1 Vision

In this section a complete walk through of the thoughts, methods and results of the different vision algorithms, that has been implemented are analysed and tested to locate a breach. The methods that will be analysed in this section include: Fourier Transformation to find the rectangular pattern in a fence structure, image segmentation using U-Net deep-learning method, a canny edge detection as well as a convolutional approach to segment the fence from the background and lastly a Convolutional Neural Network.

1.1 Analysing the different sensor options

Using a LiDAR could prove to be a powerful tool and a very interesting approach to follow. However, with the given drone, which has a current weight of 2500g, see Appendix **??**, and a maximum takeoff weight (MTOW) of 3200g, this lead to a total added weight from external sensors of: $3200g - 2500g = 700g$. For the lower end price of LiDAR's it is not uncommon that it weights over 1kg. Which is undesirable. It is possible to find a LiDAR below 700g. However, they come with a cost which is not suitable for this project. Thus, no further exploration of the use of LiDAR will be conducted.
A cheap and viable solution would be to use a Intel Realsense D435 Depth Camera. This will give access to multiple sensor within this one device, as well as a cheap cost of below 2500 DKK. It will grand access to a mono-RGB camera, depth sensor and even stereo cameras if needed [**Intel_RS_price**].

1.1.1 Depth Camera Analysis

Depth camera is a powerful tool which provides depth and distance information for the objects in the images. The basic principle of depth camera is either projecting IR or structured light to get depth information or using closely placed cameras for capturing and comparing images (Stereo setup). Depth images are relatively easy to process, because of the depth information associated with the pixels. We tested the Intel RealSense D435 Depth Camera which uses active IR stereo to generate depth images.

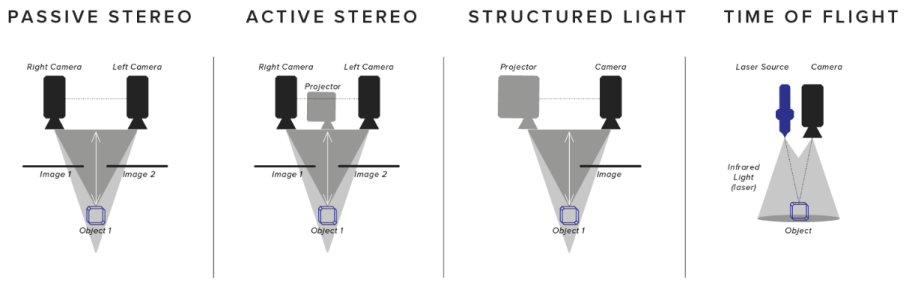


Figure 1: Depth Sensor Configurations

Initially the idea of using the depth camera was to generate a point cloud only consisting of points between 1 and 4 meters away. These point would then be used to reconstruct the grid structure of the fence and a breach could be detected. In theory this would work, but as we discovered, when dealing with actual data from the sensor, the data was unusable for this approach, see Figure 2.

