NOVIA
UNIVERSITY OF APPLIED SCIENCES

Water Level Measurement From Images Using Object Detection

Bo-Anders Näs

Degree Thesis

Thesis for a Master of Engineering (UAS) degree

Automation Technology

Vaasa 2023

DEGREE THESIS

Author: Bo-Anders Näs
Degree Programme and place of study: Automation Technology, Vaasa
Specialisation: Intelligent systems
Supervisor(s): Ray Pörn, Tina Sørvik

Title: Water Level Measurement From Images Using Object Detection

---

Date: 27.09.2023 Number of pages: 63 Appendices: 3

Abstract

The technology is advancing rapidly and today object detection is becoming increasingly common. Using a camera and object detection in water level measurements is useful and affordable, as cameras are often readily available at such locations. In this thesis, an application was developed to measure water level from images of a staff gauge using a Yolov5m model.

The quality of the captured images is often a problem. Strong light, reflections, and dirty staff gauges often cause the object detection to fail. To this category belongs also night-time and blurry images. A separate application was developed to handle this filtering. This application filters out these night-time and blurred images, images that otherwise the Yolov5m model would fail on.

In this thesis, two alternatives for the water level measuring application were compared. In the first alternative Yolov5m was used for detecting the staff gauge and numbers on the staff gauge and image processing was used for locating the water line. In the second alternative, the Yolov5m model was used for all three detections, for the staff gauge detection, the numbers detection on the staff gauge, and for the water line detection on the staff gauge. The second alternative using the Yolov5m model for water line detection outperformed the first alternative. The second alternative estimated the water level correctly in 40% of the test images and 83% of the test images, it estimated the water level within 3 cm of the correct level.

Language: English
Key Words: Water level, Staff gauge, Object detection, Machine learning, Yolov5

Table of Contents

Acknowledgment

This thesis was carried out in collaboration with Hafslund Eco. I would like to sincerely thank my supervisors Tina Sørvik from Hafslund Eco and Ray Pörn from Novia UAS for their invaluable support. I would also like to thank Geir Gryttingslien from Hafslund Eco, for making this possible.

Vasa, September 2023

Bo-Anders Näs

Abbreviations

CNN	Convolutional Neural Networks
COCO	Common Objects in Context
CUDA	Compute Unified Device Architecture
GPU	Graphics Processing Unit
IoU	Intersection over Union
mAP	mean Average Precision
mAP0,5	mean Average Precision for an IoU with a threshold equal to 0,5 and averaged over all classes
mAP0,5:0,95	mean Average Precision for IoU with thresholds between 0,5 and 0,95, averaged over all classes
RCNN	Region-based Convolutional Neural Networks
ReLU	Rectified Linear Unit
SPPNet	Spatial Pyramid Pooling Networks
SSD	Single Shot MultiBox Detector
YOLO	You Only Look Once

1 Introduction

Being able to assess, measure, and quantify things in the environment around us is something fundamental. It is of great importance to get a value of what things are, for many reasons. Measuring water levels has been done for many centuries or even millennia. There are records of water level measurements of the river Nile in Egypt, as early as 3000 BC.

Water level measuring, especially in a country like Norway with its high mountains and deep and numerous valleys, is essential for flood prediction and prevention. The water is also a very important source of energy. In Norway, 98 % of all energy production comes from hydropower (Renewable Energy Production in Norway, 2016). Being able to measure the water levels in lakes and dams means that you are able to measure the energy content and theoretical output of a hydropower plant that uses the water from the lake or dam. It also allows you, together with other parameters, to predict and plan when to produce electricity and how much to produce.

Today water level measuring in general is usually done using float-type, pressure, ultrasonic, microwave/radar, or optical sensors. These sensors and the whole setup needed can be expensive and the sensors also usually need some form of calibration. It can be a challenge to use these kinds of sensors in a country like Norway, that have very distinct seasons. During the winter season, these sensors also need heating or other measures to prevent freezing.

A camera with an internet connection is usually a much cheaper investment and such equipment is in many locations already available. In many mountain lakes or dams from which water is used in energy production, a visual staff gauge is installed so that the public can follow the water level. A camera can capture images of this staff gauge and by using specialized computer software, the water level could be extracted from these images.

2 Literature review

There are not that many articles that deal with water level measuring from images using object detection, but some that use traditional image processing techniques. When no object detection procedures are used, the region of interest needs to be manually located in the images and the field of view of the camera must not change during the capturing of images.

Amongst the reviewed articles only one used image processing methods for finding the staff gauge. Dou et al. used image processing and an OpenCV function called “findContours” to identify the location of the staff gauge. After locating the staff gauge, number area, and scale line area, multiple images were captured in a row. This method is sensitive to disturbances as the camera must not move during image capturing as no new image detection was performed in between. (Dou et al., 2022)

Only one article was found where they made use of machine learning in locating the staff gauge. Qiao et al. used Yolov5s to detect the staff gauge in the image. This approach showed good precision of the detection in most cases, except when there are reflections of the staff gauge on the water surface (Qiao et al., 2022). In the rest of the reviewed articles, the staff gauge was located manually within the images, before use. This, of course, makes the setup very vulnerable to interference, as Zhang et al. observed. Wind gusts made the camera shake, which made the region of interest move in relation to the captured staff gauge. (Zhang, Zhou, Liu, & Gao, 2019)

The located staff gauge can be tilted and needs to be straightened. Dou et al. and Qiao et al. used Hough transform in their respective work, to identify and correct possible tilting. (Dou et al., 2022; Qiao et al., 2022)

Out of the reviewed articles all but one applied image processing to locate the water line. The aim of the image processing was first to remove noise and disturbances, and then binarize the image. The hypothesis was that the water body is darker than the staff gauge and therefore the binarized image will clearly show the water line. The number of black pixels was then counted and summed up for each row and when the sum fell below a threshold value, this identified the water line. (Sabbatini et al., 2021; Dou et al., 2022; Zhang, Zhou, Liu, Zhang, et al., 2019; Zhang, Zhou, Liu, & Gao, 2019; Qiao et al., 2022) The

exception was de Oliveira Fleury et al. who proposed their own Convolutional Neural Network (CNN) model to find the water line and read out the water level (De Oliveira Fleury et al., 2020). They created the dataset by labelling images and classifying them according to the water level. They compared the performance of their own CNN model to two other CNN models, ResNet50 and MobileNetV2, and their CNN model was superior. Their model showed that the technology works, but the error their model had was still rather large, a root mean square error of 0,29(m), considering that their dataset had images with water levels in the range of 86,35 m to 88.89 m. (De Oliveira Fleury et al., 2020)

To calculate the water level, generally, the pixel-to-cm ratio is first determined and then the pixels between a reference point and the located water line, are counted. A reference point could be the numbers on the staff gauge, as Zhang et al. proposed, or the top of the staff gauge itself as Qiao et al. suggest. (Zhang, Zhou, Liu, & Gao, 2019; Qiao et al., 2022)

Qiao et al. used a Yolov5s model to locate the numbers on the staff gauge and another self-developed CNN model to classify the detected numbers. These numbers were then used as a reference for the calculation of the water level. (Qiao et al., 2022)

Many of the authors pointed out that image quality plays a role in the performance of the applications. Sabbatini et al. was one of two articles dealing with the image quality problem. They created an algorithm to sort out bad-quality images and split the good-quality images into day and night images. The day and night images were then processed by different methods to find the water line. (Sabbatini et al., 2021) Zhang et al. took another approach to solving the problem with image quality due to bad weather conditions. They made use of infrared imaging. (Zhang, Zhou, Liu, & Gao, 2019) Staff gauge materials and the water body reflect infrared light differently than visible light, possibly enhancing the contrast (Mangold et al., 2013). After locating the water line, the same kind of image processing procedure was performed as described earlier.

2.1 Purpose of the study

The purpose of this thesis is to create an application that computes the water level from images. To use the Yolov5 model for object detection was decided after reviewing articles on automatic water level measuring. This thesis will include all the steps needed to create this application, except for collecting the images.

3 Theory

The human eye detects the light that reaches the retina and converts the information to electrical signals. These electrical signals are then processed in the brain, into an image. A digital camera works similarly. The retina is replaced by a digital sensor that captures the light and converts it into binary form. The digital sensor is a two-dimensional matrix of photo-sensitive detectors, where each element only detects one colour, usually red, green, and blue. (Fraser, 2004)

The captured image data is then stored in an unprocessed raw format. Before processing it needs to be converted into a more easily workable format, like the RGB format. For example, a colour image with a resolution of 256x256 pixels, contains three two-dimensional matrices, one matrix per colour. Each of these matrices has the same size as the final image, 256x256 pixels, and each cell in the matrices contains the intensity value of one colour ranging from 0 to 255, as depicted in Figure 1. (Fraser, 2004)

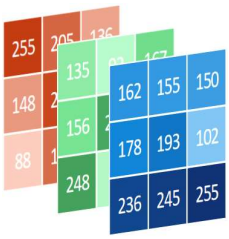

Figure 1. An RGB image illustrated in matrix format.

A grayscale conversion of this image will merge the three matrices into one, retaining the resolution. The three colours are not necessarily merged in the same proportions, which means that the pixels in the grayscale image do not contain one-third of each colour intensity. A weighted method is usually used to make the image more natural-looking. The weighted method means that the three colour intensities are multiplied with a constant that defines their proportions in the grayscale image. These constants are 0,299 for the red colour, 0,587 for the green colour, and 0,114 for the blue. These values originate from the standard:" Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios". (Radiocommunication Sector of International Telecommunication Union, 2011)

The colour in an image is defined by the intensity of the three colours described above. The edges of an object in an image are recognized by larger changes in the intensity values of neighbouring pixels. This is a feature that is one of the main things in object detection and object recognition.

3.1 Object detection and object recognition

Object detection and object recognition means the technology of using a computer to identify objects in digital images. These objects can usually be faces, humans, cars, or animals, but the possibilities are many. In 2001 Viola and Jones published the first real object detection application for facial detection and this was the beginning of a new era. (Zou et al., 2023)

With the rebirth of the Convolutional Neural Networks, or CNN, and its introduction in object detection by Krizhevsky et al. (Krizhevsky et al., 2012) and Girshick et al. (Girshick et al., 2014) in the first half of the 2010s, the development started to gain momentum. The neural network mostly used in object detection is CNN. CNN is used in a myriad of models like Single Shot multibox Detector (SSD), You Only Look Once (YOLO), Retina-Net, Region-based Convolutional Neural Networks (RCNN), and Spatial Pyramid Pooling Networks (SPPNet) among many others. (Zou et al., 2023)

3.2 Convolutional neural networks – CNN

Convolutional neural network is perhaps the most common type to be used for image recognition and that is for a reason. It is very effective in detecting features in an image. In 2014 Girshick et al. published a technical report on object detection (Girshick et al., 2014). This work is one of the earliest works where the use of CNN is used in object recognition. After this publication, the evolution of object detection took off. (Zou et al., 2023)

It is, as the name “neural network” implies, influenced by the human brain and the network of neurons it contains. The CNN is built up of layers grouped as input layer, hidden layers, and output layers (Du, 2018). The neurons, name also adapted from the human brain, are inter-connected between the layers. In these interconnections, multiplications are performed using weights and biases. (Khan et al., 2018; Zafar et al., 2018) In Figure 2 below, the neurons’ interconnections between layers are illustrated.

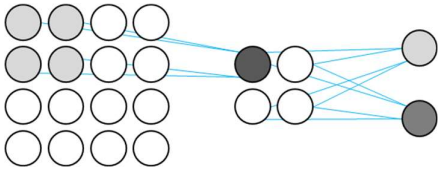

Figure 2. CNN neuron connections between layers.

Activation functions are found between the layers, and they are attached to the output of each neuron. One commonly used activation function is the Rectified Linear Unit or ReLU. It maps the positive values of the neuron and if the value is negative it is mapped to zero, which is illustrated in the upper left square in the left and middle image, in Figure 3 below. The output of the activation function can further be modified using a pooling function. The pooling function reduces the size of the feature maps by compressing smaller regions of neurons. This is done by only keeping the highest number in each square. In Figure 3, the upper left circle in the rightmost image, number four is the result of max pooling. (Du, 2018; Khan et al., 2018; Zafar et al., 2018)

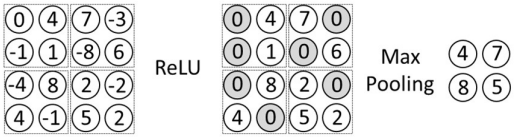

Figure 3. Example of ReLU function and Pooling function (Max pooling function).

The main building blocks in a CNN are the convolutional layers. It is in these layers all the computations occur. There are many of these layers in a CNN and they are processing the data, usually in sequence. Each of these layers is focused on detecting specific types of features. The first layers detect simple patterns like lines or curves and further along in the network the layers detect more complex features like for example faces or animals. (Du, 2018; Khan et al., 2018)

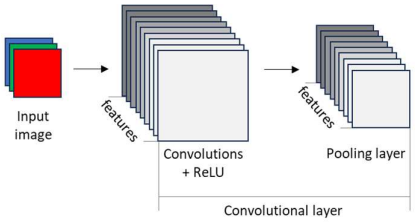

Figure 4. A convolutional layer is illustrated.

To detect these features the convolutional layers use a matrix that it convolves through an image from top-left to bottom-right. This matrix is called a filter or kernel and it consists of a square with the pixel size of 3x3, 5x5, or similar. This square contains random or pseudo-random numbers or as Figure 5 below.

1	0	-1
1	0	-1
1	0	-1

Figure 5. A 3x3 filter or kernel used in CNN

This filter, when it convolves across the image performs dot multiplication with the pixel values, i.e., pixel intensities (see Figure 6). This process is the base for feature detection in an image. With filter A, in Figure 7 below, the horizontal edges of an object are detected and with filter B vertical edges are detected. (Khan et al., 2018)

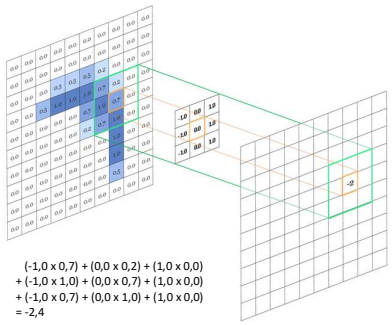

Figure 6. Computation logic of the filter in convolutional neural networks

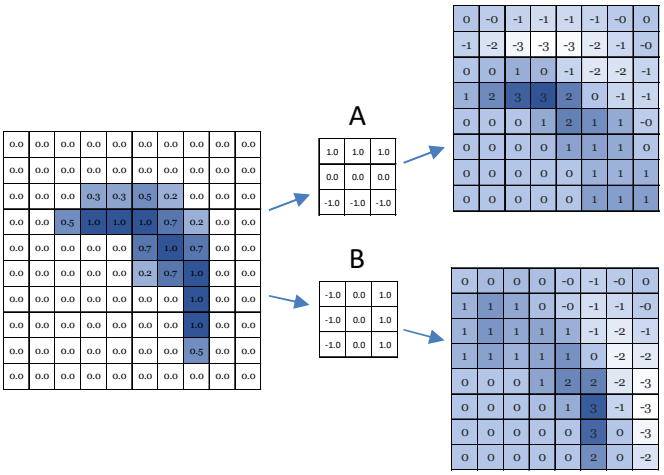

Figure 7. The function of different filters in convolutional neural networks.

The filter format is modified during the training of the network, so the format of the filters in the beginning is not the most important thing and usually, the initialization of the weights is random. (Khan et al., 2018; Zafar et al., 2018)

The learning process in a CNN is based on minimizing the prediction error, or loss. In this process usually two subprocesses are used, forward propagation and back propagation. In forward propagation, the network takes the input and moves through the network's all layers to produce the output. After this the loss is calculated, comparing the predicted result with the ground truth, thus evaluating the performance of the network. After this, the backpropagation is performed, where the impact of each weight on the loss is evaluated and the gradient is calculated with respect to these weights. This gradient shows the direction and how much the weight needs modifying to minimize the loss. An optimization function is then updating the weights based on these gradients, taking the defined learning rate into account. (Goodfellow et al., 2016; Zafar et al., 2018; Khan et al., 2018)

Many architectures in object detection, developed during the years use CNN in one form or the other. Yolov5, which will be described next, also makes extensive use of CNN in its own architecture.

3.3 Yolov5

Yolo, You Only Look Once, was released in 2015 by Joseph Redmon et al. It was a new approach to object detection as previous approaches tried to reuse classifiers, but Yolo look at it as a regression problem. The image is divided into parts in a grid-like format, and it predicts bounding boxes, confidence, and class probabilities for these parts, all in one go. (Redmon et al., 2015)

Redmon and Farhadi continued developing Yolo and they released two more versions before others took over. (Redmon & Farhadi, 2017; Redmon & Farhadi, 2018) Yolov4 was in turn released in 2019 by Alexey Bochkovskiy et al. (Bochkovskiy et al., 2020) Research in the subject continued and one company that jumped on the train was Ultralytics. They released their first version of Yolo in the summer of 2020. This version got the name Yolov5. This is the Yolo version that is used in this thesis. There are newer versions of Yolo out now and the most recent version, at the time of writing, is version 8 by Ultralytics.

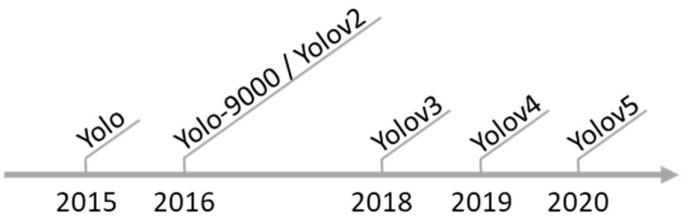

Figure 8. The timeline of the Yolo evolution up to version 5.

The early versions of Yolo, version one through to version four, were developed using the Darknet neural network framework. Since the release of Yolov5, the model has been built on the PyTorch framework. Both Darknet and PyTorch are platforms for building and training neural networks, while they contain the needed libraries and tools. The move to use PyTorch is one of the main advances that was made in Yolov5 compared to previous versions. PyTorch is written in Python and, with more users, makes it more approachable than Darknet which is written in C. PyTorch also makes Yolov5 easier to install and deploy. (Jacob Solawetz, 2020b)

Yolov5 is a pre-trained model. It is pre-trained on the COCO dataset, which contains over 200k labelled images. The Common Objects in Context dataset, or COCO is a large dataset provided by Microsoft. The advantage of using a pre-trained model is that you do not need to start from scratch and collect and label huge amounts of images. The pre-trained model is, as the name indicates, already trained to some extent and you do not need as many labelled images to train it for your specific application. Using a pre-trained model to train with your own dataset is called transfer learning.

