to continue using one of these.

Chapter 7

Conclusions

From the results it's possible to see that a varied dataset is important, and the presented dataset has a lot of improvement potential when it comes to an evenly distributed number of images per label. Further the results from the pretrained and trained models shows that training a model on thermal images improve theirs results, but can also lead to more false predictions. This error can be a result from overfitting, and a better dataset can minimize this. Since inference speed is very important to detect a person falling in water a model with high speed seems to be the best choice. Here the YOLO models stands out both in accuracy and speed, and seems like the best choice to use in further development. If the solution need to run on limit hardware the YOLO tiny model can be a good choice due to it being by far the smallest and fastest model tested. When is comes to framework the TensorFlow framework seems to be the most popular, but also here due to the speed of the models in the darknet framework it seems like the best choice. All in all this thesis gives a good starting point for further development in the master thesis.

With the results from this preproject the master thesis will benefit in several ways. Firstly, since the pipeline is finished the time it takes to trained new models are almost zero. Secondly the need for more data is clear, and this gathering of new data can start as soon as possible. While gathering more data the focus can shift over to other methods that can compliment the ML like a motion tracker that can see if a person goes from a safe to a unsafe area. This will make the machine learning even more robust.

7.1 Further work

There are several things that can be improved and hopefully will be in the master thesis:

- Improve the dataset
- Test solutions on different hardware
- Extract the data from the predictions to raise an alarm when someone is falling in the water
- Implement motion tracking or some other method to help with predictions
- Live test on Solsiden

Bibliography

[1] AlexeyAB. Yolov4 pre-release, 05 2020. URL https://github.com/AlexeyAB/darknet/releases. accessed 21.10.21.

[2] Zebop Avalon. Zebop avalon, 09 2021. URL https://www.zebopavalon.com/. accessed 12.09.21.

[3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020. URL https://arxiv.org/abs/2004.10934.

[4] Trøndelag brann-og redningstjeneste IKS. elvrespekt, 08 2021. URL https://www.facebook.com/watch/?v=371537721215435&ref=sharing. accessed 28.09.21.

[5] Jason Brownlee. Overfitting and underfitting with machine learning algorithms, 08 2019. URL https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/. accessed 15.09.21.

[6] Christopher Brunner, Thierry Peynot, Teresa Vidal-Calleja, and James Underwood. Selective combination of visual and thermal imaging for resilient localization in adverse conditions: Day and night, smoke and fire, 11 2012. URL https://doi.org/10.1002/rob.21464. accessed 15.12.21.

[7] Xander CAI. How does faster r-cnn work: Part i, 4 2021. URL https://www.lablab.top/post/how-does-faster-r-cnn-work-part-i/. accessed 21.10.21.

[8] cdeterman. what is a 'layer' in a neural network, 02 2016. URL https://stackoverflow.com/a/35347548. accessed 27.10.21.

[9] COCO. Coco dataset, 09 2021. URL https://cocodataset.org/#home. accessed 15.09.21.

[10] diagrams. draw.io, 09 2021. URL https://www.diagrams.net/. accessed 12.09.21.

[11] IBM Cloud Education. Machine learning, 07 2020. URL https://www.ibm.com/cloud/learn/machine-learning#toc-machine-le-K7VszOk6. accessed 08.10.21.

[12] Maël Fabien. Local features, detection, description and matching, 03 2019. URL https://maelfabien.github.io/computervision/cv_4/#. accessed 30.09.21.

[13] FLIR. Boson 640, 11 2021. URL https://www.flir.com/products/boson/?model=20640A018. accessed 03.11.21.

[14] Arun Gandhi. Data augmentation how to use deep learning when you have limited data part 2, 05 2021. URL https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/. accessed 30.09.21.

[15] git. git, 09 2021. URL https://git-scm.com/. accessed 12.09.21.

[16] Ekta Goel. Software framework vs library, 09 2020. URL https://www.geeksforgeeks.org/software-framework-vs-library/. accessed 09.09.21.

[17] Google. Descending into ml: Training and loss, 02 2020. URL https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss. accessed 25.09.21.

[18] Google. Tensorflow, 09 2021. URL https://www.tensorflow.org/. accessed 09.09.21.