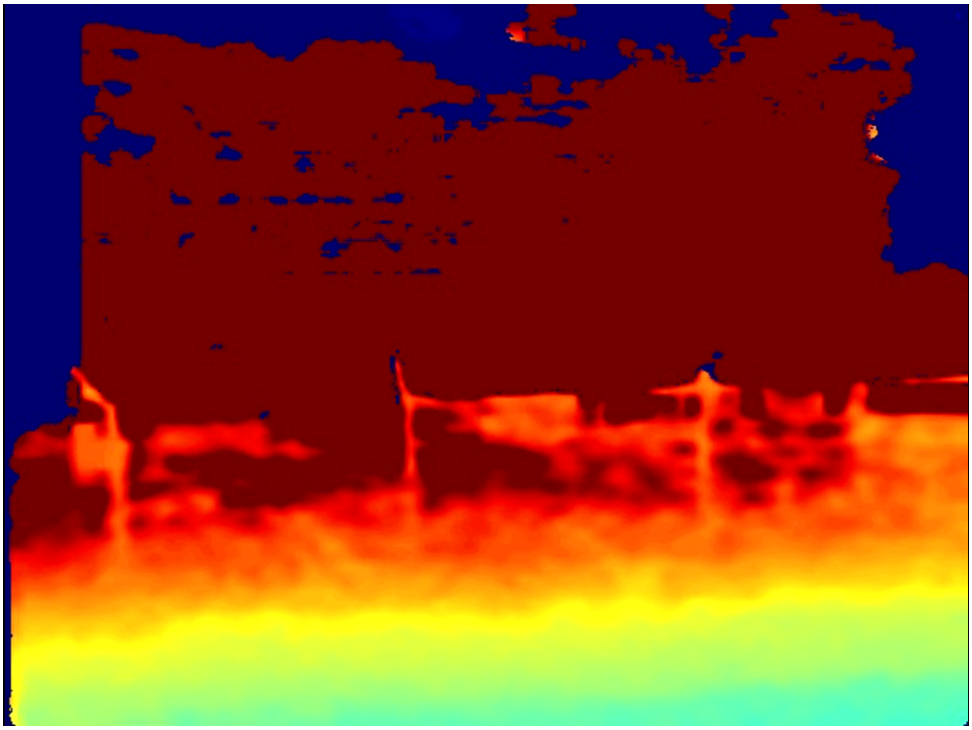


Figure 2: Real life data from depth sensor

This data is not usable for our approach, however a new hypothesis was made, that using some mean value distance of multiple points within the center of the image could perhaps be utilised to maintain a certain distance from the fence at all times.

1.2 Segmentation of The Image

This section will go through and analyse four different approaches to segment an image. During segmentation the goal is to remove as much as the background as possible, without compromising the fence structure. This is required to leave as little as possible to interfere with the screening algorithm. See subsubsection 1.3.1.

This algorithm will systematic go through the grid structure of the fence, locating intersections and detecting obstructions in this structure to verify a breach and its size.

1.2.1 Fourier Transformation

A way to separate the background from the fence was to use the Fourier Transformation which can decompose an image into its sine and cosine components. The output will be given in the frequency domain with the input given in the spatial domain. In this frequency domain each point represents a given frequency in the spatial image domain.

The two-dimensional Discrete Fourier Transformation (DFT) and its inverse is given in Equation 1.1 respectively.

$$F(k,l) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f(i,j)e^{\tau 2\pi\left(\frac{ki}{N}+\frac{lj}{N}\right)} \qquad f(a,b) = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} F(k,l)e^{\tau 2\pi\left(\frac{ka}{N}+\frac{lb}{N}\right)} \tag{1.1}$$

Here the $f(a,b)$ is the image in the spatial domain with the exponential as the basis function that corresponds to every point $F(k,l)$ in the Fourier transform. Hence the value for each point $F(k,l)$ is obtained by a multiplication of the spatial image with the corresponding base function and summing the results. The $F(0,0)$ is the DC-component (average brightness) and $F(N-1,N-1)$ is the highest frequency. The term $\frac{1}{N^2}$ is used as a normalisation term. The Fourier Transform is

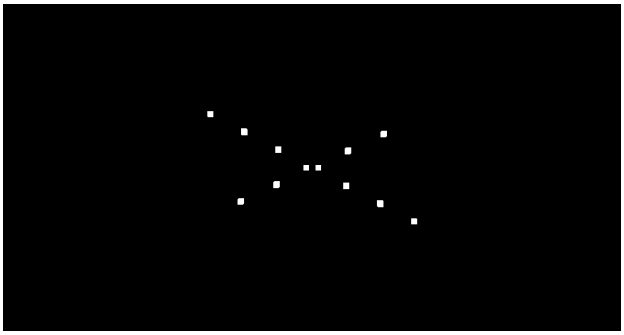
separable therefore it can be rewritten as seen in Equation 1.2.

$$f(a,b) = \frac{1}{N}\sum_{b=0}^{N-1} P(k,b)e^{\tau 2\pi \frac{lb}{N}} \qquad P(k,b) = \frac{1}{N}\sum_{a=0}^{N-1} f(a,b)e^{\tau 2\pi \frac{ka}{N}} \tag{1.2}$$