Yolov5 is a one-step object detection model. This means that when Yolov5 is detecting an object in an image it is, at the same time, predicting its class and calculating its position. A one-step model has the advantage of being fast but on the downside, its accuracy may suffer.

3.3.1 The Yolov5 architecture and features

There are three main components in the Yolov5 architecture. They are called backbone, neck, and head. The backbone is the main part of the network, and it is where the features are detected. In the backbone architecture, a type of CNN structure is used, which has the purpose of facilitating the gradient flow through the network. The neck connects the different features and serves the head, which is the part generating the output. The head in Yolov5 is the same as in Yolov3. In Figure 9 is a graphical description of Yolov5. (Jocher & Waxmann, 2023)

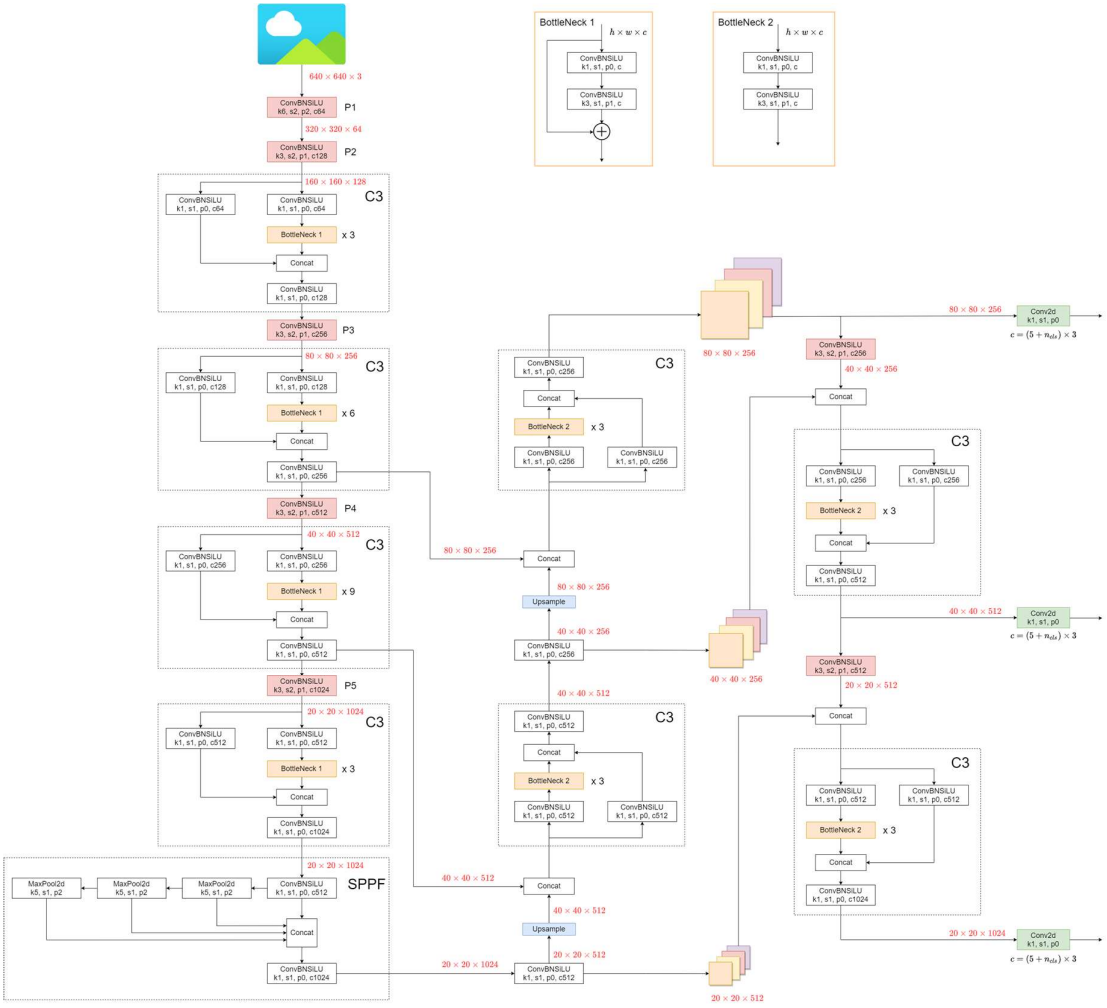

Figure 9. The Yolov5 v6.0 architecture (Jocher & Waxmann, 2023)

3.3.1.1 Grid cells

One feature that has lived on since the first Yolo version is grid cells. The image is divided into NxN equally shaped cells, called grid cells. The localization of objects and the prediction of class are calculated in relation to each grid cell. The predictions' probability and confidence values are also calculated for each cell. (Redmon et al., 2015)

Figure 10. The grid cells with black outlines are cells that are responsible for the localization and prediction of the staff gauge object.

After an object is detected in the image, a bounding box is predicted around it. The cells in which the object is located are identified and Yolov5 calculates the coordinates for these bounding boxes using regression (Jocher & Waxmann, 2023). The format of the vector that represents the bounding boxes is:

$$Y = (pc, bx, bh, bw, c1, c2, \dots) \tag{2.1}$$

where

pc is the probability value of the combined cells that contain the object, objectness score.

bx and ***by*** are the coordinates of the center of the bounding box in relation to the grid cell it is found in.

bh and ***bw*** is the height and width of the bounding box in relation to the grid cell it is found in and ***c1***, ***c2***... are the classes available.

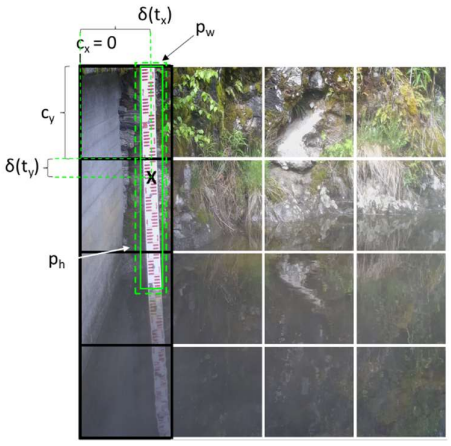

Figure 11. The prediction of a bounding box. The dashed lined box is the anchor box prior from the model parameters and the solid lined box is the prediction.

In Figure 11 above the black **X** is the center of the bounding box. The formulae to calculate ***bx***, ***by***, ***bh,*** and ***bw*** for the bounding box are:

$$b_x = (2 \cdot \delta(t_x) - 0{,}5) + c_x \tag{2.2}$$

$$b_y = (2 \cdot \delta(t_y) - 0{,}5) + c_y \tag{2.3}$$

$$b_h = p_h \cdot (2 \cdot \delta(t_h))^2 \tag{2.4}$$

$$b_w = p_w \cdot (2 \cdot \delta(t_w))^2 \tag{2.5}$$

where ***pw*** and ***ph*** is the anchor box prior.

tx and ***ty*** are normalized values of ***x*** and ***y*** coordinates by using the Sigmoid function. The center point offset range of the Sigmoid function is adjusted from (0,1) to (-0,5, 1,5). When the offset is used in this way, it allows the offset to easier become 0 or 1 (see Figure 12 below). (Jocher & Waxmann, 2023)

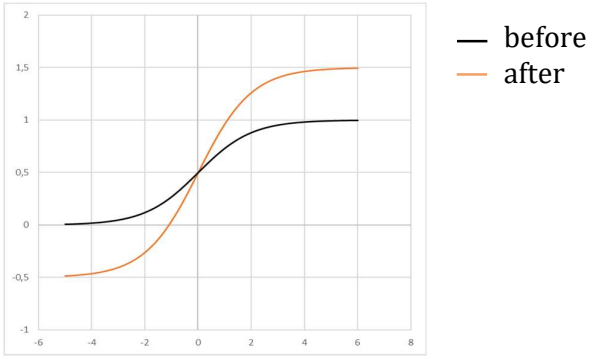

Figure 12. The normal sigmoid function is in black and the modified in red

3.3.1.2 Intersection over Union and non-maximum suppression

The objects often cover several grid cells and can therefore have multiple grid cell candidates for prediction. The idea of the intersection over union, or IoU in short, is to identify the relevant grid cells and only keep them. The user defines a threshold value for IoU as a parameter during inference. Yolov5 then calculates a value for each grid cell in which the object resides. In the case that the bounding box covers several grid cells, the threshold will be important. The value is the intersection area divided by the union area. If the value is less than the defined threshold, that grid cell is then ignored. This is called non-maximum suppression, which means that only those grid cells with the highest probability value are kept. IoU is illustrated in Figure 13 below. (Jocher & Waxmann, 2023)

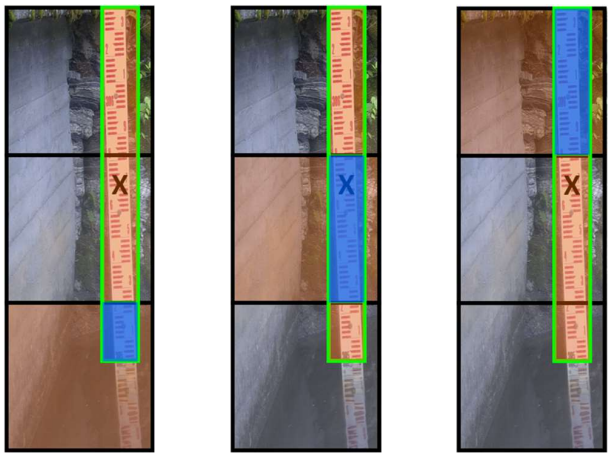

Figure 13. Intersection area in each grid cell illustrated. Intersection area is blue, and the union area is red and blue.

3.3.2 Build targets

Building targets is an important process for making training efficient and the model accurate. Yolov5 tries to localize the ground truth bounding boxes and assign them to a grid cell in the output map. The model scales the anchor box templates and tries to match the scaled versions to the ground truth. A limit of 4 is set for the ratio in the model. Larger scaled versions of the anchor box templates are not tested. The anchor bounding box template that totally surrounds the bounding box will then have the maximum ratio (Figure 14 below). (Jocher & Waxmann, 2023)

$$r_w = \frac{w_{gt}}{w_{at}} \tag{2.6}$$

$$r_h = \frac{h_{gt}}{h_{at}} \tag{2.7}$$

$$r_w^{max} = \max\left(r_w, \frac{1}{r_w}\right) \tag{2.8}$$

$$r_h^{max} = \max\left(r_h, \frac{1}{r_h}\right) \tag{2.9}$$

$$r^{max} = \max\left(r_w^{max}, r_h^{max}\right) \tag{2.10}$$

$$r^{max} < 4 \tag{2.11}$$

where

r is the ratio, ***w*** is width, ***h*** is height, ***gt*** is ground truth, and ***at*** is anchor bounding box template.

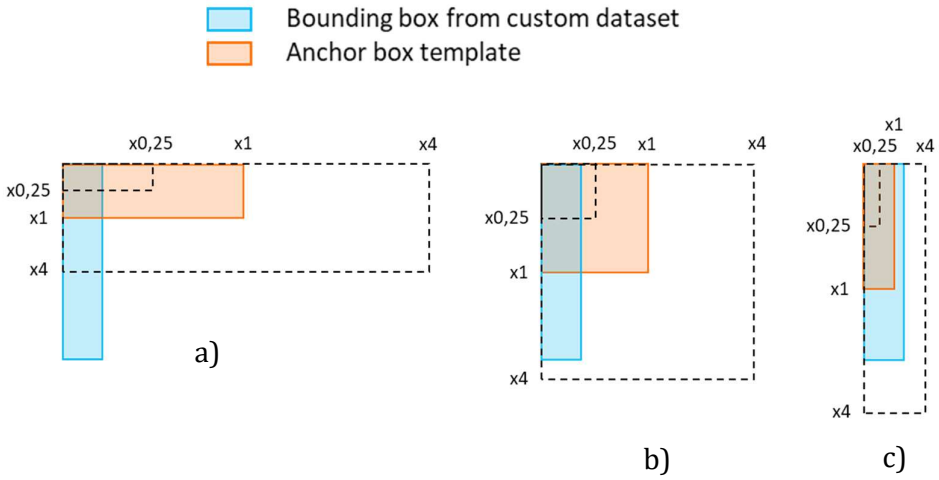

Figure 14. a) shows a failed matched anchor box template. Making the anchor box template smaller or larger will not cover the bounding box from the custom data set. Figures b) and c) shows successful matching. Enlarged anchor box templates will cover the bounding box from the custom data set. (Jocher & Waxmann, 2023)

3.3.3 Yolov5 labelling

To be able to train Yolov5 the images need to be labelled. This is easiest done by using a dedicated software. The software used in this thesis is called labelImg.exe (v1.8.1) (Lin, 2018). The more accurately the labelling is performed, the better. With this software, the object instances in each image are boxed to mark the exact location of the objects. Leaving space around the box will also lessen the accuracy of the final custom-trained model. The format of the label file is presented in Figure 15 below.

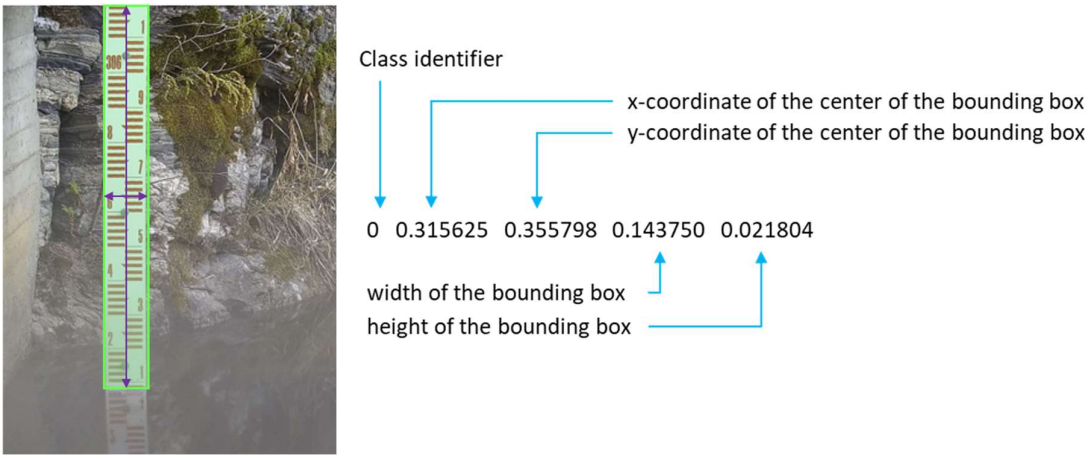

Figure 15. A staff gauge with a bounding box.

The x and y coordinates are normalized values of the image containing values from 0,0 to 1,0. The value 0,0 for the x-coordinate means a point at the leftmost edge of the image and 1,0 a point at the rightmost edge of the image. For the width and height, the value 0,5 would mean that the width is half of the width of the image or for height half of the height of the image. The label file is a text file (.txt) and it has the same name as the image file it belongs to. The labels of all objects in an image are stored in one label file.

3.3.4 Performance metrics

Yolov5 detects objects in images based on the knowledge it has acquired from the labelled images in the training dataset. To be able to measure the performance of the models, some metrics are needed. The basis for these metrics is intersection over union.

3.3.4.1 Intersection over Union

Intersection over union, or IoU in short, is a measure of how well the predicted bounding box matches the ground truth bounding box. IoU is calculated as the overlapping area between the bounding box and the object, divided by their union. If the IoU perfectly matches the ground truth the result is one and if the predicted bounding box is not overlapping the ground truth bounding box at all the result is zero. The IoU result can have values from zero to one. (Zafar et al., 2018)

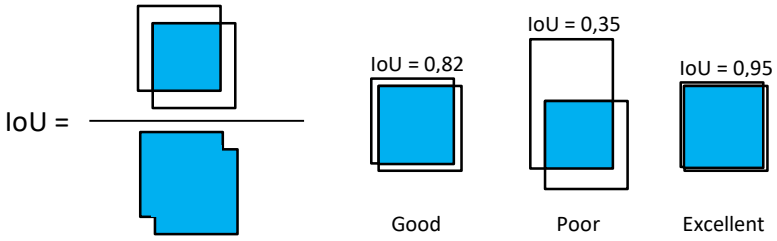

Figure 16. IoU and its performance illustrated.

3.3.4.2 Precision and recall

Precision and recall are two important metrics in machine learning. They are a measure of the model's ability to localize bounding boxes correctly based on the IoU. Precision metrics describe how many detections are correctly made and recall, or sensitivity means how

many actually correct detections are made (Padilla et al., 2021). These two metrics will have values between 0 and 1. These two metrics are used together and for a well-performing model, the values for the metrics must both be as high as possible.

To be able to calculate these values each detected bounding box must be defined as one of the following:

- True positive detection, or a correctly detected ground truth bounding box.
- False positive detection, or an incorrectly detected bounding box for an existing object, or detection of a wrong object.
- False negative detection, or a ground truth bounding box not detected.

The value for precision is calculated as follows:

$$Precision = \frac{True\ positive\ detections}{True\ positive\ detections + False\ positive\ detections}$$

The formula for calculating the recall value is the following:

$$Recall = \frac{True\ positive\ detections}{True\ positive\ detections + False\ negative\ detections}$$

3.3.4.3 Average precision

The average precision or AP is the precision averages across all values between zero and one. AP can also be described as the area under the precision-recall curve. The mean average precision or mAP is the average precision over one or multiple IoU thresholds. The metrics that are commonly used are mAP0,5 and mAP0,5:0,95. mAP0,5 corresponds to the average precision for an IoU with a threshold equal to 0,5 and averaged over all classes. The mAP0,5:0,95 metric corresponds to the average precision for IoU with thresholds between 0,5 and 0,95, averaged over all classes. The higher these values are the better the model is performing.

3.3.4.4 Loss functions

The loss is what the Yolov5 model is trying to minimize when training. In Yolov5 the loss calculations are divided into three parts: Classes loss, objectness loss, and location loss. (Jocher & Waxmann, 2023)

- Classes loss is a measure of the error in the classification of the objects.
- Objectness or confidence loss measures the error in the detection of objects in grid cells, that is whether there is an object located in a grid cell or not.
- Location loss calculates the localization error of an object in a grid cell.

All three loss functions are calculated as mean square errors and the errors are also affected by a defined constant or the IoU value between the prediction and ground truth. (Zafar et al., 2018)

The terms $\mathbb{1}_{ij}^{obj}$, $\mathbb{1}_{i}^{obj}$ and $\mathbb{1}_{ij}^{noobj}$, used in the equations below, are used to modify the loss in specific circumstances (Zafar et al., 2018). They will be explained together with their respective equation.