[19] Tom Harris. How cameras work, 09 2021. URL https://electronics.howstuffworks.com/camera.htm. accessed 02.09.21.

[20] HMKCODE. Backpropagation step by step, 11 2019. URL https://hmkcode.com/ai/backpropagation-step-by-step/. accessed 09.10.21.

[21] Jonathan Hui. map (mean average precision) for object detection, 03 2018. URL https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173. accessed 25.09.21.

[22] IBM. What is computer vision?, 09 2021. URL https://www.ibm.com/topics/computer-vision. accessed 02.09.21.

[23] STEMMER IMAGING. Region of interest (roi), 09 2021. URL https://www.stemmer-imaging.com/en-pl/knowledge-base/region-of-interest-roi/. accessed 08.09.21.

[24] Rohan Ippalapally, Sri Harsha Mudumba, Meghana Adkay, and Nandi Vardhan H. R. Object detection using thermal imaging, 2020.

[25] JavaTpoint. Programming language, 09 2021. URL https://www.javatpoint.com/programming-language. accessed 02.09.21.

[26] Sungrae KIM and Hyun Kim. Zero-centered fixed-point quantization with iterative retraining for deep convolutional neural network-based object detectors. *IEEE Access*, PP:1–1, 01 2021. doi: 10.1109/ACCESS.2021.3054879. accessed 21.10.21.

[27] Simeon Kostadinov. What is deep transfer learning and why is it becoming so popular?, 11 2019. URL https://towardsdatascience.com/what-is-deep-transfer-learning-and-why-is-it-becoming-so-popular-91acdcc2717a. accessed 15.09.21.

[28] Mate Krišto, Marina Ivasic-Kos, and Miran Pobar. Thermal object detection in difficult weather conditions using yolo. *IEEE Access*, 8:125459–125476, 2020. doi: 10.1109/ACCESS.2020.3007481. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9133581.

[29] Nishant Kumar. Digital image processing basics, 07 2021. URL https://www.geeksforgeeks.org/digital-image-processing-basics/. accessed 29.09.21.

[30] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015. URL http://arxiv.org/abs/1512.02325. accessed 21.10.21.

[31] Lynred. Infrared technology and thermal cameras: How they work, 09 2021. URL https://www.lynred.com/blog/infrared-technology-and-thermal-cameras-how-they-work. accessed 08.09.21.

[32] MathWorks. Edge detection methods for finding object boundaries in images, 09 2021. URL https://www.mathworks.com/discovery/edge-detection.html. accessed 30.09.21.

[33] Maximinusjoshus. Understanding the concept of channels in an image, 04 2021. URL https://medium.com/featurepreneur/understanding-the-concept-of-channels-in-an-image-6d59d4dafaa9. accessed 29.09.21.

[34] Nick McCullum. Deep learning neural networks explained in plain english, 06 2020. URL https://www.freecodecamp.org/news/deep-learning-neural-networks-explained-in-plain-english/. accessed 27.10.21.

[35] Microsoft. Visual studio code, 09 2021. URL https://code.visualstudio.com/. accessed 12.09.21.

[36] Farzeen Munir, Shoaib Azam, Muhammd Aasim Rafique, Ahmad Muqeem Sheri, Moongu Jeon, and Witold Pedrycz. Exploring thermal images for object detection in underexposure regions for autonomous driving, 2021.

[37] Vanessa Ndonhong, Anqi Bao, and Olivier Germain. Wellbore schematics to structured data using artificial intelligence tools, 04 2019. URL https://www.researchgate.net/publication/308320592_Fast_Single_Shot_Detection_and_Pose_Estimation. accessed 12.10.21.

[38] NRK. Henter opp døde mennesker fra nidelva hvert eneste år, 11 2019. URL https://www.nrk.no/trondelag/henter-opp-dode-mennesker-fra-nidelva-hvert-eneste-ar-1.13911303. accessed 02.09.21.

[39] NumPy. What is numpy?, 06 2021. URL https://numpy.org/doc/stable/user/whatisnumpy.html. accessed 12.09.21.

[40] Nvidia. 1080ti, 09 2021. URL https://www.nvidia.com/en-gb/geforce/graphics-cards/geforce-gtx-1080-ti/specifications/. accessed 11.09.21.