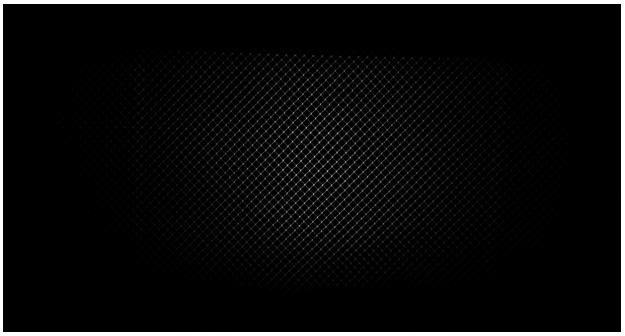
Calculating a two-dimensional Fourier Transformation can be done using a series of two one-dimensional calculations. This decreases the amount of required computations, yielding a $O(N^2)$ computational time. To improve further a Fast Fourier Transformation (FFT) could be used, which reduces the time complexity to $O(Nlog_2N)$. This was one of the main reasons for choosing the Fourier Transformation to segment the fence from the background. Since this enables the calculations to be done very fast and utilises the possibility of detecting breaches closer to real-time.[**FFT**]

Analysis:
The performance evaluation of the FFT will be based on images in different configurations. This is done to stress test the algorithm, in order to find its limitations. The code used for this algorithm was heavily inspired by a script given from Henrik Skov Midtiby (*hemi@mmmi.sdu.dk*) which involves FFT in Python and OpenCV. The procedure will follow the approach described in Section 1.2.1. Given the image in Figure 3a, the Fourier spectrum was found and visualised in Figure 3b. Here the structure of the fence is nicely highlighted as seen in the frequency spectrum of the FFT. The mask is now multiplied with the input image and the results can be seen in Figure 3c.

(a) Input image of fence without breaches.



(b) Fourier spectrum of input image (mask).

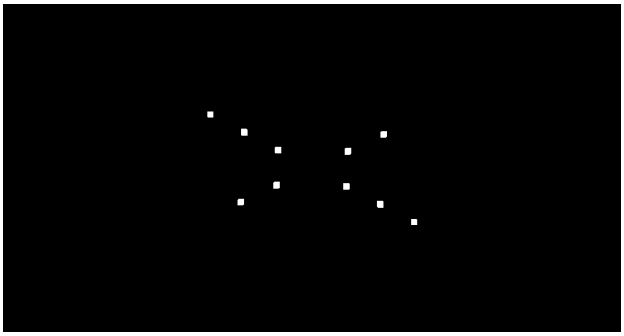


(c) Multiplication of mask and input image.

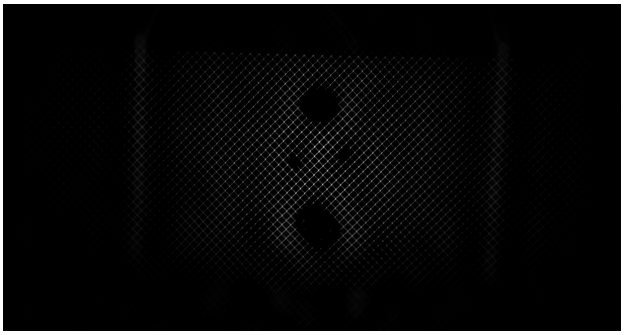
Figure 3: Illustration of the working of the Fourier transformation resulting in a background subtraction of the image in Figure 3c

The results is a segmented image where only the structure of the fence is visible. It may the noticed that this image is underexposed which results in a fine separation of the foreground (fence) and background (forest). This image is taken under ideal conditions in regard to optimising the performance of the FFT.

(a) Input image of fence with breach.



(b) Fourier spectrum of input image (mask).

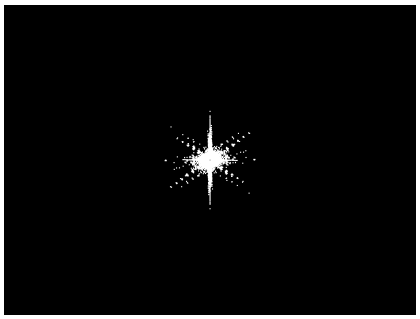


(c) Multiplication of Mask and input image with breaches

Figure 4: Input image with breaches and the resulting background subtracted image suing Fourier transformation

Now giving an input image with customised breaches, the Fourier spectrum can be seen in Figure 4c. Applying this mask on the input image the resulting segmentation can be seen in Figure 4b. From this result it is possible to detect the breaches. The algorithm made to conquer this problem can be seen in Section 1.3.1.