The equation for the entire loss function is as follows.

$$\text{loss}_{Yolov5} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \tag{2.12}$$

$$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right] \tag{2.13}$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left(C_i - \hat{C}_i\right)^2 \tag{2.14}$$

$$+\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} \left(C_i - \hat{C}_i\right)^2 \tag{2.15}$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{C \in classes} [(p_i(C) - \hat{p}_i(C))^2] \tag{2.16}$$

As mentioned earlier the loss function is divided into three parts and the first part of the equation is location loss. The term $\mathbb{1}_{ij}^{obj}$ in equations 2.17 and 2.18 will have the value 1 if the bounding box ***j*** has the highest IoU score and it matches cell ***i***, otherwise, the value will be 0.

$$location\ loss_{loc} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \quad (2.17)$$

where ***B*** is the number of bounding boxes and ***S2*** is the number of grid cells in the image. Further ***x*** and ***y*** are the coordinates of the predicted bounding box and $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$ are the coordinates of the ground truth data in the training dataset. $\boldsymbol{\lambda_{coord}}$ is a weight used for increasing the influence of bounding box location errors on the total loss. (Zafar et al., 2018)

A similar equation is also found for the width and height of the bounding boxes.

$$location\ loss_{dim} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{w}_i - \sqrt{\hat{w}}_i \right)^2 + \left(\sqrt{h}_i - \sqrt{\hat{h}}_i \right)^2 \right] \quad (2.18)$$

where ***w*** is the width and ***h*** is the height of the predicted bounding box. $\hat{\boldsymbol{w}}$ and $\hat{\boldsymbol{h}}$ is the width and height from the ground truth training dataset. To make the errors in small bounding boxes more important than small errors in large bounding boxes, a square root is added to the equation and the weight $\boldsymbol{\lambda_{coord}}$ is given the value 5, as default. (Zafar et al., 2018)

The next part of the loss function is dealing with the confidence score. The term $\mathbb{1}_{ij}^{noobj}$ in equation 2.19 will have the value 1 if the bounding box ***j*** has the highest IoU score but does not match cell ***i***, otherwise, the value will be 0.

$$confidence\ loss = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 \quad (2.19)$$

where $\boldsymbol{C}$ is the confidence score and $\hat{\boldsymbol{C}}$ is the confidence score for the predicted bounding box and the ground truth. To lessen the influence of this loss when no object is found the weight $\boldsymbol{\lambda_{coord}}$ is set to 0,5, by default. (Zafar et al., 2018)

Classification loss is the third part of the loss function. In equation 2.20 below the term $\mathbb{1}_i^{obj}$ will have the value 1 if cell ***i*** contains an object and if not, the value will be 0.

$$classification\ loss = \sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{C \in classes} (p_i(C) - \hat{p}_i(C))^2 \tag{2.20}$$

This loss is calculated as the error squared. As mentioned earlier the term $\mathbb{1}_i^{obj}$ is zero if no object exists in the grid cell ***i***. This means that the classification loss is not taken into account in terms of the entire loss function if no object is found. (Zafar et al., 2018)

3.3.5 Yolov5 configuration

Yolov5 is a well-constructed model that can be used as is, but a few parameters are worth addressing to improve the chances of getting better results. Below is a description of these parameters:

- batch size: defines how many samples are processed before the model's internal parameters are updated. It is recommended to set this parameter as large as the computer hardware allows. A larger value means faster training. (Jason Brownlee, 2022) The best value for this parameter can only be found by testing.
- epochs – the number of rounds of training through the whole dataset. During one epoch every image is analysed and contributes to the updated internal parameters of the model. One epoch usually contains more than one batch. The value for epochs needs to be sufficiently large so that the error from the model is low enough. The value is usually in the hundreds or thousands. (Jason Brownlee, 2022) When training with Yolov5 it is possible to continue training from the last round. The model stores both the best and the last custom-trained model, which makes it easy to continue training with more epochs.
- weights — file path containing initial weights, which means the pre-trained model.

- cache —Yolov5 cache images for faster training. This is highly recommended to use.
- img — image size in pixels (default — 640). Yolov5 has pre-trained models that accept 1280 pixel images. A larger image is more likely to improve the detection of smaller objects. The larger the image size mean the longer the training time and also the longer the inference time. It is recommended to use the same value for inference as was used in training.

Yolov5 is a pre-trained model which means that fewer images are needed to train a custom model than if training from scratch. But the same fact still stands that the more labelled images there are the better the training results will be. The more diverse images there are in the dataset the better. (Jocher & Waxmann, 2023)

The Yolov5 model comes in different sizes. Training with a larger model could improve the results, but the larger model used the slower the training and inference will be. Yolov5 model's different sizes are nano, small, medium, large, and extra large. Which model is which can be seen from the letter at the end of the model, for example, Yolov5m. The architecture of these models is the same, only the number of parameters increases with the size.

Evolving parameters, or optimizing parameters, is also a method that could improve the training results. Evolving parameters is a function that modifies the model parameters to better suit the custom dataset. Performing parameter evolution is a time-consuming procedure, but it might be worthwhile. (Jocher & Waxmann, 2023)

4 Method

Determining the water level from an image is a process consisting of several subtasks. The developed process divides the main part into 6 sub-tasks, each with its own objective. The objective of subtask number 5 is to find the water line in the image. In this developed process, there are two variants of this subtask. In one image processing is used and in the other one object detection and Yolov5. The work order is described in Figure 17.

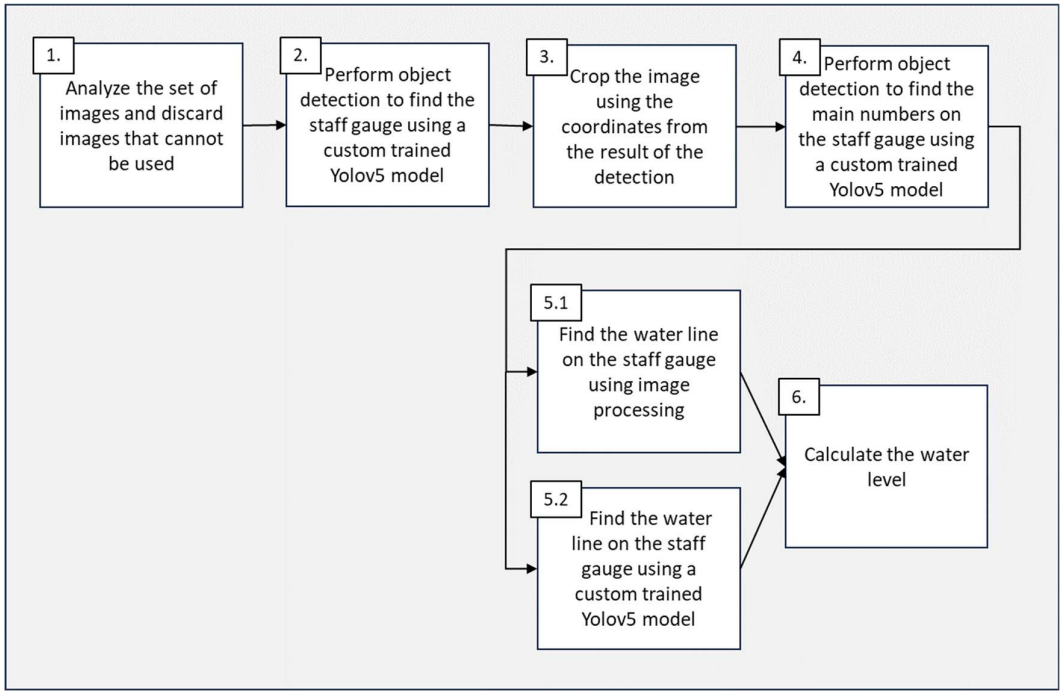

Figure 17. Workflow of the application versions

In this thesis, a machine learning model named Yolov5 is used to detect the staff gauge, the numbers on the staff gauge, and the water line. This model needs labelled images for training and the training needs to be supervised so that the best possible results can be achieved. The version of the model used is Yolov5m. Below, in Figure 18, is a diagram of the workflow for creating the datasets, training, and optimizing the model.

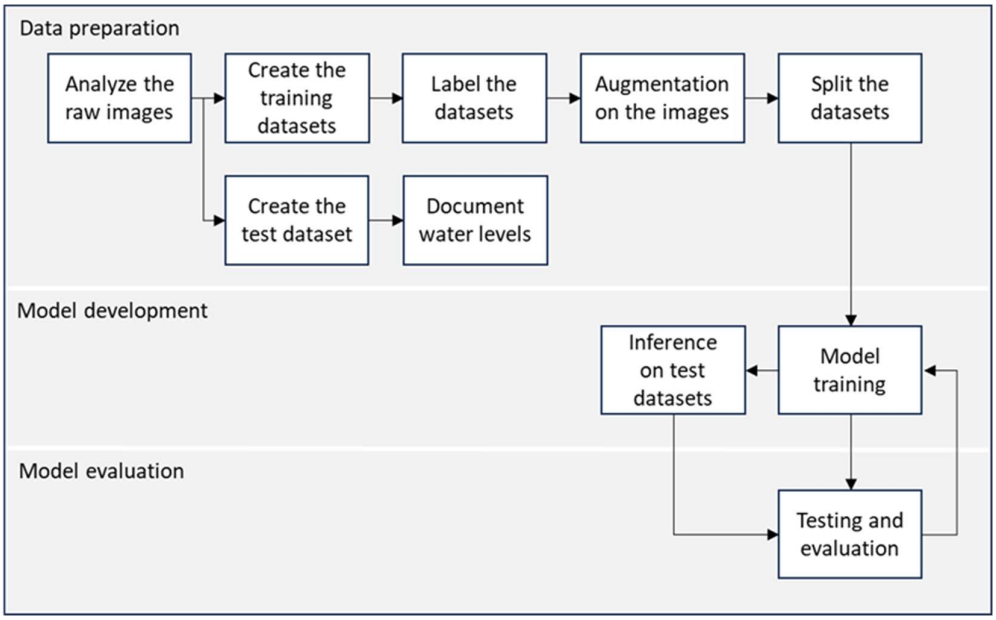

Figure 18. The work process for the use of the machine learning model Yolov5.

The code for the application is developed in Visual Studio Code using Python v3.10.7.

4.1 Image acquisition and the datasets

4.1.1 The camera

The location of the site is a dam for a hydropower plant in Norway. It has a tri-colored staff gauge mounted at the side of the dam near the actual dam construction. A camera is mounted on a pole attached to the dam wall. The camera used for capturing the images is a Hikvision DS-2CD2746G2T-IZS(C) with a 2,8-12mm zooming lens. The lens provides a 30-108 degrees field of view horizontally and a 17-56 degrees field of view vertically. According to Horenstein (Horenstein, 2005) a lens with a greater field of view of 100 degrees will cause a fisheye effect. The fisheye effect is when the image is distorted and the objects further from the center of the image become curved (see Figure 19 below).

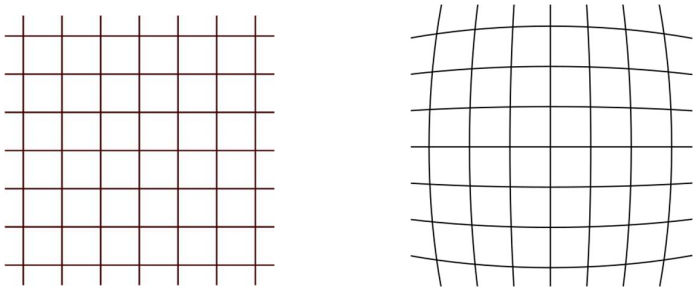

Figure 19. Mesh illustrating the distorted image a fisheye effect can give. To the left is no distortion and to the right there is fisheye distortion.

The camera is equipped with infrared light to illuminate the field of view during low-light situations. It also has an IR-cut filter, which is a filter that filters out infrared light during the daytime and is automatically swivelled out during night-time to allow the camera to capture images during the night using infrared light.

The maximum resolution is 2688x1520 pixels which means 4MP. The camera captures images every 10 minutes, all year round.

4.1.2 The datasets

There are three different object detections to be performed. These are in task 2, in task 4, and in task 5.2. Therefore, three different datasets need to be prepared.

4.1.2.1 The dataset for the staff gauge detection

The dataset for training and validation is manually collected from a batch of 52610 images. The conditions at the location vary a lot, so a wide variety of images were chosen to form the first dataset, for the detection of the staff gauge. These images were chosen to best cover all different weather and illumination conditions as possible. To increase the number of images in the dataset, image augmentation was performed. The Yolov5 model performs its own augmentation on the dataset as well. The augmentation it carries out is random affine, MixUp, colour space adjustments, copy and paste, advanced mosaic augmentation, and albumentations. (Jocher & Waxmann, 2023)

The image files are then split into two parts, one part for training and one part for validation. These files are saved in the following folder structure in a Yolov5 subfolder, as illustrated in Figure 20.

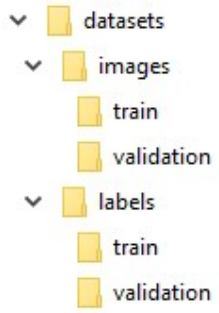

Figure 20. Folder structure for the Yolov5 training dataset.

For the staff gauge dataset, 1253 images were picked out with as great a variety as possible. These images were then labelled. To enlarge the dataset, some of the images were augmented by darkening, brightening, red-tinting, blue-tinting the images and by combining different patterns of dark and light areas in the images. The total number of augmented images was 963. 90 background images were also added. The background images did not contain any staff gauge and these images were, of course, not labelled either. The total number of images that were used in the staff gauge detection training was

2306. These were then divided into training and validation images in a ratio of 83% training and 17% validation.

4.1.2.2 The dataset for the numbers and water line detection

The samples for the dataset for *the numbers* object detection were taken from the same dataset that was used for the staff gauge detection. Images, where the numbers on the staff gauge were not clearly visible, were omitted and the raw dataset for the number detection consisted of 549 images. These images were then cropped using the software Gimp 2 and rotated to make the staff gauge close to vertical. These images were then augmented similarly as the staff gauge was, to make the dataset larger for training. The total number of images for the number training was 5904 and they were divided into 4666 training and 1238 validation images. 21% of the images were validation images.

For the water line detection training, the same 549 cropped images that were used for the number training, label file inclusive. The water line was labelled so that the height of the bounding box was approximately 4 cm, with the water line in the middle of the box. The width of the bounding box was slightly wider than the staff gauge, approximately 1 cm on each side of the staff gauge. This way the features next to the staff gauge at the water line are also captured (see Figure 22 below). To save time in labelling, the water line was labelled for one image. This fourth class was then copied to all the existing label files. This way, the labelled images only needed adjustment of the bounding box location for the fourth class in the labelling software instead of going through each image and adding the fourth class from scratch. These images were also augmented the same way as the other datasets making a total of 5868 images. These images were divided into 67% training and 33% validation images. In Figures 21 and 22 examples of labelled images are presented.

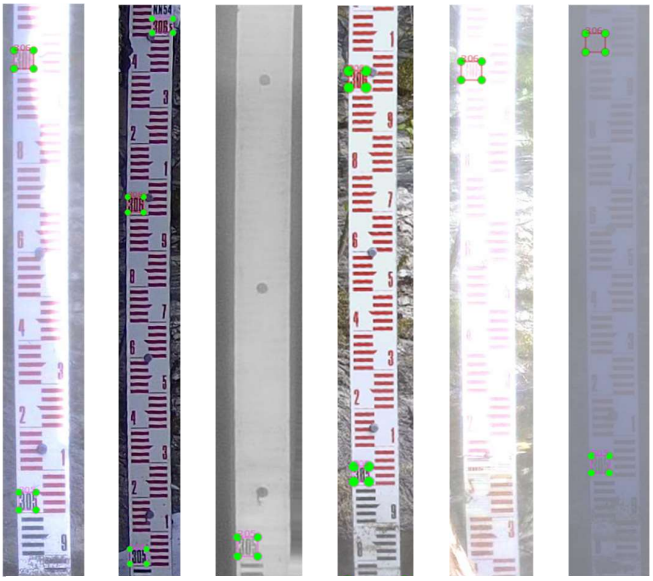

Figure 21. Examples of staff gauges with labelled numbers.

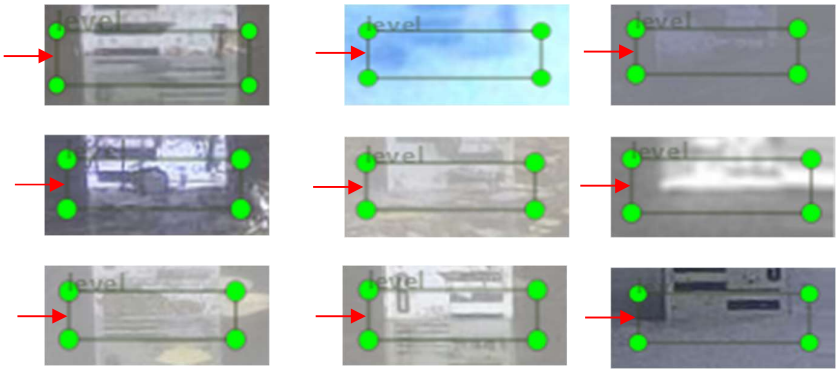

Figure 22. Close-up images of water line labelled on staff gauge. The red arrow marks the center of the bounding box, and the center line would be returned as the water line from the detection. The red arrow is just added for clarification, it is not part of the labelling.

4.1.2.3 The test dataset

Finally, a test dataset was put together. For this dataset, all the images that the trained models should be able to detect on were visually checked. To make the selection process more objective, a Python code was constructed to gather images from between May and November, from the following hours 03, 06, 09, 12, 15, 18, and 21. Choosing images from around the clock would ensure different lighting conditions in the captured images and that the water level would vary if daily fluctuations occurred due to energy production.

The images are captured every 10 minutes, which means that there are 6 images captured every hour, which in turn means approximately 42 images from every day within the time interval described above. Before collecting the test images, the blurry images and the night-time images were removed using the filtering code, described in chapter 4.1.6. This means that there could be less than 42 images per day. The images were then manually checked, and images were removed to get a final count of one thousand test images.

The test images were also verified so that none of them would also be included in the training datasets. To do this an application was created in Python, which calculated the SHA1 hash for both the selected test images and the images used in training.

These images were then visually inspected, and the water level was documented to the nearest cm.

4.1.3 The data analysis

The training was done on a PC with an AMD Ryzen 9 5900x processor, 32GB of RAM, and a NVIDIA RTX 3080 graphics card. The operating system used was Ubuntu 22.04 LTS which was run from a USB drive. The GPU was used in the training and the CUDA version installed was 11.7. The software versions used were in accordance with, at the time, current requirements for Yolov5.