[41] opencv. opencv, 09 2021. URL https://opencv.org/. accessed 12.09.21.

[42] Overleaf. About us, 09 2021. URL https://www.overleaf.com/about. accessed 12.09.21.

[43] Python. Python, 09 2021. URL https://www.python.org/about/. accessed 02.09.21.

[44] Joseph Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016. accessed 12.09.21.

[45] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018. URL https://pjreddie.com/darknet/yolo/. accessed 13.10.21.

[46] Redningsselskapet. Redningsselskapet krever nullvisjon for drukningsulykker, 09 2021. URL https://kommunikasjon.ntb.no/pressemelding/redningsselskapet-krever-nullvisjon-for-drukningsulykker-?publisherId=89422&releaseId=17915950. accessed 25.09.21.

[47] riptutorial. Activation functions, 10 2021. URL https://riptutorial.com/machine-learning/example/31624/activation-functions. accessed 09.10.21.

[48] Christopher Dahlin Rodin, Luciano Netto de Lima, Fabio Augusto de Alcantara Andrade, Diego Barreto Haddad, Tor Arne Johansen, and Rune Storvold. Object classification in thermal images using convolutional neural networks for search and rescue missions with unmanned aerial systems. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2018. doi: 10.1109/IJCNN.2018.8489465.

[49] Sumit Saha. A comprehensive guide to convolutional neural networks, 12 2018. URL https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. accessed 09.10.21.

[50] David C. Schedl, Indrajit Kurmi, and Oliver Bimber. Search and rescue with airborne optical sectioning. *Nature Machine Intelligence*, 2(12):783–790, Nov 2020. ISSN 2522-5839. doi: 10.1038/s42256-020-00261-3. URL http://dx.doi.org/10.1038/s42256-020-00261-3.

[51] ScienceDirect. Feature extraction, 09 2021. URL https://www.sciencedirect.com/topics/engineering/feature-extraction. accessed 30.09.21.

[52] Petru Soviany and Radu Tudor Ionescu. Optimizing the trade-off between single-stage and

two-stage object detectors using image difficulty prediction, 08 2018. URL http://arxiv.org/abs/1803.08707. accessed 12.10.21.

[53] Sreenu and Saleem Durai. Intelligent video surveillance: a review through deep learning techniques for crowd analysis, 12 2018. URL https://doi.org/10.1186/s40537-019-0212-5. accessed 15.12.21.

[54] Suman. Artificial intelligence, 10 2019. URL https://ai.stackexchange.com/questions/15859/is-machine-learning-required-for-deep-learning. accessed 08.10.21.

[55] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection, 2020.

[56] techzizou. Train a custom yolov4 object detector on windows, 08 2021. URL https://techzizou.com/train-a-custom-yolov4-object-detector-on-windows/. accessed 21.10.21.

[57] TensorFlow. Deploy machine learning models on mobile and iot devices, 10 2021. URL https://www.tensorflow.org/lite. accessed 13.10.21.

[58] TensorFlow. Tensorflow 2 detection model zoo, 05 2021. URL https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md. accessed 21.10.21.

[59] Fram tid i nord. Så mange døde av drukning i 2018, 01 2019. URL https://www.framtidinord.no/nyheter/2019/01/10/S%C3%A5-mange-d%C3%B8de-av-drukning-i-2018-18219965.ece. accessed 02.09.21.

[60] Tzutalin. Labelimg, 07 2021. URL https://github.com/tzutalin/labelImg. accessed 12.09.21.

[61] videolan. Vlc features, 09 2021. URL https://www.videolan.org/vlc/features.html. accessed 12.09.21.

[62] Lyudmil Vladimirov. Tensorflow 2 object detection api tutorial, 06 2021. URL https:

//tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/. accessed 21.10.21.

[63] Lun Zhang, Stan Z. Li, Xiaotong Yuan, and Shiming Xiang. Real-time object classification in video surveillance based on appearance learning. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007. doi: 10.1109/CVPR.2007.383503.

Petter Tafjord Drønnen

Detection of a person in the water from thermal images

NTNU
Kunnskap for en bedre verden