(a) Input image of fence



(b) Fourier spectrum (mask)

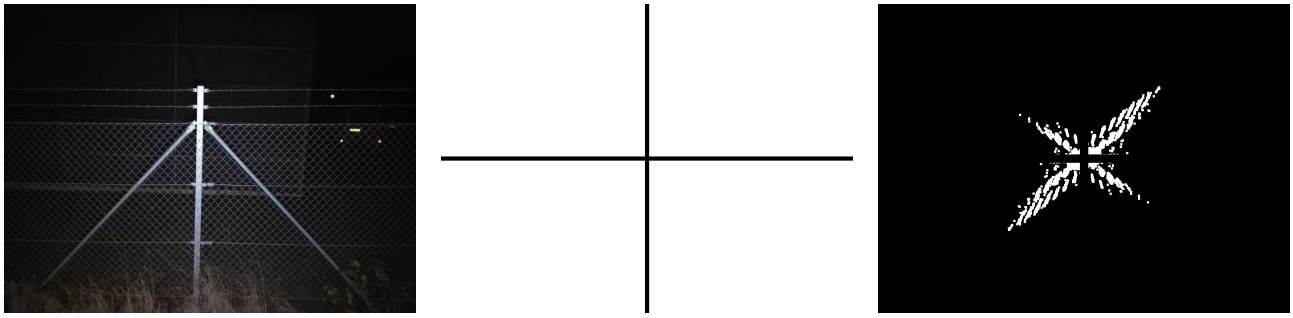
(c) Segmented image

Figure 5: Input image and the resulting background subtracted image using Fourier transformation

Because the FFT takes into account certain patterns in the image, it is highly sensitive to variations in light, background and other structures which shows some kind of symmetry. Especially if light is projected on structures in the image which is of no interest. Such an image can be seen in Figure 5a. Here the fence is hard to detect with a gray background. Moreover, the light is not projected on the fence the same way as in Figure 3a. This make the problem considerably more challenging. This Fourier spectrum can be seen in Figure 5b. One may notice that the Fourier spectrum includes a lot more frequencies then the ones got in Figure 4c. As it can be seen in Figure 5c, the background is not properly removed from the input image yielding a very bad segmentation.

Ways to deal with this problem could be to project a light source on the drone reflecting upon the fence. This could make

a better separation of the fence from the background. Furthermore, the mask in the Fourier spectrum could be multiplied with a user defined mask excluding the frequencies which have no interest. A visualisation of these improvements can be seen in Figure 6.



(a) Light projected on the fence *(b) User defined mask* *(c) Resulting Fourier spectrum*

Figure 6: Input image and the resulting background subtracted image using Fourier transformation

Unfortunately, these improvements did not result in a general improvement of the segmentation of the image. The spectrum in the FFT still has to be tuned to fit the given conditions from where the image is taken. This is a considerable drawback of using this algorithm for segmentation.

1.2.2 Segmentation using U-Net

Another approach for using neural networks, is the fence segmentation using neural networks such as U-Net and then performing the breach detection using the fence re-construction methods discussed in the previous sections. U-Net configuration is particularly good at segmenting images. A typical U-Net used for image segmentation is shown in Figure 7.