Training started with the Yolov5 s model for all three detection parts and the number of epochs was set to 2000.

4.1.4 Staff gauge detection training

The first training, the training for the staff gauge detection used 2306 images in total. 1920 of these images were used for the actual training and 386 were validation images. The batch size was set to 20 and the number of epochs was 2000. All the other parameters were left as default. The best model is saved, as default in a specific folder.

4.1.5 *The numbers* detection training

The numbers training or the training for the detection of the numbers on the staff gauge used 4666 images for training and 1238 images for validation. As in the training for detection of the staff gauge, batch size was set to 20 and epochs to 2000. No other parameters were altered here either. The best model was also saved here in a specific folder.

An attempt to improve the training results was made by first optimizing the hyper-parameters using the evolve parameter's function. The evolving was done for 10 epochs and 300 generations of optimizing. After the optimization was performed, the training was done again using the same parameters as before except for the new optimized hyper-parameters. This optimization was performed on the dataset that includes the water line class.

Hyperparameters	default	optimized
lr0:	0,01	0,01082
lrf:	0,01	0,01
momentum:	0,937	0,98
weight_decay:	0,0005	0,00077
warmup_epochs:	3	1,7103
warmup_momentum:	0,8	0,9495
warmup_bias_lr:	0,1	0,08293
box:	0,05	0,07744
cls:	0,5	0,45676
cls_pw:	1	1,0538
obj:	1	1,4496
obj_pw:	1	0,89871
iou_t:	0,2	0,2
anchor_t:	4	5,6574
fl_gamma:	0	0
hsv_h:	0,015	0,01063
hsv_s:	0,7	0,60342
hsv_v:	0,4	0,29999
degrees:	0	0
translate:	0,1	0,07852
scale:	0,5	0,19589
shear:	0	0
perspective:	0	0
flipud:	0	0
fliplr:	0,5	0,5
mosaic:	1	1
mixup:	0	0
copy_paste:	0	0
anchors:		3,805

Table 1. Yolov5 hyper-parameters. To the left are default parameters and to the right are the same parameters optimized.

For a more thorough description of the separate parameters, please refer to the documentation at Ultralytics. (Jocher & Waxmann, 2023)

4.1.5.1 Water line detection training

The final training for the detection of the water line used 3912 images for the training and 1956 images for the validation. The same settings were used here as in the previous training sessions, batch size 20 and 2000 epochs. Like in the other training runs, the best model for the water line detection could be found in a specific folder.

To try to improve the training results an optimization was performed. This optimization is described in section 3.3.5 as the same dataset was used for both *the numbers* and water line detection training.

4.1.6 Task 1 – Analyse the set of images and discard images that cannot be used

For this thesis, there was a dataset of 52610 images to start with. These images are, as mentioned before, captured every 10 minutes day and night, which means that the range of quality of the images varies quite a lot. The camera is mounted in a fixed position, but the zooming factor can be changed. This was usually done little depending on the water level at that time. The image composition was fairly consistent throughout the sample set.

Yolov5 can utilize a range of image qualities depending on the task. Determining the water level using Yolov5 requires relatively high image quality due to the subtle characteristics of the water line. The images need to have some sharpness and contrast to be able to find the edge that represents the water line on the staff gauge. Training with low-quality images does not necessarily improve model performance. It might in fact decrease the overall performance of the model. (Dodge & Karam, 2016)

Sabbatini et al. used a method for filtering out unwanted images (Sabbatini et al., 2021). In this method, they calculated the image:

- Root mean square of image saturation channel
- The maximum difference between pixel intensities of all colour channels
- Mean value of pixels of all colour channels

No other pre-filtering out images were done in the review of articles done for this thesis, but many authors pointed out the importance of image quality (Sabbatini et al., 2021; Zhang, Zhou, Liu, Zhang, et al., 2019; Zhang, Zhou, Liu, & Gao, 2019; Qiao et al., 2022). When capturing images outdoors you will always be at the mercy of the elements. Poor illumination, rain, hail, snow, fog, bright sunlight, shadows, dirt, and debris are all things that influence the quality of the image. It is difficult if not impossible to prepare for all circumstances. In this thesis, a modified version of the one described by Sabbatini et al. (Sabbatini et al., 2021), is used.

There are two categories of images that needed filtering out. The first category is night-time images, as the numbers on the staff gauge are poorly visible in these images. The second type is blurred images. A blurred image will not have the sharper edges that are needed for finding the water level.

Night-time images are usually grayscale with a strong leaning towards black in the histogram. The contrast in these images is quite low and they mostly occur during the winter season. In these images the numbers are very difficult to read, so they are therefore discarded. To identify a night-time image the following values were calculated:

- Root mean square of the image saturation
- Maximum inter-pixel intensity difference
- Mean intensity value of all colour channels

To calculate the root mean square of an image saturation, the following formula is used:

$$s_{rms} = \sqrt{\frac{1}{n}(s_1^2 + s_2^2 + \cdots + s_n^2)} \tag{3.1}$$

where ***n*** is the number of pixels, or pixel saturation values and ***s*** is the saturation.

The maximum inter-pixel intensity difference is calculated by taking the value of the pixel with the highest intensity in the image and subtracting the value of the pixel with the lowest intensity in the image.

The mean intensity value of all colour channels is calculated using the following formula:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{3.2}$$

where ***n*** is the number of pixels and ***x*** is the intensity.

The tree values described above were calculated for each image and the threshold values used in the code were found experimentally and differ slightly from the values Sabbatini et al. used (Sabbatini et al., 2021). The three threshold values are listed in Table 2 below.

Root mean square of the image saturation	< 0,03
Maximum inter-pixel intensity difference	> 190
Mean intensity value of all colour channels	< 170

Table 2. The thresholds for images being night-time.

Some images were out of focus and therefore blurred and with low contrast. These kinds of images cannot be enhanced sufficiently, so these images also need filtering out. To

identify a blurred image, the fact that a blurred image has fewer marked edges is used. In a sharp image, the general adjacent pixels intensity difference will be bigger than in an unsharp image.

The Sobel operator is well-known when it comes to edge detection. When applied to an image, it returns a modified image with accentuated edges. Calculating the Sobel operator on both the x-axis and y-axis and then calculating the average value of the magnitudes gives an idea of the occurrence of edges in the image. Using this function alone as a filtering function will unfortunately also filter out such images that have poor contrast but could still be used. Using the average value of the Sobel magnitudes together with the standard deviation of the Dark channel prior proved to work better.

The Dark channel prior is mainly used for dehazing images. It is based on the statistical fact that non-sky local regions in haze-free outdoor images have some pixels with very low intensity in at least one colour channel. On the other hand, in hazy images, these low-intensity pixels will have higher intensities due to airlight. (He et al., 2009) Using the standard deviation of the dark channel prior of an image in blur detection is somewhat controversial, but it was experimentally found to work fairly well together with the Sobel magnitudes described above and therefore it will be included in the procedure.

To reduce the computation time, the calculations were not performed on the whole image. The staff gauge is mainly found in the upper third of the left half of the images. The above calculations are therefore only done on this region.

The threshold value used for the average gradient magnitudes is 24,5 and for the standard deviation of the dark channel variance 0,08. These threshold values were found experimentally.

4.1.7 Task 2 – Perform object detection to find the staff gauge

The staff gauge is detected using a Yolov5 model. For the inference, the IoU parameter was set to 0,90 and the confidence value to 0,10. The location of the bounding box for the detected staff gauge is stored for use in task 3.

4.1.8 Task 3 – Crop the image using the coordinates from the result of the detection

The staff gauge detection detects the staff gauge in the image and the model returns the coordinates of the bounding box for the object (the staff gauge). This bounding box will not necessarily contain the water line, so the cropping area needs to be enlarged. During testing, it was found that enlarging the cropping area by 300 pixels along the y-axis downwards and 50 pixels along the x-axis to the right, was a suitable enlargement. With this larger area, it could be made sure that the water line will be located within. The image was cropped according to these coordinates.

The detected staff gauge is not necessarily perfectly vertical and it might be tilted. To check for this a code snippet was created. The image is processed with the following functions. The image is:

- converted to grayscale, using OpenCV.
- blurred to remove noise, using Gaussian blur from the OpenCV library.
- binarized to enhance the contrast, using Threshold and Otsu from the OpenCV library.
- eroded to remove small artefacts, using Erode from the OpenCV library.
- edge detected using Canny, from the OpenCV library.

The Hough transform is finally performed to be able to reveal a possible rotation angle. The image is then rotated using this angle to get an almost vertical image of the staff gauge.

4.1.9 Task 4 – Perform object detection to find the main numbers on the staff gauge

A Yolov5 model is run to detect the numbers on the staff gauge. The parameters IoU and confidence value are set to 0,50 and 0,85 respectively. The locations for the bounding boxes for the numbers are stored and used by the next task.

4.1.9.1 Task 5.1 – Find the water line using image processing

The first of two methods for finding the water line in the images is using image processing. As a first step, the image is converted to grayscale using a function from the Open CV library in Python. This colour conversion function uses the format described above in theory chapter 2. After that, the image was blurred using an OpenCV function called

“bilateralFilter”. The blurring function removes small imperfections in the image. Then the image was binarized using an OpenCV function called Otsu and finally, all small areas in the image were filled using the function “binary_fill_holes” from the Scipy library.

This binarized, black-and-white image is then looped through line by line starting from the 100th last row and moving upwards. The reason for starting at the 100th row is that on the last rows, there are usually a lot of disturbances that would otherwise interfere. During the looping, the number of black pixels on each row was counted and when there were less than 90% black pixels on five consecutive rows, the first of these rows is then defined as the location of the water line.

4.1.9.2 Task 5.2 – Find the water line using object detection and Yolov5

Object detection for the water line on the cropped staff gauge is performed with IoU parameter 0,85 and confidence value 0,01. If a bounding box is found, the horizontal center line of the bounding box is defined as the water line. If no water line is detected, the function returns the water level 0,0.

4.1.10 Task 6 – Calculate the water level

The numbers detection done in task 4, will give the location of the detected numbers on the staff gauge. The location furthest down should be the number 305, but the classification is not necessarily correct. In the code section that calculates the water level, a verification of the detection is included.

The first step is to verify if any bounding box has been detected. If no bounding box is detected, this function returns 0,0 as the water level. If at least one bounding box is detected, the following checks are performed.

The checks are in three parts:

1. If the midpoint of the bounding box is found to the right in the cropped image of the staff gauge, it must be number 306,5.

2. If the midpoint of the bounding box is found to the left in the cropped image of the staff gauge and if it is located in the top half of the cropped image, it must be the number 306.

3. If the midpoint of the bounding box is found to the left and is located in the lower half of the cropped image of the staff gauge, it must be the number 305.

The checks described above are done on all detected bounding boxes. If duplicates are found for checks 1 or 2, this is noted as an error. If a duplicate is found on check 3, the location of these two bounding boxes that is the furthest down, is discarded.

The next step is to define the scale of the image and calculate the distance between the water line and a reference point. The scale of the image is calculated using the location of the bounding boxes that are the furthest apart. For example, if the bounding boxes for 306,5 and 305 were found, their distance is 150 cm. 150 is then divided by the pixel difference of the two bounding boxes in the image. If only one bounding box is detected, its height is the base for the scale. The actual height of a bounding box is approximately 4 cm and all bounding boxes have the same height. Four is then divided by the height of the detected bounding box in the image, in pixels, to get the scale cm/pixels. The water level is then calculated as the distance between the water line and the detected bounding box located the furthest down in the image using the scale described above. If the calculations fail, 0,0 is recorded as the water level. An example of labelled numbers is shown in Figure 23.

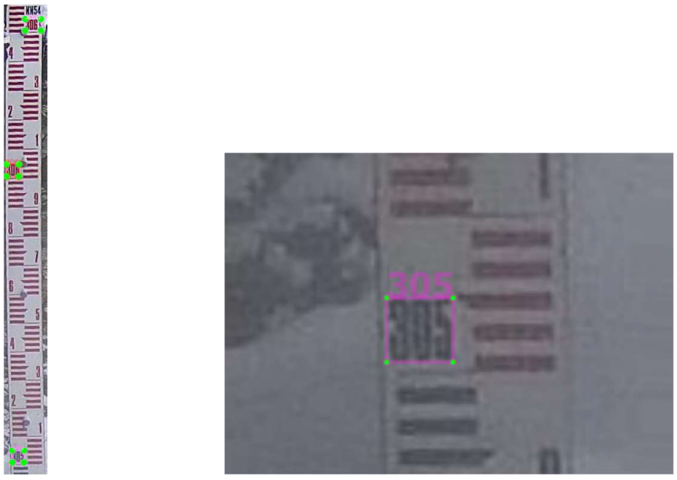

Figure 23. To the left, the staff gauge with labelled numbers, and to the right one labelled number in close up.

4.1.11 Interpretation of results

To better be able to compare the results between the two applications, it was decided that the results are divided into four groups depending on their prediction error. The accuracy of the visually documented measurements on the test images is ±0,5 cm and thereby the first group contains the results with a prediction error within 0,5 cm. The intervals for the rest of the groups are based on experimental results.

The groups are:

- Result with a prediction error within 0,5 cm.
- Result with a prediction error within 3 cm.
- Result with a prediction error between 3 and 30 cm.
- Result with a prediction error greater than 30 cm.

The group with a prediction error within 0,5 cm is defined as a correct result.

5 Results

The results are presented in different chapters. In the first chapter, the results of the first task are displayed. In the second chapter, the results of all the Yolov5 model training sessions are presented and in the last two chapters, the results of the two application versions are presented respectively.

The task of reading out the water level from captured images was divided into six sub-tasks. This program contains all, but the first sub-tasks and reads out the water level from all images in a folder, in one run. Two versions of the application were developed and compared. Both image processing and object detection with Yolov5 were used to find the water line. The first sub-task is the image selection.

5.1 Task 1 – Analyse the set of images and discard images that cannot be used

The first task was to filter out the blurry images and night-time images. The number of images before the filtering was 52610 images and 22329 images defined as night-time images and 1958 images were defined as blurred images. This means that the remaining 28323 images passed through. The filtering procedure takes quite a long time. It took roughly two hours on the testing PC, an i5-12400 CPU PC.

There were a few images that were falsely not filtered out. How many images that were falsely filtered out or not filtered out is difficult to establish, because there is no exact definition nor a measure for a night image or a blurred image. Below in Figures 24 – 26 are examples of images that were picked up as blurry, as night-time images, or as not filtered out by the filtering code.

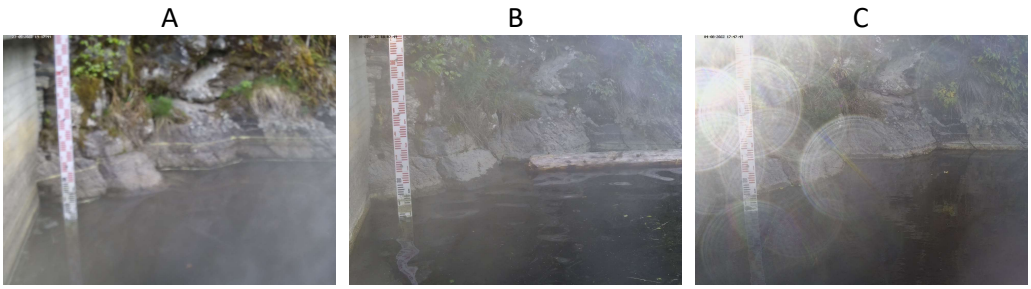

Figure 24. Images defined as blurred by the filtering code in task one.

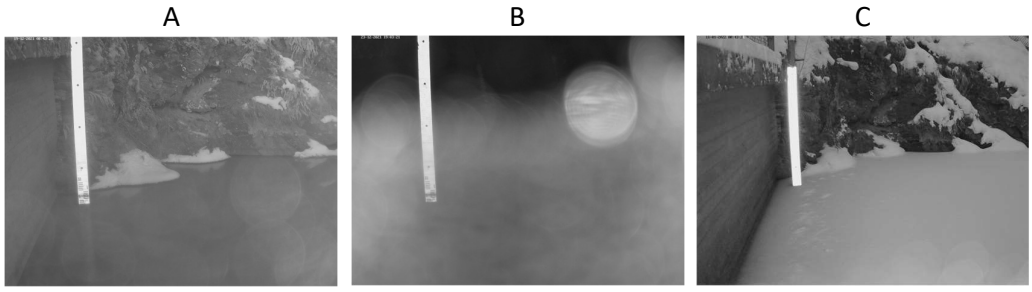

Figure 25. Three images filtered out as night-time images by the filtering code in task one.

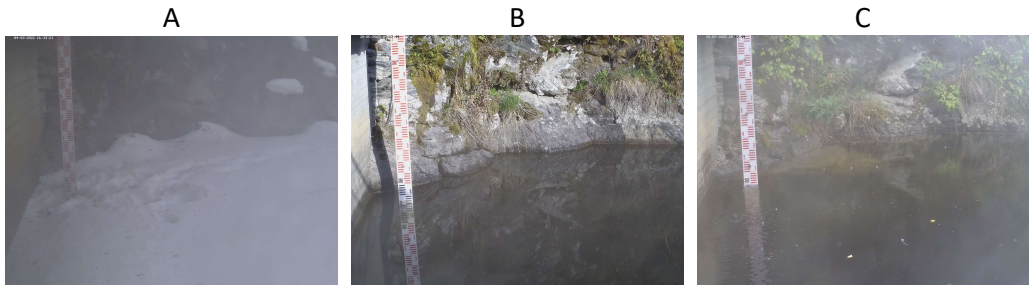

Figure 26. Example of images that passed the filtering in task one.

5.2 Results of training the Yolov5 models

The training of the Yolov5 was done three times, with different aims and with somewhat different datasets. The methods are described in chapter 3.

The first Yolov5 training was training the staff gauge detection model. The training took just over 4 hours, and it needed 318 epochs before it stopped training due to no improvements for the last 100 epochs. The best model was achieved after 218 epochs. This model contains 212 layers, 20852934 parameters, and no gradients. The training had a steep learning curve and the final model had a precision of 0,994, a recall of 1, a mAP0,5 value of 0,993, and a mAP0,5:0,95 value of 0,98. The result is presented in the curves in Figure 27 below.