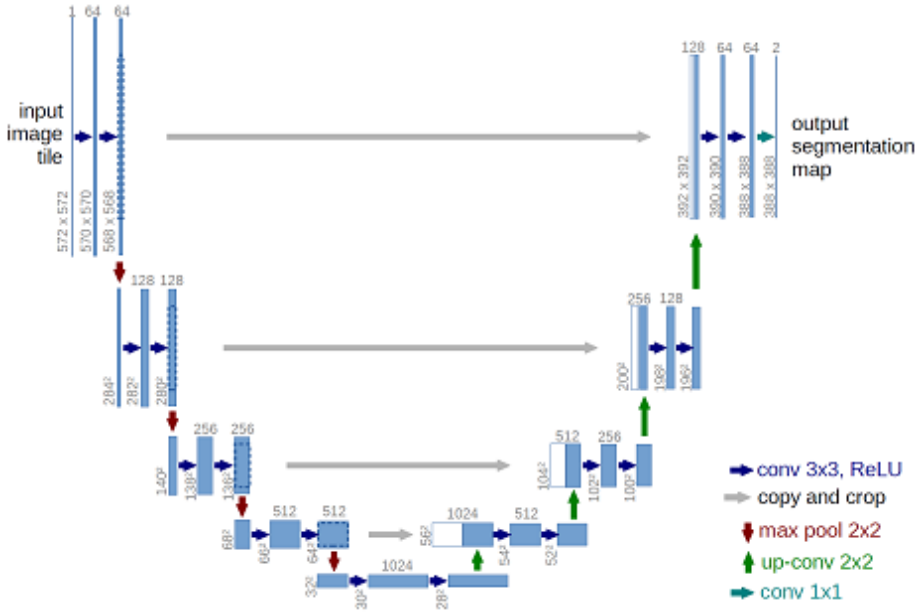


*Figure 7: U-Net Architecture [***UNET_fig***]*

For each pixel in the input image the U-Net outputs a binary pixel classifying it as either background or foreground. This configuration, if properly trained, could provide a very strong segmentation tool. However, to train the U-Net hundreds of different fence images where needed with specified ground truth or mask of the image. This data was not readily available in our case. However, using generated artificial data sets, some level of performance from the U-Net approach could be achieved.

An example of the U-Net input image and ground truth is shown in Figure Figure 8.

Figure 8: U-Net Input Image and Mask

1.2.3 Canny edge detection

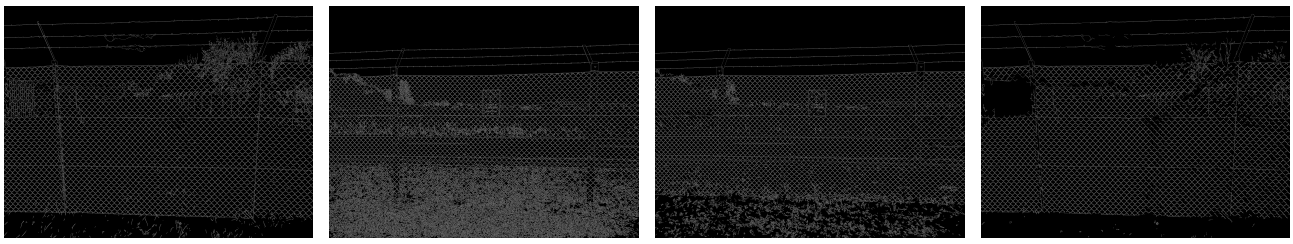
One of the first possible solutions tried was using a simple Canny edge detector to detect the edges of the fence in the hope of being able to segment the fence from the background.
The method consisted of extracting the blue colour channel, smoothing the image with a Gaussian blur and use the Canny edge detector on the image. The reason for choosing the blue image channel was because the vegetation in the background showed up less on the blue channel compared to green, red and gray, while the fence structure remained as clear.

In Figure 9 two input images can be seen and these images will be used for test and comparison of this method. The Canny edge detector needs a lower and an upper threshold as parameters.

(a) Input image one *(b) Input image two*

Figure 9: Input images for the Canny edge detection

The method was tuned individually for each image and then run on both images. The result of this can be seen in Figure 10.



(a) Input one, tuned for one *(b) Input two, for one* *(c) Input two, tuned for two* *(d) Input one, tuned for two*

Figure 10: The respective output images for the two inputs run on both tunings

It can be seen that the tune for the darker of the two input images performs the best on both images where the tune for the lighter image performs significantly worse on both images. It was found that this method required under exposed images to perform its best, but still struggles with vegetation or other structures in the background. The method also required a lot of fine tuning to produce a satisfactory result that varies a fair bit between input images.
This method was never really considered but were used as a comparison to see how well other methods fared against it, and to start developing breach detection algorithms.

1.2.4 Convolution based segmentation

This method was based on a different idea. Instead of trying to segment the fence structure itself, the intersections in the fence structure would be detected instead resulting in a list of pixel coordinates. A breach detection algorithm could then be developed to find breaches in the structure of pixel locations.

Different approaches was considered but a convolution based solution was chosen. The algorithm would run sequential on each each step based on the blue colour channel. A 33 x 33 kernel was made to generate a high response when being in the center of an intersection. This kernel can be seen as an image in Figure 11. Black correspond to negative contribution, gray zero and white to positive contribution. This kernel was convoluted with the image to produce a response map featuring high responses were an intersection was found. This image was then normalised to values between 0 and 255 and then thresholded. Contours will then be found in the thresholded image and the centroids of those returned as the final pixel position.