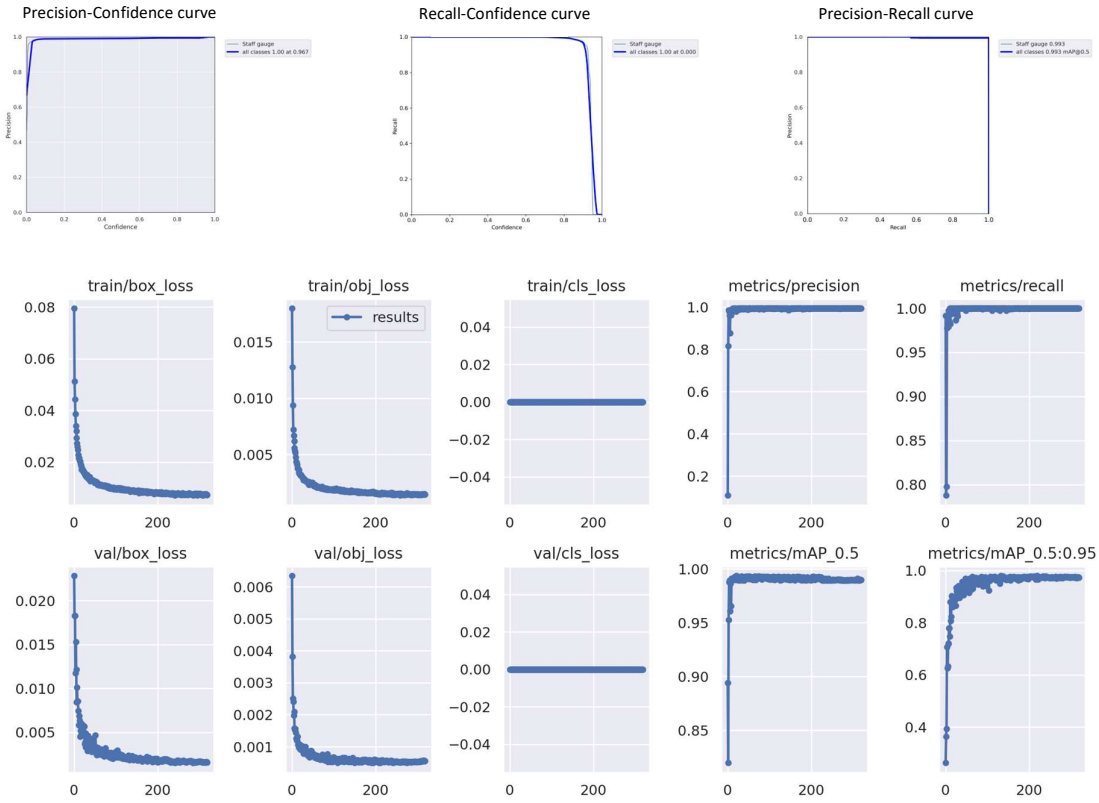

Figure 27. Results of the staff gauge detection training.

The second training concerned *the numbers* detection model. This training was done as two separate ones. The first one was done straight on the dataset with default hyper-

parameters and the second one was done using optimized hyper-parameters, described in chapter 3.3.5.

This training took a little longer than the one for the staff gauge. Using default hyper-parameters, it completed 1220 epochs in 22 hours and 40 minutes, and using optimized hyper-parameters it completed 1335 epochs in 23 hours and 54 minutes. The best model was found at epochs 1112 and 1235 respectively. *The numbers* model contains 212 layers, 20861016 parameters, and no gradients and the combined model including the water line class contains 212 layers,20877180 parameters, and no gradients. Below, Table 3 is showing the results per class. In the training using optimized hyper-parameters the water line class was also included. This training was joint for both *the numbers* detection and the water line detection. The rest of the training results are shown in Figures 28-29 below.

Class	Precision	Recall	mAP0,5	mAP0,5:0,95
All	1	1	0,995	0,987
306	1	1	0,995	0,991
305	1	1	0,995	0,979
306,5	1	1	0,995	0,992

Class	Precision	Recall	mAP0,5	mAP0,5:0,95
All	1	1	0,995	0,977
306	1	1	0,995	0,972
305	1	0,999	0,995	0,973
306,5	1	1	0,995	0,992
level	0,999	1	0,995	0,97

Table 3. Results of training of *the numbers* detection model. To the left is trained with default hyper-parameters and the table to the right is trained with optimized hyper-parameters. The water line class was included in the training with optimized hyper-parameters.

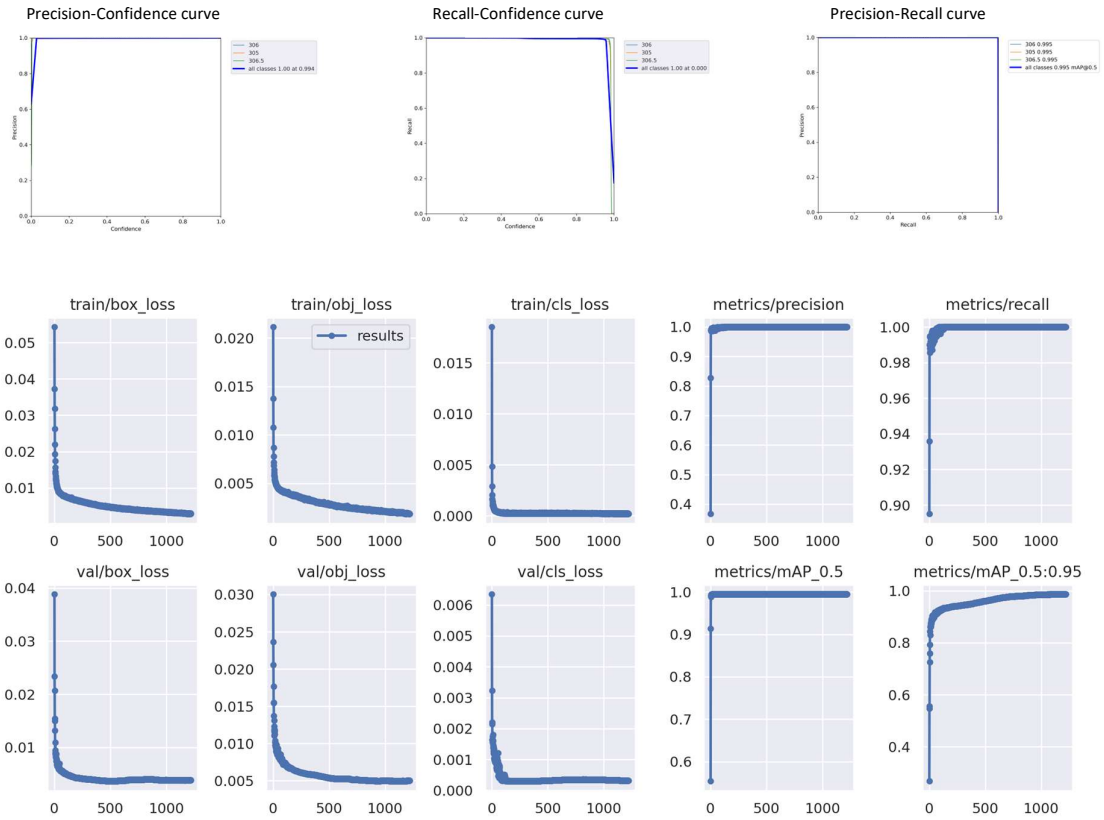

Figure 28. Results of the training of *the numbers* detection model with default hyperparameters.

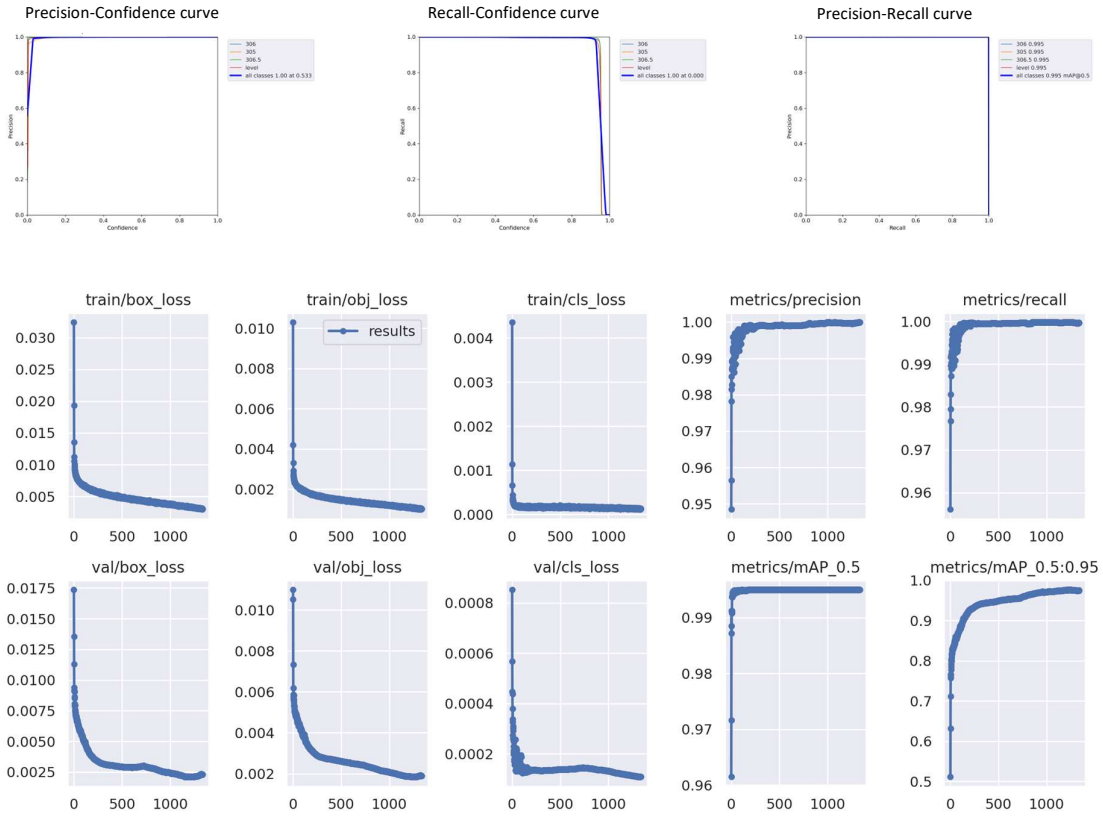

Figure 29. Further results of training of *the numbers* detection model with optimized hyper-parameters.

The third training was the training of the water line detection model. This training took the longest. It was also done as two separate training runs. The first run was done using default hyper-parameters and the second run was using optimized hyper-parameters, described in chapter 3.1.5. The training with optimized hyper-parameters was joint with *the numbers* training. It took 30 hours and 23 hours and 54 minutes respectively and encompassed 1711 and 1335 epochs. The best model was achieved 100 epochs earlier, at 1610 and 1235 epochs respectively. The results of the training runs are shown in Table 4 and Figure 30 below.

Class	Precision	Recall	mAP0,5	mAP0,5:0,95
level	0,999	1	0,995	0,949

Class	Precision	Recall	mAP0,5	mAP0,5:0,95
level	0,999	1	0,995	0,97

Table 4. Results of training of water line detection model. To the left is trained with default hyper-parameters and the table to the right is trained with optimized hyper-parameters.

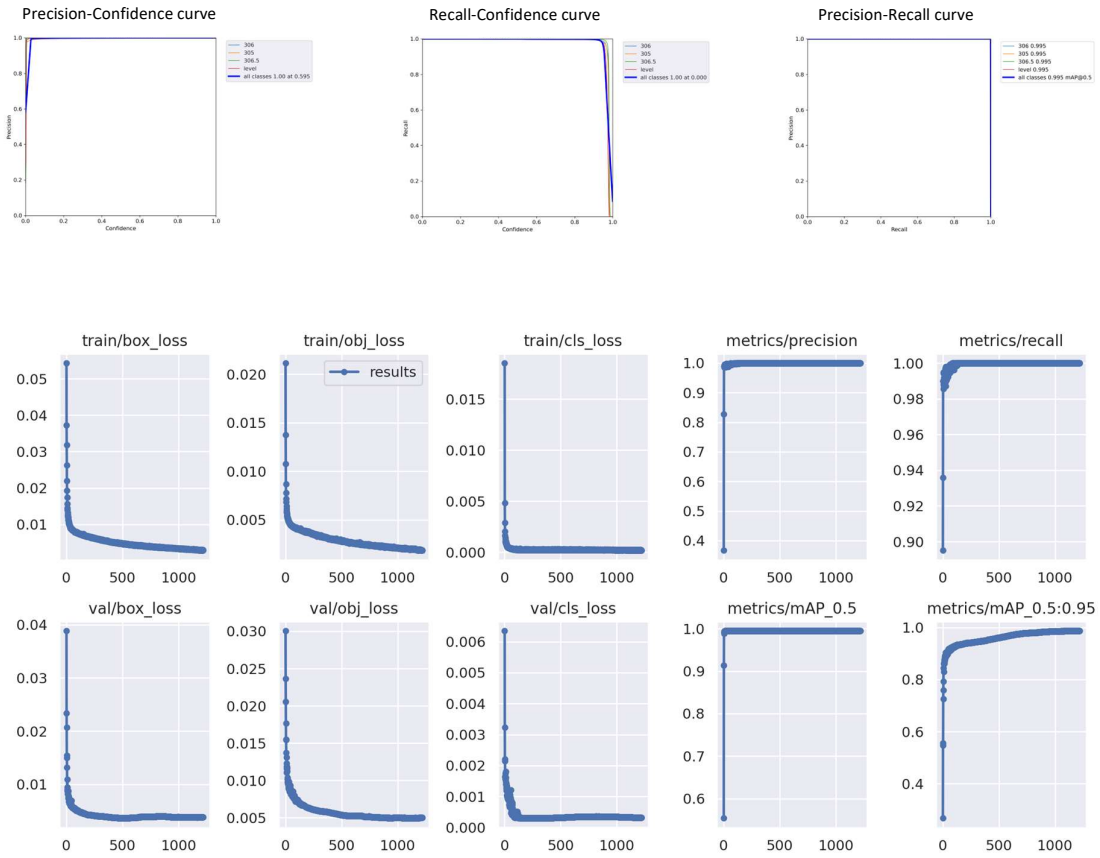

Figure 30. Results of the water line training using default hyper-parameters.

Further results using optimized hyper-parameters are found in Figure 29 above.

5.3 Application version 1 – Using image processing to find water line

Two different runs were done for the application version 1, for comparison. The first run was using *the numbers* model trained with default hyper-parameters and the second run was using the model trained with optimized hyper-parameters. The results of both runs are found in Table 5 and Figures 31-32 below.

	Run 1		Run 2	
Total number of images:	**1000**		**1000**	
Number of images where staff gauge not detected:	**0**		**0**	
Number of images where numbers not detected:	**14**	1,4 %	**11**	1,1 %
Number of images with > 30cm difference:	**129**	13 %	**170**	17 %
Number of images between 3cm and 30cm difference:	**254**	26 %	**222**	22 %
Number of images with < 3cm difference:	**617**	63 %	**608**	61 %
Number of images correctly estimated, within ±0,5cm:	**255**	26 %	**143**	14 %

Table 5. Results of the image processing application. In run 1 default hyper-parameters were used for Yolov5 and in run 2 optimized hyper-parameters were used. The percentage values are the percentage of the 1000 testing images minus the images where the numbers were not detected.

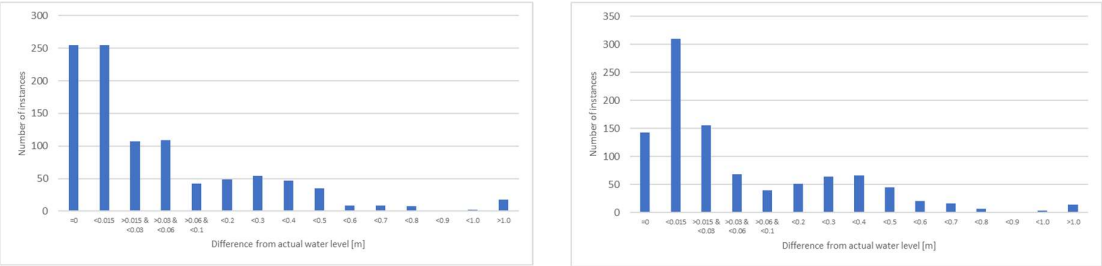

Figure 31. Results of run 1 (to the left) and run 2 (to the right), presented as occurrences of water level estimation in intervals.

Figure 32. Results of run 1 and run 2. The chart above: Both runs presented as occurrences of water level estimation in intervals. The chart below: Both runs presented as cumulative occurrences of water level estimation.

5.4 Application version 2 – Using Yolov5 to find the water line

Three runs were done for application version 2, to find the best detection model combination. The first run used the best performing separate model for *the numbers* detection and the first and separate training session for the water line detection. The second run used the combined training with optimized hyper-parameters for both *the numbers* and water line detection and the third run used the separate best-performing model for *the numbers* detection and the combined trained model with optimized hyper-parameters for the water line detection. The same model for staff gauge detection was used in all three runs.

The results of the three runs are presented in Table 6 and Figures 33-34 below.

	Run 1		**Run 2**		**Run 3**	
Total number of images:	**1000**		**1000**		**1000**	
Number of images where staff gauge not detected:	**0**		**0**		**0**	
Number of images where numbers not detected:	**14**	1,4 %	**11**	1,1 %	**14**	1,4 %
Number of images where water line not detected:	**15**	1,5 %	**18**	1,8 %	**21**	2,1 %
Number of images with > 30cm difference:	**18**	2 %	**72**	7 %	**22**	2 %
Number of images with > 30cm difference where both numbers and water line detected:	**3**	0,3 %	**54**	5 %	**1**	0,1 %
Number of images between 3cm and 30cm difference:	**160**	16 %	**124**	13 %	**159**	16 %
Number of images with < 3cm difference:	**822**	83 %	**804**	81 %	**819**	83 %
Number of images correctly estimated, within ±0,5cm:	**368**	37 %	**247**	25 %	**398**	40 %

Table 6. Results of the object detection application. The percentage values are the percentage of the 1000 testing images minus the images where the numbers were not detected.

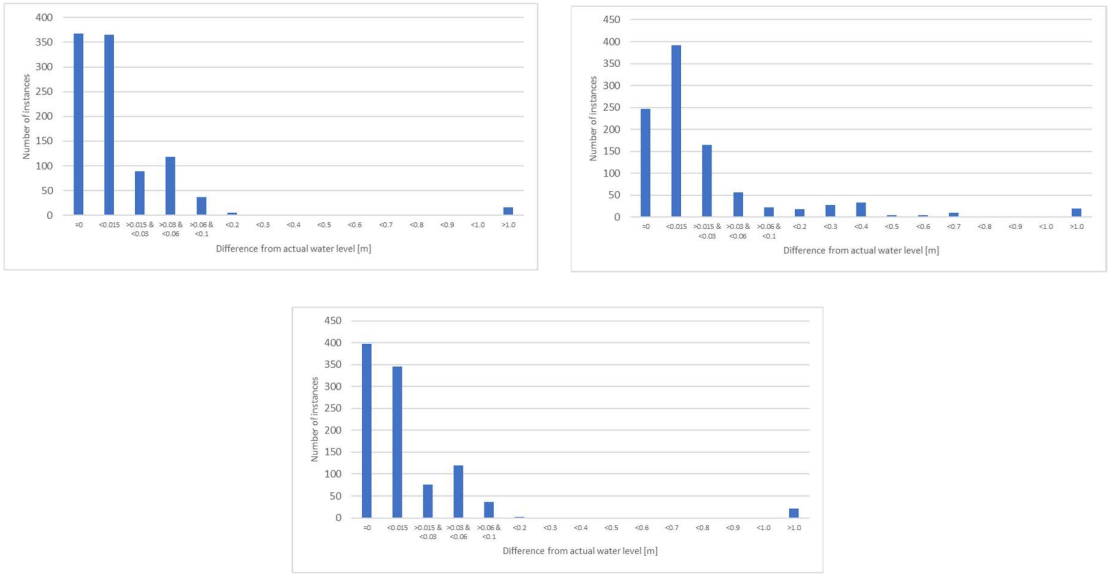

Figure 33. Results of run 1 (to the top-left), run 2 (to the top-right), and run 3 (below), presented as occurrences of water level estimation in intervals.

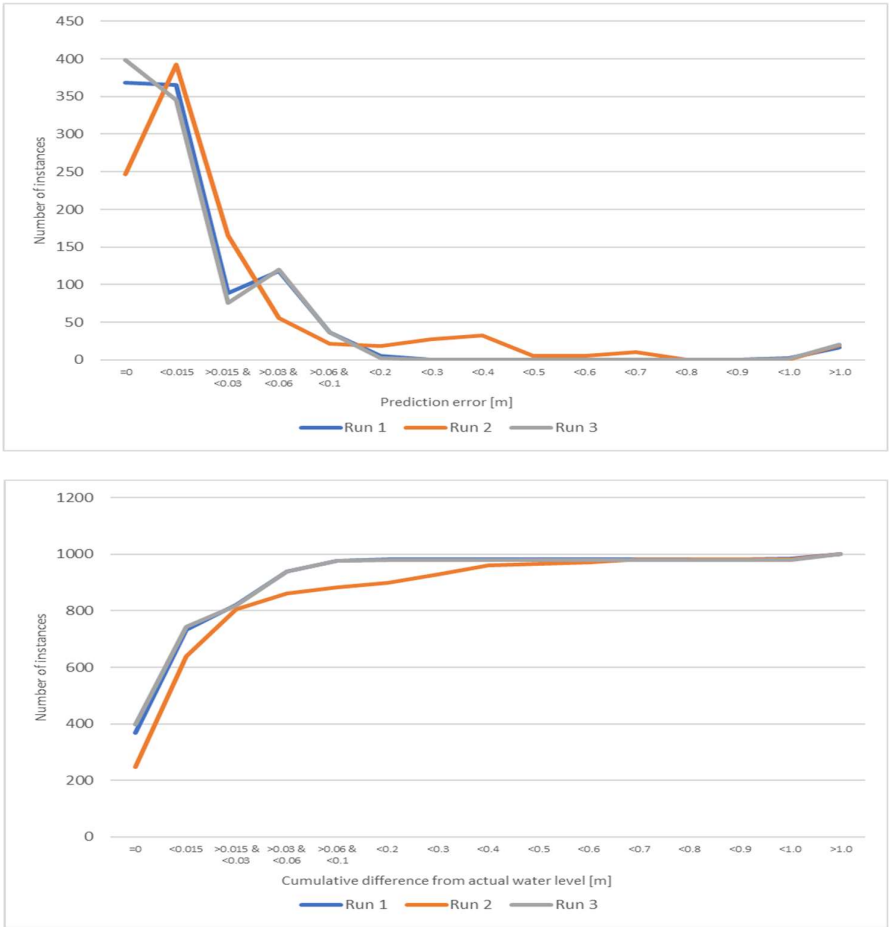

Figure 34. Results of all three runs. The chart above: All runs presented as occurrences of water level estimation in intervals. The chart below: All runs presented as cumulative occurrences of water level estimation.

6 Discussion

The discussion chapter consists of subsections addressing different parts of the results. The subjects for discussion are training the Yolov5 model, the application version 1 that measures the water level using image processing, the application version 2 that measures the water level using Yolov5, and finally a general discussion.

6.1 Discussion about the training of Yolov5

The training started with the model for staff gauge detection. When training the Yolov5, the performance metrics described in chapter 2.3.4, were monitored to evaluate the results of the training. When training the staff gauge detection model the results were quite good already in the beginning. The precision topped at 0,994, the recall at 1, the mAP0,5 value reached 0,993, and the mAP0,5:0,95 value 0,98. The training stopped at 318 epochs, and training was continued, but it stopped as no better results were achieved. It was decided to stop training this model as no improvements were achieved.

The next training was the training of *the numbers* detection model. This model started with a result that also was quite good. The training stopped at 1220 epochs and no further improvement could be achieved with this dataset. Optimizing the hyper-parameters was done on the water line detection dataset, which was a joint dataset with *the numbers.* The results of the training with the optimized hyper-parameters are discussed in the following paragraph.

The third training was the training of the water line detection model. In this training the dataset was pretty much the same as for *the numbers* training, just the water line bounding box labels were added to the dataset. This model's hyper-parameters were then evolved, or optimized before another training run was done. The results of the two training sessions were very similar. The only difference was in the mAP0,5:0,95 values, where the first run gave 0,949 and the second run gave 0,97. So the evolution of the hyper-parameters gave an improvement in the training result for the water line detection model. When training with the optimized hyper-parameters *the numbers* model was trained at the same time, because it was in this training the same model. The result for the metrics for the numbers, the only improvement was for the number 306,5 which had an increase in the mAP0,5:0,95 value, from 0,979 to 0,992. For the rest of the metric values, a slight deterioration could be

seen, which could be a sign of overfitting. Another reason for the deterioration could also be that a too steep learning rate was used. It was decided not to investigate this further as the results were still acceptable.

The metric values are so high that it is very hard to say if the impact on the models' performance is noticeable.

On the staff gauge there also exists a number 304,5, but as this number only was found on a few images it could not be included in the training. The code in the application is made to ignore this number, although it can be falsely detected as another number. The location of 304,5 is on the lower half and to the right on the staff gauge. Such a location for a number is disregarded by the code.

Training was also done with larger models of Yolov5, but no improvement of results was noticed. The results were similar to the results of the Yolov5m model.

6.2 Filtering out bad quality images

Getting images filtered out correctly proved to be a difficult task. Images contain a lot of information, and it is difficult to find a method that works in every case.

Images A and B in Figure 35 are treated equally by the filtering code. If one looks at image B and compares it to image A, they are visually similar. Image A is covered by a haze and image B is out of focus. The images are numerically different and react to image processing differently. Changing the thresholds in the program to correctly filter these images, caused other similar occurrences by other types of images. To create a filtering code that would correctly filter out only blurred images requires deep knowledge of image properties and processing. This was not the main goal of this thesis, so it was decided that the results achieved were acceptable.

Figure 35. Image A is covered by a haze while image B is out of focus.

6.3 Discussion on application version 1 – Using image processing to find the water line

The first of the two application versions used image processing to locate the water line. There were two different runs executed with this application. The staff gauge was detected in all images, in both runs, which was a very good result. The background was very similar in all test images and the staff gauge location did not change in relation to the background and changed only slightly in relation to the image. The staff gauge itself has very clear borders and it is clearly distinguishable from the background, even in bad weather. These are factors that can explain the good results for the staff gauge detection.

Numbers were not detected in a few of the images and the result was pretty similar for both runs. Run 1 was performing much better as the number of correctly estimated images was 255, against 143 for run 2. 255 correctly estimated images equal 26% of the test images. Both runs had similar results for estimating with a difference of less than ± 3 cm, 617 and 608 respectively. This is 63% and 61% of the test images that are estimated within a ± 3 cm margin of the correct level. The number of images with a difference greater than ± 30 cm to the actual water level was for run 1, 129 images to 170 for run 2. Looking at the cumulative chart in Figure 32, one can clearly see that run 1 performs better than run 2.

As mentioned earlier, the first training session with default hyper-parameters resulted in slightly better results on all but one number detection. The results of this model's performance on the test images are also slightly better than the model with optimized hyperparameters.

In this application, there was also image processing included. All but one article reviewed in this thesis relied on image processing to find the water line in the images. This requires that the lighting and weather conditions are very similar for the image quality to be stable. Small changes in the conditions ask for changes in parameters. However, it is also very difficult to find direct connections between the conditions and the parameters. Figure 36 below is an example of images that are visually similar, but when applying the image processing to find the water line, the results are very different. The parameters or thresholds are the same for all three images. This part of the application is its Achilles heel.

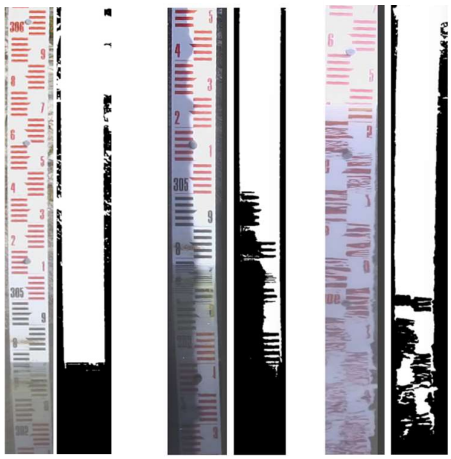

Figure 36. Different staff gauge images with their corresponding binarized image to find the water line.

6.4 Discussion on application version 2 – Using Yolov5 to find the water line

There were three different runs with application version 2 and the staff gauge was detected in all three runs. The first and the third run had clearly better results than the second run. The third run estimated the water level correctly in 40% of the test images. That is 398 images. 83% of all test images, or 819 images, the water level was estimated to be within 3cm of the actual level. This is a very good result for this application, and it performs much better than application version 1, which uses image processing. This shows that the machine learning model Yolov5 has huge capabilities.

The calculation of the water level relies on a reference point and the scale of the image. The reference point is the detected bounding box that is the furthest down on the staff gauge. The scale is calculated as the actual distance between two of the detected bounding

boxes that are the furthest apart divided by the pixel difference. If only one bounding box is detected, its height which is 4cm is divided by the pixels it has in the image. In this case, if the bounding box is far from the water level, it can cause a measuring error due to the inexact scale. The inexact scale is due to the bounding box being small and then one pixel has much greater weight than if the bounding box or distance is bigger. Figure 37 below is an example of such a case.

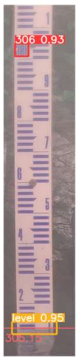

Figure 37. Only one bounding box is found, and it is far from the water line.

6.5 General discussion

There are ways that might improve the performance of the water level measuring application version 2. One way could be by increasing the number of images in the datasets.

Yolov5 performed worse in detecting the staff gauge when the images were distorted in some way. Image distortion could consist of strong lighting reflecting off the staff gauge, staff gauges with half of the staff gauge, in the vertical direction, covered by a shadow. Images with high reflections off the water body or staff gauges covered by dirt are also troublesome images for the model. The extreme in conditions is a problem for a machine-learning model like Yolov5. This is no surprise as it is very difficult to visually detect these features as well. You could enlarge the datasets including more of these kinds of images, but then the question arises: “Will it actually enhance the performance of the model?” According to Dodge and Karam, it might make the performance worse. (Dodge & Karam, 2016)

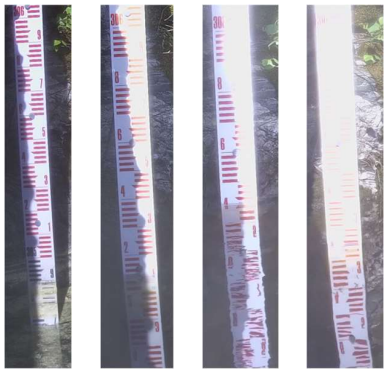

Figure 38. Examples of images that are difficult for Yolov5 to detect the staff gauge and the numbers on.

Another way could be by addressing the anchor box templates. The Yolov5 model uses anchor box templates against which the bounding boxes in the training dataset are evaluated. (Jacob Solawetz, 2020a) These anchor box templates are calculated from the training with the COCO dataset and might differ from the custom dataset. The staff gauge is tall and narrow and has a slightly different form than the objects in the COCO dataset. It could therefore be an idea to alter the anchor box templates manually to better fit this custom dataset.

Adding a barrier around the staff gauge will reduce the amount of debris gathering around and on the staff gauge otherwise interfering with water line detection. This barrier should prevent or at least greatly reduce leaves, wooden sticks and branches, pollen, and other objects from coming too close to the staff gauge. This way there will be more good images to detect on.

The challenge for Yolov5 to detect in images that are captured in extreme lighting conditions is big. Using a camera for near-infrared image capturing seems to give an advantage in finding the water line as the water body does not reflect infrared light the same way it reflects visible light. Water absorbs infrared light more than visible light, which causes the water line to be more clearly detectable. (Mangold et al., 2013) Zhang et al. found out in laboratory tests that different materials of staff gauges and paints reflect infrared light differently. (Zhang, Zhou, Liu & Gao, 2019) The images in Figure 39 below are examples of images captured in visible and infrared lighting.

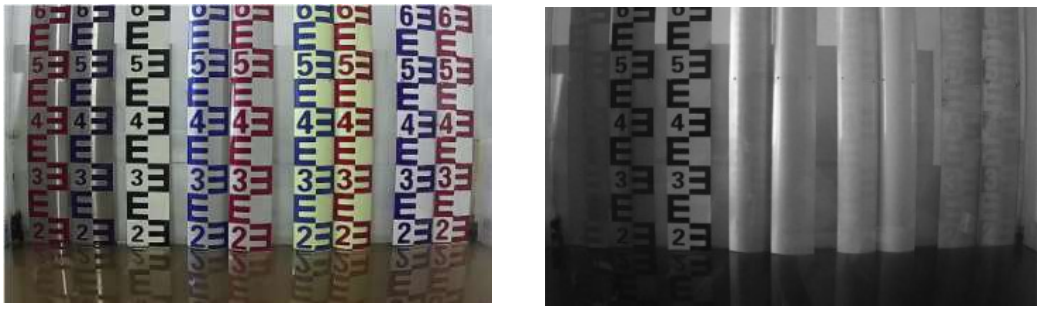

Figure 39. The image to the left is captured in normal visible light and the image to the right is captured in infrared light.

Water reflections and strong direct sunlight proved to be a problem for the object detection. This should be, if possible, paid attention to when installing a staff gauge. Installing the staff gauge in a place where direct sunlight can shine on it should be avoided. The staff gauge could also be installed on a dark-colored background plate that reaches approximately 10cm on each side of the staff gauge. This would limit the interference from the background.

The grayscale conversion used in the process of finding the water level uses the standard weighted method. (Radiocommunication Sector of International Telecommunication Union, 2011) As this is not using an equal amount of all three colours. As almost 60% of the green intensity is used and only just over 10% of the blue, could this be unfavourable in certain light conditions? In the evenings just after sunset, the blue colour tends to be dominant as the red colour tends to be dominant the hour before sunset and the hour after sunrise. As images are numerical matrices, this could have an impact on how these images are processed.

A polarizing filter will reduce the reflection on the water making the reflection darker. This could enhance the image so that the water line would be easier to detect. The polarizing filter is one of the filters that cannot be replicated by software due to that the image itself does not contain data of polarizing light. (Wikipedia, 2023)

Clearly marking defined water levels on this background plate with a line could also have a beneficial effect on finding a reference point for the measurement. Reducing the external interfering factors might give less fluctuations in image quality thereby making it easier to develop a pre-processing setup and detect water levels.

7 Conclusions

Yolov5 is a powerful machine-learning model. Looking at the results in this thesis it outperforms image processing when it comes to finding the water line. Image processing can be effective in good-quality images, but when it comes to images with less contrast or otherwise challenging features, Yolov5 works better. It would be interesting to look more closely at only using the infrared mode on the camera. Zhang et al. found out that certain materials of the staff gauge work better when using infrared cameras than others. (Zhang, Zhou, Liu & Gao, 2019) To find a staff gauge of such material, add a plate of suitable material and colour behind it, use of infrared imaging and constructing a barrier to prevent debris from attaching itself to the staff gauge, the results might be better and more stable.

8 References

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. https://github.com/AlexeyAB/darknet.

De Oliveira Fleury, G. R., Do Nascimento, D. V., Galvão Filho, A. R., Lima Ribeiro, F. D. S., De Carvalho, R. V., & Coelho, C. J. (2020). Image-based river water level estimation for redundancy information using deep neural network. *Energies*, *13*(24). https://doi.org/10.3390/en13246706

Dodge, S., & Karam, L. (2016). *Understanding How Image Quality Affects Deep Neural Networks*.

Dou, G., Chen, R., Han, C., Liu, Z., & Liu, J. (2022). Research on Water-Level Recognition Method Based on Image Processing and Convolutional Neural Networks. *Water (Switzerland)*, *14*(12). https://doi.org/10.3390/w14121890

Du, J. (2018). Understanding of Object Detection Based on CNN Family and YOLO. *J. Phys*, 12029. https://doi.org/10.1088/1742-6596/1004/1/012029

Fraser, B.-A. W. paper. (2004). *Understanding Digital Raw Capture*.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5)*. http://www.cs.berkeley.edu/˜rbg/rcnn.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. https://www.deeplearningbook.org/

He, K., Sun, J., & TTang, X. (2009). Single image haze removal using dark channel prior. *2009 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009*, 1956–1963. https://doi.org/10.1109/CVPRW.2009.5206515

Horenstein, H. (2005). *Black and White Photography* (3rd ed.). Little, Brown and Company.