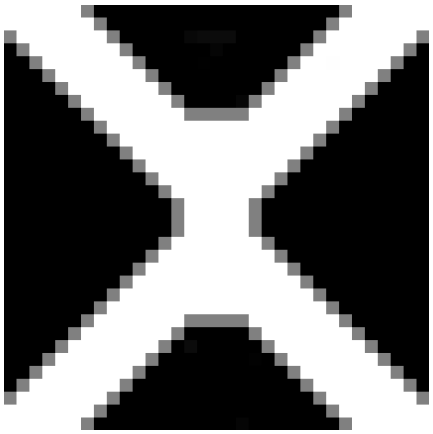


Figure 11: The kernel used for the method

In Figure 12 the different stages of the process can be seen. On the left is the blue colour channel as input, next to that the response from the convolution, then the thresholded image and finally the centroids for the found contours drawn onto the original image.
The tuning in this algorithm lies in the thresholding values for the binary thresholding. The methods was tuned to perform its best on the input image.

(a) Input blue channel *(b) Output convolution* *(c) Output thresholding* *(d) Output pixel positions*

Figure 12: Illustration of the processing steps from start to finish

In Figure 13 another lighter image was run though the method and was again tuned to give the best result. It was generally found that the method were relatively forgiving regarding the thresholds as long as the images were underexposed.

(a) Input blue channel *(b) Output convolution* *(c) Output thresholding* *(d) Output pixel positions*

Figure 13: Illustration of the processing steps from start to finish

The method handled grass really well but struggled a bit with tress. The result from this would need some filtering of the found points to be useful for detecting breaches.

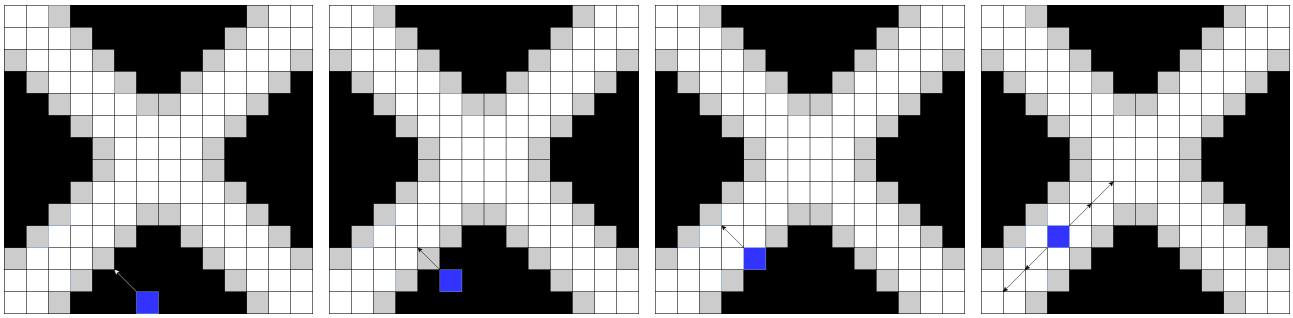
It was considered to maybe connect the points with lines to produce a perfect grid structure that could then be passed to the breach detection algorithm described in subsubsection 1.3.1.

1.3 Breach detection

1.3.1 Screening of segmented images

Given segmented images where the background has been removed with only the foreground (fence) left in the image, the resulting image can be screened for possible breaches. This is done using an algorithm with the following three stages namely *searching for a fence*, *fence following* and *peak and endpoint detection*. The peaks (junctions) is defined as the intersections where the elements of the fence meets to form a junction. This is the midpoint of the images seen in Figure 14.