Jacob Solawetz. (2020a, July 13). *What are Anchor Boxes in Object Detection?* https://blog.roboflow.com/what-is-an-anchor-box/

Jacob Solawetz. (2020b, July 29). *What is YOLOv5? A Guide for Beginners.* https://blog.roboflow.com/yolov5-improvements-and-evaluation/

Jocher, G., & Waxmann, S. (2023). *Ultralytics Yolov5*. https://docs.ultralytics.com/yolov5/

Khan, S., Rahmani, H., Shah, S. A. A., & Bennamoun, M. (2018). A Guide to Convolutional Neural Networks for Computer Vision. In G. Medioni & S. Dickenson (Eds.), *Synthesis Lectures on Computer Vision* (Issue 1). Morgan & Claypool Publishers.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Conference on Neural Information Processing Systems. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Lin, T. (2018, December 3). *labelImg*. https://github.com/heartexlabs/labelImg/releases

Mangold, K., Shaw, J. A., & Vollmer, M. (2013). The physics of near-infrared photography. *European Journal of Physics*, *34*(6). https://doi.org/10.1088/0143-0807/34/6/S51

Padilla, R., Passos, W. L., Dias, T. L. B., Netto, S. L., & Da Silva, E. A. B. (2021). A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit. *Electronics 2021, Vol. 10, Page 279*, *10*(3), 279. https://doi.org/10.3390/ELECTRONICS10030279

Qiao, G., Yang, M., & Wang, H. (2022). A Water Level Measurement Approach Based on YOLOv5s. *Sensors*, *22*(10). https://doi.org/10.3390/s22103714

Radiocommunication Sector of International Telecommunication Union. (2011). Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios. In *Recommendation ITU-R BT.601-7*. Radiocommunication Sector of International Telecommunication Union. https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.pdf

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). *You Only Look Once: Unified, Real-Time Object Detection*. http://arxiv.org/abs/1506.02640

Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 7263–7271. https://doi.org/10.1109/CVPR.2017.690

Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement*. https://pjreddie.com/yolo/.

Renewable energy production in Norway. (2016, May 11). Ministry of Petroleum and Energy. https://www.regjeringen.no/en/topics/energy/renewable-energy/renewable-energy-production-in-norway/id2343462/

Sabbatini, L., Palma, L., Belli, A., Sini, F., & Pierleoni, P. (2021). A computer vision system for staff gauge in river flood monitoring. *Inventions*, *6*(4). https://doi.org/10.3390/inventions6040079

Wikipedia. (2023, July 23). *Polarizing filter (photography)*. Wikipedia. https://en.wikipedia.org/wiki/Polarizing_filter_(photography)

Zafar, I., Tzanidou, G., Burton, R., Patel, N., & Araujo, L. (2018). *Hands-On Convolutional Neural Networks with TensorFlow Solve computer vision problems with modeling in TensorFlow and Python* (A. Varangaonkar, S. Mandal, A. Gour, & V. Dwivedi, Eds.; 1st ed.). Packt Publishing Ltd.

Zhang, Z., Zhou, Y., Liu, H., & Gao, H. (2019). In-situ water level measurement using a NIR-imaging video camera. *Flow Measurement and Instrumentation*, *67*, 95–106. https://doi.org/10.1016/j.flowmeasinst.2019.04.004

Zhang, Z., Zhou, Y., Liu, H., Zhang, L., & Wang, H. (2019). Visual measurement of water level under complex illumination conditions. *Sensors (Switzerland)*, *19*(19). https://doi.org/10.3390/s19194141

Zou, Z., Chen, K., Shi, Z., Guo, Y., & Ye, J. (2023). *Object Detection in 20 Years: A Survey*. https://arxiv.org/pdf/1905.05055.pdf?fbclid=IwAR0ILGAWTwU-9-iH6lZyPFXYXA5JRWarM_XoSJ78QEhmnn-txvr_iGEzCio

Appendix A – Code for application alternative 1

```
# daFinalCode_01.py
# Find the staff gauge in an image using a custom trained Yolov5m model. The code also
# finds the numbers (hundreds only, not singulars) using a custom trained Yolov5m
# model.
# Thereafter it finds the waterline using image processing and calculates the water level
# using the found number as level reference.
# Version 1.0
# Author: Bo-Anders Näs - 2023

import os
import cv2 as cv
import numpy as np
import math
import torch
from skimage.transform import hough_line, hough_line_peaks
from scipy.ndimage import binary_fill_holes

# Assign directory where the image files are
directory = 'D:/Thesis Images for final test/'

# Assign a directory where the results are saved
resultDirectory = 'D:/Results/'

# Assign directory where the staff gauge model is
staffGaugeDirectory = 'D:/WaterlevelMeasuring_Yolov5m/Yolov5_Custom_StaffGauge'

# Assign directory where the numbers model is
numbersDirectory = 'D:/WaterlevelMeasuring_Yolov5m/Yolov5_Custom_Numbers'

# Assign directory where the Water line model is
waterlineDirectory = 'D:/WaterlevelMeasuring_Yolov5m/Yolov5_Custom_Waterline'

# Enlarges the crop area down and to the right (h, w)
cropCorrection = [300, 50]

# Convert normalized values for x, y, width and height to conform with the image size
def convert(size,x,y,w,h):
    coordinates = np.zeros(4)
    dw = 1./size[0]
    dh = 1./size[1]
    x = x/dw
    w = w/dw
    y = y/dh
    h = h/dh
    coordinates[0] = x-(w/2.0)
```

```
    coordinates[1] = x+(w/2.0)
    coordinates[2] = y-(h/2.0)
    coordinates[3] = y+(h/2.0)

    return (coordinates)

# Identify a possible slanting angle of the staff gauge
def getStaffAngle(image, imgLabel):

    # Get the coordinates of the bounding coordinates4Box
    imgLabel = imgLabel.tolist()
    x0 = int(imgLabel[0][0])
    x1 = int(imgLabel[0][2])
    y0 = int(imgLabel[0][1])
    y1 = int(imgLabel[0][3])

    # Load image
    img = cv.imread(image)

    # Crop image a bit larger than the found coordinates4Box
    # This because the found box is not necessary covering the whole staff nor including
    # the water line
    imgCropped = img[y0:y1+cropCorrection[0], x0:x1+cropCorrection[1]]

    # Process the image to more easily find the borders of the measurement staff
    imgGray = cv.cvtColor(imgCropped, cv.COLOR_BGR2GRAY)
    imgBlur = cv.GaussianBlur(imgGray, (9, 9), 0)
    imgThresh = cv.threshold(imgBlur, 0, 255, cv.THRESH_BINARY_INV +
                                              cv.THRESH_OTSU)[1]

    # Apply erode to "remove" the small parts in the image.
    # A smaller value for the kernel will more effectively remove the small parts.
    kernel = cv.getStructuringElement(cv.MORPH_CROSS, (3, 3))
    imgErode = cv.erode(imgThresh, kernel, iterations=5)

    # Canny will bring out the edges
    image = cv.Canny(imgErode, 10, 10)

    # Straight-line Hough transform
    # Set a precision of 0.5 degree, this means the range -pi/2 to pi/2 (-90 to 90 degrees)
    # divided in 360 parts = 0,5 degrees precision.
    tested_angles = np.linspace(-np.pi / 2, np.pi / 2, 360, endpoint=False)
    h, theta, d = hough_line(image, theta=tested_angles)

    returnAngle = [0]*100
    i = 0
    # Find the angle of the staff gauge
    for _, angle, dist in zip(*hough_line_peaks(h, theta, d)):
        (x0, y0) = dist * np.array([np.cos(angle), np.sin(angle)])
```

```
        lineSlope = np.tan(angle + np.pi/2)

        returnAngle[i] = math.degrees(np.tan(1/lineSlope))
        i +=1

    return imgCropped, returnAngle[1]

# Rotate the image around its center
def rotateImage(image, angle: float):
    # Get the shape of the image
    (h, w) = image.shape[:2]

    # Get center
    center = (w // 2, h // 2)

    # Define the rotation matrix
    M = cv.getRotationMatrix2D(center, angle, 1.0)

    # Create a white background
    whiteBackground = np.zeros([h,w,3],dtype=np.uint8)
    whiteBackground[:] = 255

    # Rotates the image using the rotation matrix
    newImage = cv.warpAffine(image, M, (w, h), flags=cv.INTER_CUBIC,
                                                    borderMode=cv.BORDER_WRAP)

    return newImage

# Crop the image using threshold
def cropAgain(image, angle):
    # This code calculates a new cropping area based on original labeling
    # and the corrected rotation angle

    # Get image measurements
    height = image.shape[0] - cropCorrection[0]
    width = image.shape[1] - cropCorrection[1]

    # Calculate the possible reduction in width due to rotation
    a = (height*math.tan((angle*3.141593)/180)/2)

    # New coordinates for cropped image
    x0 = int(a)
    x1 = width - int(2*a/3) + cropCorrection[1]
    y0 = 0
    y1 = height + +cropCorrection[0]

    # and crop the image ... again
    imgCropped = image[y0:y1, x0:x1]
```

```
    return imgCropped

# Locate the water line
def findWaterline(image):

    # Remove noise from the image using a bilateral filter.
    def blurImage(image, d = 5, sigmaColor = 75, sigmaSpace = 50 ):

        # Blur image to reduce noice
        imgBlur = cv.bilateralFilter(image, d, sigmaColor, sigmaSpace)

        return imgBlur

    # Get image shape
    h, w = image.shape[:2]

    # Convert image to grayscale
    imgGray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    # Apply adaptive blurring to remove noise
    imgBlur = blurImage(imgGray)

    # Calculate the optimal threshold values using Otsu's method
    _, thresh = cv.threshold(imgBlur, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)

    # Fill holes in the binary image
    imgFilled = binary_fill_holes(thresh)

    # Find the row where there are more than 10% white pixels
    nonBlackCount = 0
    rowMem = 0
    # Loop through the image from almost bottom upwards, row by row and count number
    # of black pixels
    for i in range(len(imgFilled)-100, 0, -1):
        # Count how many black pixel on each row
        blackCount = np.sum(imgFilled[i,:] == 0)
        # If there are more than 10% white pixels on more 5 rows in a row,
        # the first row of these is the row where the waterline is
        if blackCount < 0.9*w:
            if nonBlackCount == 0:
                rowMem = i
            nonBlackCount += 1
        else:
            nonBlackCount = 0
        if nonBlackCount > 5:
            break

    return rowMem
```

```
# Calculate the water level
def calculateWaterLevel(boundingBoxes, waterlineRow, imageSize):

    # Convert bounding box tensor to list
    boundingBoxes = boundingBoxes.tolist()
    h, w = imageSize
    level = 0.0
    duplicate = 0

    # As some models also include the water level bounding box, this has to be removed to
    # not interfere with the water level calculations
    i=0
    for row in coordinates4Boxes:
        if row[-1].item() == 3.0:
            boundingBoxes = np.delete(boundingBoxes, i, axis=0)
        i=+ 1

    # First element: Number to the left of image center line = 1 and number to the right = 0
    # Second element: Is the number in the upper third of the image = 0,
    # in the mid third = 1 or in the lower third = 2
    # Third element: Is there two numbers to the left of the image center line = 1, else = 0
    if len(boundingBoxes) > 0:

        # Define on which level the bounding box(es) are
        box = []
        position = [0,0,0]
        error = ["error", 0.0, 0.0, 0.0]
        for i in range(0, len(boundingBoxes)):

            # Check if the midpoint of the bounding box is to the right of the cropped image
            # center line
            if (boundingBoxes[i][0] + boundingBoxes[i][2]) / 2 > w/2:
                # Check if the lowest edge is above the center line of the cropped image
                if boundingBoxes[i][3] < h/2 and position[0] == 0:
                    box.append(("306.5", boundingBoxes[i][0], boundingBoxes[i][3],
                                        abs(boundingBoxes[i][3] - boundingBoxes[i][1])))
                    position[0] = 1
                else:
                    box.append(error)
            else:
                # Check if the midpoint of the bounding box is above the center line of the
                # cropped image
                if (boundingBoxes[i][1] + boundingBoxes[i][3]) / 2 < h/2:
                    if position[1] == 0:
                        box.append(("306", boundingBoxes[i][2], boundingBoxes[i][3],
                                        abs(boundingBoxes[i][3] - boundingBoxes[i][1])))
                        position[1] = 1
                    else:
```

```
            box.append(error)

        if (boundingBoxes[i][1] + boundingBoxes[i][3]) / 2 > h/2:
            box.append(("305", boundingBoxes[i][2], boundingBoxes[i][3],
                                    abs(boundingBoxes[i][3] - boundingBoxes[i][1])))
            position[2] += 1

# If a duplicate of bounding box 3 is found (305), it is registered
if position[2] == 2:
    duplicate = 2

if position[2] < 3:
    # If a duplicate is found for the 305 bounding box, the bounding box furthest down
    # is disregarded.
    box4Use = [0,0,0,0]

    tmpBox = [0,0,0,0]
    for i in range(len(box)):
        if box[i][0] == '306.5':
            box4Use[0] = box[i]
        if box[i][0] == '306':
            box4Use[1] = box[i]
        if box[i][0] == '305':
            if duplicate == 2:
                tmpBox = box[i]
                duplicate = 1
            elif duplicate == 1:
                if tmpBox[3] > box[i][3]:
                    box4Use[2] = box[i]
                else:
                    box4Use[2] = tmpBox
            else:
                box4Use[2] = box[i]

    # Calculate the scale for the image by calculating the distance between the two
    # bounding boxes that are the furthest vertically.
    # Depending on if there exist one or several bounding boxes, calculate the
    # difference and divide with the actual distance 50, 100 or 150cm, or if only one
    #bounding box take the height of it and divide with 4cm as measured from staff
    # gauge
    if position[0] and position[2]:                 # 306,5 and 305
        scale = 150 / (box4Use[2][2] - box4Use[0][2])   # scale = cm / pixel
        pixelDifference = waterlineRow - box4Use[2][2]
        level = 305.0 - (pixelDifference * scale) / 100
    elif position[0] and position[1]:               # 306,5 and 306
        scale = 50 / (box4Use[1][2] - box4Use[0][2])    # scale = cm / pixel
        pixelDifference = waterlineRow - box4Use[1][2]
        level = 306.0 - (pixelDifference * scale) / 100
    elif position[1] and position[2]:               # 306 and 305
```

```
            scale = 100 / (box4Use[2][2] - box4Use[1][2])   # scale = cm / pixel
            pixelDifference = waterlineRow - box4Use[2][2]
            level = 305.0 -  (pixelDifference * scale) / 100
        elif position[0] and not position[1] and not position[2]:
            scale = 4 / box4Use[0][3]                   # scale = cm / pixel, the bounding box is
                                                          approximately 4cm high
            pixelDifference = waterlineRow - box4Use[0][2]
            level = 306.5 - (pixelDifference * scale) / 100
        elif position[1] and not position[0] and not position[2]:
            scale = 4 / box4Use[1][3]                   # scale = cm / pixel, the bounding box is
                                                          approximately 4cm high
            pixelDifference = waterlineRow - box4Use[1][2]
            level = 306.0 - (pixelDifference * scale) / 100
        elif position[2] and not position[0] and not position[1]:
            scale = 4 / box4Use[2][3]                   # scale = cm / pixel, the bounding box is
                                                          approximately 4cm high
            pixelDifference = waterlineRow - box4Use[2][2]
            level = 305.0 - (pixelDifference * scale) / 100
        else:
            level = 0.0
    else:
        level = 0.0

  if level != 0.0:
    return level, "Water level detected"
  else:
    return level, "No water line could be detected"

# Load the custom trained Yolov5m models. Alternative 1 was the best performing model
modelStaff = torch.hub.load(staffGaugeDirectory, 'custom', path=staffGaugeDirectory +
                '/staffGauge/staffGaugeBest.pt', source='local')
# modelNumbers = torch.hub.load(numbersDirectory, 'custom', path=waterlineDirectory
                        + '/Waterline/combinedBest.pt', source='local')
                                                                    # used in alternative 2
modelNumbers = torch.hub.load(numbersDirectory, 'custom', path=numbersDirectory +
                        '/Numbers/best.pt', source='local')         # used in alternative 1

# Set allowed confidence limit and Iou (intersection over union)
# The IoU value is used to filter out unwanted overlapping bounding boxes
modelStaff.conf = 0.1  # confidence threshold (0-1)
modelStaff.iou = 0.9  # NMS IoU threshold (0-1)
modelNumbers.conf = 0.50  # confidence threshold (0-1)
modelNumbers.iou = 0.85  # NMS IoU threshold (0-1)

# Get all the image file names in the folder
images = [f for f in os.listdir(directory) if f.endswith('.jpg')]

# Open a file for storing the results
f = open(resultDirectory + 'inferenceResultsModel01_kontroll_2.csv','w')
```

```
for imgName in images:
    # Define a new filename for the newly saved file
    resFile = resultDirectory + "/ResMod01_" + imgName

    # Locaction of image
    imgIn = directory + imgName

    # Run the detection and store in resultsStaff
    resultsStaff = modelStaff(imgIn)

    # Get coordiates for bounding box
    coordinates4Box  = resultsStaff.xyxy[0]

    # Initilaize variables
    staffGaugeFound = "False"
    numbersFound = "False"
    numbers = 0
    level = 0.0

    # If staff gauge is found
    if len(coordinates4Box) > 0:
        # Staff gauge found
        staffGaugeFound = "True"

        # Crop image and straigthen it
        img, angle = getStaffAngle(imgIn, coordinates4Box)
        imgRotated = rotateImage(img, -angle)

        # Get the cropped and straightened image size
        h, w, _ = imgRotated.shape
        imgSize = (h,w)

        # Run the numbers detection and store in resultsNumbers
        resultsNumbers = modelNumbers(imgRotated)

        # Get coordiates for bounding boxes
        coordinates4Boxes  = resultsNumbers.xyxy[0]

        # Check if coordinates4Boxes contain numbers (extra check when the combined
        # model is used to get numbers reference correct)
        for row in coordinates4Boxes:
            if row[-1].item() < 3.0:
                numbers = 1

        # If numbers on staff gauge is found... used as water level reference
        if len(coordinates4Boxes > 0) and numbers == 1:
            # Numbers found
            numbersFound = "True"
```

```
            # Crop the image again to remove the background found at the side of the
            # previously cropped image
            imgCropped = cropAgain(imgRotated, angle)

            # Find the water line and store the row at which the water line was found
            row = findWaterline(imgRotated)

            # Calculate the water level based on the found number boxes and the row where
            # the water line was found on.
            # The water level is 0.0 if no water level could be calculated
            # First level is aquired via image processing and second is using Yolov5
            level, comment = calculateWaterLevel(coordinates4Boxes, row,
                                                 imgRotated.shape[:2])
            level = round(level, 2)

            # Add lines to the image where the water line is located
            cv.line(imgRotated, (0, row), (w, row), (255, 100, 100), 2)   # Red

            # Add text to the image stating the water level
            cv.putText(imgRotated, str(level), (5, row + 40), cv.FONT_HERSHEY_SIMPLEX, 1,
                                                        (255, 100, 100),  2, cv.LINE_AA)

        # Write the result to a csv file
        data = resFile + ", " + staffGaugeFound + ", " + numbersFound +", " + str(level) + "\n"
        f.writelines(data)

# Close the csv file
f.close()
```