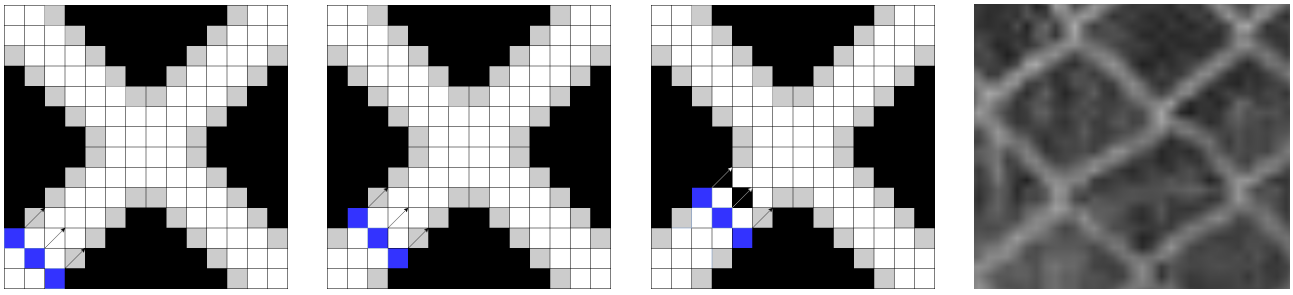
The basic idea is to find the fence starting from a random location in the middle of the image. The pointer can move in an either up-right, down-right, down-left or up-left direction. This is done so that the pointer initially ends up on the structure on the fence, which it now can follow. This is illustrated in Figure 14 where the fence is white, background black and the pointer labelled blue.



(a) Step one *(b) Step two* *(c) Step three* *(d) Step four*

Figure 14: Illustration of searching for a fence in the image. Fence (white), background (black), white arrows (illegal stopping criteria) and black arrows (valid stopping criteria)

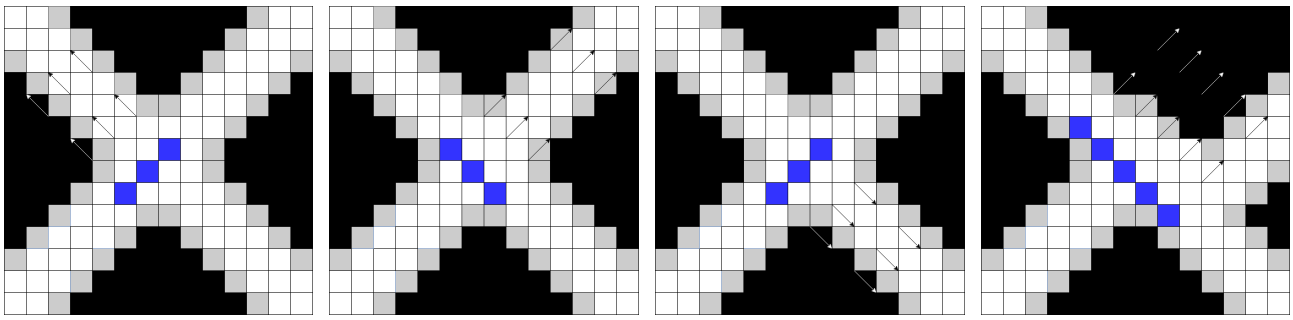
The pointer continues to move in a certain direction until the giving pixel value is above a threshold. When this happens, the pointer will now move in either two directions which is orthogonal to the last move direction. This can be seen in Figure 15. Now the pointer moves in a direction with a chosen step size for each move. The step size can be given as a list of values so that the pointer can move further away from its last position in each step. This is done to take into account low pixel values in the next step even thought the pointer is still on the fence. Moreover, the next step of the pointer can be moved in orthogonal directions according to the last move if an abnormal fence structure is seen like the one in Figure 15d.



(a) Step one *(b) Step two* *(c) Step three* *(d) Abnormal fence structure*

Figure 15: Illustration of the line (fence) follower approach. Fence (white), background (black), white arrows (illegal stopping criteria) and black arrows (valid stopping criteria)