Appendix B – Code for application alternative 2

```
# daFinalCode_02.py
# Find the staff gauge in an image using a custom trained Yolov5m model. The code also
# finds the numbers (hundreds only, not singulars) using a custom trained Yolov5m
# model. Thereafter it finds the waterline and calculates the water level using the found
# number as level reference.
# Version 1.0
# Author: Bo-Anders Näs - 2023

import os
import cv2 as cv
import numpy as np
import math
import torch
from skimage.transform import hough_line, hough_line_peaks

# Assign directory where the image files are
directory = 'D:/Thesis Images for final test/'

# Assign a directory where the results are saved
resultDirectory = "D:/Results/"

# Assign directory where the staff gauge model is
staffGaugeDirectory = 'D:/WaterlevelMeasuring_Yolov5m/Yolov5_Custom_StaffGauge'

# Assign directory where the numbers model is
numbersDirectory = 'D:/WaterlevelMeasuring_Yolov5m/Yolov5_Custom_Numbers'

# Assign directory where the Water line model is
waterlineDirectory = 'D:/WaterlevelMeasuring_Yolov5m/Yolov5_Custom_Waterline'

# Enlarges the crop area down and to the right (h, w)
cropCorrection = [300, 50]

# Convert normalized values for x, y, width and height to conform with the image size
def convert(size,x,y,w,h):
    coordinates = np.zeros(4)
    dw = 1./size[0]
    dh = 1./size[1]
    x = x/dw
    w = w/dw
    y = y/dh
    h = h/dh
    coordinates[0] = x-(w/2.0)
    coordinates[1] = x+(w/2.0)
    coordinates[2] = y-(h/2.0)
    coordinates[3] = y+(h/2.0)
```

```
    return (coordinates)

# Identify a possible slanting angle of the staff gauge
def getStaffAngle(image, imgLabel):

    # Get the coordinates of the bounding coordinates4Box
    imgLabel = imgLabel.tolist()
    x0 = int(imgLabel[0][0])
    x1 = int(imgLabel[0][2])
    y0 = int(imgLabel[0][1])
    y1 = int(imgLabel[0][3])

    # Load image
    img = cv.imread(image)

    # Crop image a bit larger than the found coordinates4Box
    # This because the found box is not necessary covering the whole staff nor including
    # the water line
    imgCropped = img[y0:y1+cropCorrection[0], x0:x1+cropCorrection[1]]

    # Process the image to more easily find the borders of the measurement staff
    imgGray = cv.cvtColor(imgCropped, cv.COLOR_BGR2GRAY)
    imgBlur = cv.GaussianBlur(imgGray, (9, 9), 0)
    imgThresh = cv.threshold(imgBlur, 0, 255, cv.THRESH_BINARY_INV +
                                              cv.THRESH_OTSU)[1]

    # Apply erode to "remove" the small parts in the image.
    # A smaller value for the kernel will more effectively remove the small parts.
    kernel = cv.getStructuringElement(cv.MORPH_CROSS, (3, 3))
    imgErode = cv.erode(imgThresh, kernel, iterations=5)

    # Canny will bring out the edges
    image = cv.Canny(imgErode, 10, 10)

    # Straight-line Hough transform
    # Set a precision of 0.5 degree, this means the range -pi/2 to pi/2 (-90 to 90 degrees)
    # divided in 360 parts = 0,5 degrees precision.
    tested_angles = np.linspace(-np.pi / 2, np.pi / 2, 360, endpoint=False)
    h, theta, d = hough_line(image, theta=tested_angles)

    returnAngle = [0]*100
    i = 0
    # Find the angle of the staff gauge
    for _, angle, dist in zip(*hough_line_peaks(h, theta, d)):
        (x0, y0) = dist * np.array([np.cos(angle), np.sin(angle)])
        lineSlope = np.tan(angle + np.pi/2)

        returnAngle[i] = math.degrees(np.tan(1/lineSlope))
```

```
        i +=1

    return imgCropped, returnAngle[1]

# Rotate the image around its center
def rotateImage(image, angle: float):
    # Get the shape of the image
    (h, w) = image.shape[:2]

    # Get center
    center = (w // 2, h // 2)

    # Define the rotation matrix
    M = cv.getRotationMatrix2D(center, angle, 1.0)

    # Create a white background
    whiteBackground = np.zeros([h,w,3],dtype=np.uint8)
    whiteBackground[:] = 255

    # Rotates the image using the rotation matrix
    newImage = cv.warpAffine(image, M, (w, h), flags=cv.INTER_CUBIC, borderMode =
                                                                    cv.BORDER_WRAP)

    return newImage

# Crop the image using threshold
def cropAgain(image, angle):
    # This code calculates a new cropping area based on original labeling
    # and the corrected rotation angle

    # Get image measurements
    height = image.shape[0] - cropCorrection[0]
    width = image.shape[1] - cropCorrection[1]

    # Calculate the possible reduction in width due to rotation
    a = (height*math.tan((angle*3.141593)/180)/2)

    # New coordinates for cropped image
    x0 = int(a)
    x1 = width - int(2*a/3) + cropCorrection[1]
    y0 = 0
    y1 = height + +cropCorrection[0]

    # and crop the image ... again
    imgCropped = image[y0:y1, x0:x1]

    return imgCropped
```

```
# Calculate the water level
def calculateWaterLevel(boundingBoxes, waterlineRow, imageSize):

    # Convert bounding box tensor to list
    boundingBoxes = boundingBoxes.tolist()
    h, w = imageSize
    level = 0.0
    duplicate = 0

    # As some models also include the water level bounding box, this has to be removed to
    # not interfere with the water level calculations
    i=0
    for row in coordinates4Boxes:
        if row[-1].item() == 3.0:
            boundingBoxes = np.delete(boundingBoxes, i, axis=0)
        i=+ 1

    # First element: Number to the left of image center line = 1 and number to the right = 0
    # Second element: Is the number in the upper third of the image = 0, in the mid third =
    # 1 or in the lower third = 2
    # Third element: Is there two numbers to the left of the image center line = 1, else = 0
    if len(boundingBoxes) > 0:

        # Define on which level the bounding box(es) are
        box = []
        position = [0,0,0]
        error = ["error", 0.0, 0.0, 0.0]
        for i in range(0, len(boundingBoxes)):

            # Check if the midpoint of the bounding box is to the right of the cropped image
            # center line
            if (boundingBoxes[i][0] + boundingBoxes[i][2]) / 2 > w/2:
                # Check if the lowest edge is above the center line of the cropped image
                if boundingBoxes[i][3] < h/2 and position[0] == 0:
                    box.append(("306.5", boundingBoxes[i][0], boundingBoxes[i][3],
                                        abs(boundingBoxes[i][3] - boundingBoxes[i][1])))
                    position[0] = 1
                else:
                    box.append(error)
            else:
                # Check if the midpoint of the bounding box is above the center line of the
                # cropped image
                if (boundingBoxes[i][1] + boundingBoxes[i][3]) / 2 < h/2:
                    if position[1] == 0:
                        box.append(("306", boundingBoxes[i][2], boundingBoxes[i][3],
                                        abs(boundingBoxes[i][3] - boundingBoxes[i][1])))
                        position[1] = 1
                    else:
                        box.append(error)
```

```
        if (boundingBoxes[i][1] + boundingBoxes[i][3]) / 2 > h/2:
            box.append(("305", boundingBoxes[i][2], boundingBoxes[i][3],
                                    abs(boundingBoxes[i][3] - boundingBoxes[i][1])))
            position[2] += 1

# If a duplicate of bounding box 3 is found (305), it is registered
if position[2] == 2:
    duplicate = 2

if position[2] < 3:
    # If a duplicate is found for the 305 bounding box, the bounding box furthest down
    # is disregarded.
    box4Use = [0,0,0,0]

    tmpBox = [0,0,0,0]
    for i in range(len(box)):
        if box[i][0] == '306.5':
            box4Use[0] = box[i]
        if box[i][0] == '306':
            box4Use[1] = box[i]
        if box[i][0] == '305':
            if duplicate == 2:
                tmpBox = box[i]
                duplicate = 1
            elif duplicate == 1:
                if tmpBox[3] > box[i][3]:
                    box4Use[2] = box[i]
                else:
                    box4Use[2] = tmpBox
            else:
                box4Use[2] = box[i]

    # Calculate the scale for the image by calculating the distance between the two
    # bounding boxes that are the furthest vertically.
    # Depending on if there exist one or several bounding boxes, calculate the
    # difference and divide with the actual distance 50, 100 or 150cm, or if only one
    # bounding box take the height of it and divide with 4cm as measured from staff
    # gauge
    if position[0] and position[2]:                 # 306,5 and 305
        scale = 150 / (box4Use[2][2] - box4Use[0][2])   # scale = cm / pixel
        pixelDifference = waterlineRow - box4Use[2][2]
        level = 305.0 - (pixelDifference * scale) / 100
    elif position[0] and position[1]:               # 306,5 and 306
        scale = 50 / (box4Use[1][2] - box4Use[0][2])    # scale = cm / pixel
        pixelDifference = waterlineRow - box4Use[1][2]
        level = 306.0 - (pixelDifference * scale) / 100
    elif position[1] and position[2]:               # 306 and 305
        scale = 100 / (box4Use[2][2] - box4Use[1][2])   # scale = cm / pixel
```

```
            pixelDifference = waterlineRow - box4Use[2][2]
            level = 305.0 -  (pixelDifference * scale) / 100
        elif position[0] and not position[1] and not position[2]:
            scale = 4 / box4Use[0][3]                # scale = cm / pixel, the bounding box is
                                                       approximately 4cm high
            pixelDifference = waterlineRow - box4Use[0][2]
            level = 306.5 - (pixelDifference * scale) / 100
        elif position[1] and not position[0] and not position[2]:
            scale = 4 / box4Use[1][3]                # scale = cm / pixel, the bounding box is
                                                       approximately 4cm high
            pixelDifference = waterlineRow - box4Use[1][2]
            level = 306.0 - (pixelDifference * scale) / 100
        elif position[2] and not position[0] and not position[1]:
            scale = 4 / box4Use[2][3]                # scale = cm / pixel, the bounding box is
                                                        approximately 4cm high
            pixelDifference = waterlineRow - box4Use[2][2]
            level = 305.0 - (pixelDifference * scale) / 100
        else:
            level = 0.0
    else:
        level = 0.0

  if level != 0.0:
    return level, "Water level detected"
  else:
    return level, "No water line could be detected"

# Load the custom trained Yolov5m models. Alternative 3 produced the best performing
# model.
modelStaff = torch.hub.load(staffGaugeDirectory, 'custom', path=staffGaugeDirectory +
             '/staffGauge/staffGaugeBest.pt', source='local')
# modelNumbers = torch.hub.load(numbersDirectory, 'custom', path=numbersDirectory +
                       '/Numbers/best.pt', source='local')          # used in alternative 1 & 3
modelNumbers = torch.hub.load(numbersDirectory, 'custom', path =waterlineDirectory +
                       '/Waterline/combinedBest.pt', source='local')   # used in alternative 2
# modelWaterline = torch.hub.load(numbersDirectory, 'custom', path =waterlineDirectory
                       + '/Waterline/combinedBest.pt', source='local')
                                                                  # used in alternative 2 & 3
modelWaterline = torch.hub.load(numbersDirectory, 'custom', path =waterlineDirectory +
                      '/Waterline/firstBest.pt', source='local')       # used in alternative 1

# Set allowed confidence limit and Iou (intersection over union)
# The IoU value is used to filter out unwanted overlapping bounding boxes
modelStaff.conf = 0.1  # confidence threshold (0-1)
modelStaff.iou = 0.9  # NMS IoU threshold (0-1)
modelNumbers.conf = 0.50  # confidence threshold (0-1)
modelNumbers.iou = 0.85  # NMS IoU threshold (0-1)
modelWaterline.conf = 0.01  # confidence threshold (0-1)
modelWaterline.iou = 0.85  # NMS IoU threshold (0-1)
```

```
# Get all the image file names in the folder
images = [f for f in os.listdir(directory) if f.endswith('.jpg')]

# Open a file for storing the results
f = open(resultDirectory + 'inferenceResultsModel02_kontroll_alt_3.csv','w')

# Loop through the images in the image folder
for imgName in images:
   # Define a new filename for the newly saved file
   resFile = resultDirectory + "/Res_" + imgName

   # Locaction of image
   imgIn = directory + imgName

   # Run the detection and store in resultsStaff
   resultsStaff = modelStaff(imgIn)

   # Get coordiates for bounding box
   coordinates4Box  = resultsStaff.xyxy[0]

   # Initilaize variables
   staffGaugeFound = "False"
   numbersFound = "False"
   waterLineFound = "False"
   numbers = 0
   level = 0.0

   # If staff gauge is found
   if len(coordinates4Box) > 0:
      # Staff gauge found
      staffGaugeFound = "True"

      # Crop image and straigthen it
      img, angle = getStaffAngle(imgIn, coordinates4Box)
      imgRotated = rotateImage(img, -angle)

      # Get the cropped and straightened image size
      h, w, _ = imgRotated.shape
      imgSize = (h,w)

      # Run the numbers detection and store in resultsNumbers
      resultsNumbers = modelNumbers(imgRotated)

      # Get coordinates for numbers bounding boxes
      coordinates4Boxes  = resultsNumbers.xyxy[0]
```

```
# Check if coordinates4Boxes contain numbers (extra check when the combined
# model is used to get numbers reference correct)
for row in coordinates4Boxes:
   if row[-1].item() < 3.0:
      numbers = 1

# Run the water line detection and store in resultsWaterline
resultsWaterline = modelWaterline(imgRotated)

# Get coordinates for water line (and numbers) bounding boxes
coordinates4WaterLineBoxes  = resultsWaterline.xyxy[0]

# If numbers on staff gauge is found... used as water level reference
if len(coordinates4Boxes > 0) and numbers == 1:

   # Numbers found
   numbersFound = "True"


   # Crop the image again to remove the background found at the side of the
   # previously cropped image
   imgCropped = cropAgain(imgRotated, angle)

   # Check if water line bounding box found and if so, calculated the coordinated of
   # the midline of the box
   # The row is stored at which the water line was found
   for i in range(len(coordinates4WaterLineBoxes)):
      if coordinates4WaterLineBoxes[i][5] == 3:
         row = (coordinates4WaterLineBoxes[i][1] +
                  coordinates4WaterLineBoxes[i][3])/2
         # Convert tensor value to float
         row = round(float(row.item()))
         waterLineFound = "True"
         break
      else:
         row = 0

   # Calculate the water level based on the found number boxes and the row where
   # the water line was found on.
   # The water level is 0.0 if no water level could be calculated
   level, comment2 = calculateWaterLevel(coordinates4Boxes, row,
                                                imgRotated.shape[:2])
   level = round(level, 2)

   # Add lines to the image where the water line is located
   cv.line(imgRotated, (0, row), (w, row), (255, 100, 100), 2)   # Red
```

```
            # Add text to the image stating the water level
            cv.putText(imgRotated, str(level), (5, row + 40), cv.FONT_HERSHEY_SIMPLEX, 1,
                                                                (255, 100, 100),  2, cv.LINE_AA)

        # Write the result to a csv file
        data = resFile + ", " + staffGaugeFound + ", " + numbersFound +", " + waterLineFound
                                                                        +", " + str(level) + "\n"
        f.writelines(data)

# Close the csv file
f.close()
```

Appendix C – Code for image filtering

```
# imageCheck.py
# Check the image quality and discard the images that are not found to be suitable
# The code will filter out images as night-time and blurred and move these images
# to separate folders
# Version 1.0
# Author: Bo-Anders Näs - 2023

import cv2 as cv
import numpy as np
import shutil
import os

def calculateSTDDarkChannel(image, windowSize):
    # Convert the image to a floating-point data type and normalize the values
    img = image.astype(np.float64) / 255.0

    # Calculate the minimum value in each local window
    darkChannel = np.min(img, axis=2)

    # Create a structuring element for the filter
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (windowSize, windowSize))

    # Apply the minimum filter to the dark channel
    darkChannelFiltered = cv.erode(darkChannel, kernel)

    # Calculate the standard deviation of the dark channel prior
    stdValue = np.std(darkChannelFiltered)

    return stdValue


# Assign directories
# Directory containing all the images that are to be checked
directory = 'D:/Images to be checked/'

# Destination directory for images that are discarded
destinationNight = directory + 'Night time/'
destinationBlurred = directory + 'Blurred/'

# Get files in the directory
FileList = os.listdir(directory)

# Loop through all images (.jpg) in the directory
for filename in FileList:
    # Only look for image files ending with .jpg
    if (filename.endswith(".jpg")):
```

```
# Read image file
img = cv.imread(directory + filename)

# Calculate mean intensity value of all pixels in image
meanIntensity = np.mean(img)

# Calculate mean value of saturation channel of image in HSV color space
imgHSV = cv.cvtColor(img, cv.COLOR_BGR2HSV)
_, s, v = cv.split(imgHSV)

# Calculate RMS of saturation channel of image in HSV color space
rmsSaturation = np.sqrt(np.mean(np.square(s)))

# Calculate maximum inter-pixel difference
maxdifference = np.max(v) - np.min(v)

# Calculate gradients using the Sobel operator on a part of the image
height, width = img.shape[:2]
imgSlice = img[0:int(height/3), 0:int(width/2)]

# Convert to grayscale
imgSliceGray = cv.cvtColor(imgSlice, cv.COLOR_BGR2GRAY)

# Calculate the gradients using the Sobel operator
gradientX = cv.Sobel(imgSliceGray, cv.CV_64F, 1, 0, ksize=3)
gradientY = cv.Sobel(imgSliceGray, cv.CV_64F, 0, 1, ksize=3)

# Calculate the gradient magnitude
gradientMagnitude = np.sqrt(gradientX**2 + gradientY**2)

# Calculate the average gradient magnitude
AVG_gradientMagnitude = np.mean(gradientMagnitude)

# Calculate dark channel prior of part of the image
window_size = 15
stdValue = calculateSTDDarkChannel(imgSlice, window_size)

# Filter the images
if (rmsSaturation < 0.03 and maxdifference > 190 and meanIntensity < 170):
   # Move the night time image to a separate folder
   shutil.move(directory + filename, destinationNight + filename)
elif AVG_gradientMagnitude < 24.5 and stdValue > 0.08:
   # Move the blurred image to a separate folder
   shutil.move(directory + filename, destinationBlurred + filename)
```