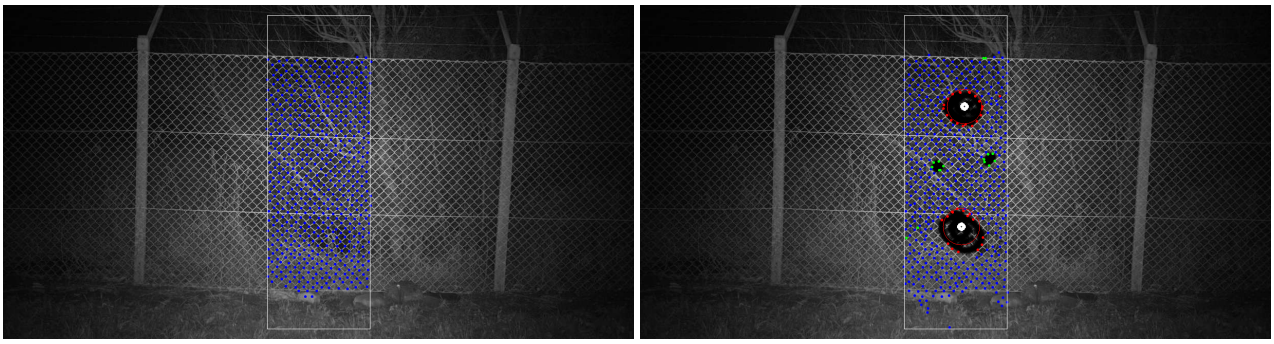
The pointer will continue to move until a peak location or end of fence is seen based on the pixel values. The way the algorithm detects peaks is illustrated in Figure 16. At every step in the fence following approach the algorithm will check for high pixel values in front of it as well as its orthogonal directions according to the last move. If the pixel value in all three directions is above a threshold it will be considered to be a peak location. The point in the image coordinates will be put on a list of already detected peak points. Now the algorithm will branch in the other directions at the peak location to continue its search for peaks and endpoints. This can be done either in an recursive or iterative manor.



(a) Step one *(b) Step two* *(c) Step three* *(d) Step four*

Figure 16: Illustration of searching for a peak (junction) in the fence. Fence (white), background (black), white arrows (illegal stopping criteria) and black arrows (valid stopping criteria)

Endpoints is detected if the fence following part of the algorithm sees only black pixels (pixels below a threshold) in the next steps to come. Hence, the algorithm will return from its given branch and put the endpoint on a list to be used in the analysis of breaches in a later step.
To make the algorithm faster a region of interest (ROI) was defined which can be seen in Figure 17. The recursion will stop if it goes beyond the ROI. As it can be seen, large custom made breaches can be found and small breaches or noise are neglected. The endpoints inside the ROI is grouped together using hierarchical clustering. This approach finds the distance between points. It will group points based on the distance from a given point to that of the mean of a cluster. This is both a fast and effective method in order to cluster points and make an approximation of the size of the breach knowing the cluster diameter.



(a) Fence without breaches *(b) Fence with custom made breaches*

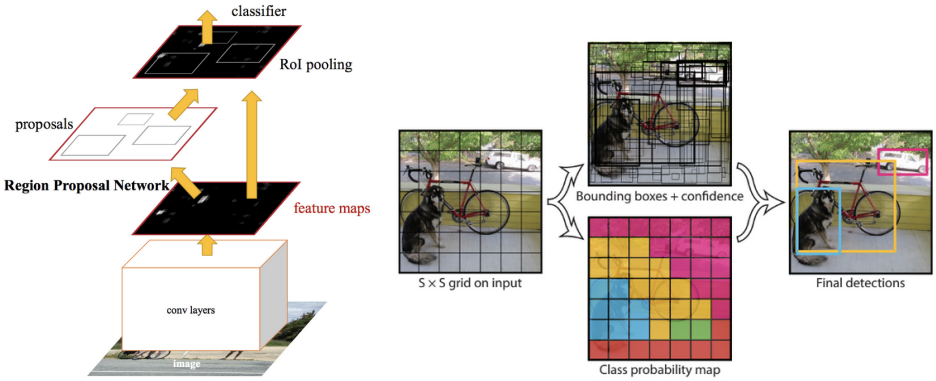
Figure 17: Illustration of screening a fence given images from the Fourier transformation. Peaks (blue), breaches (red), neglected small breaches or noise (green) and ROI (white)

The algorithm performs quite well given a nicely segmented input image. However, because the implementation is based of the need of a really good segmented input image, the algorithm will fail if the input is bad e.g. given a bad segmented image from the Fourier transformation. This is due to the fact that the pointer will follow the structure of the fence, but if the fence is badly segmented from the background, this algorithm will not work.

In theory, it is possible to find very small breaches using this screening approach of the image because it tries to find every peak and endpoint location in the image. The limitations is based on the need for a perfectly segmented input image.

1.3.2 Image classification and boundary box regression using mask-RCNN

A number of algorithms where taken into consideration in regard to deep learning. Faster-RCNN is an extension of the region-based convolution neural network (RCNN). The RCNN uses the selective search algorithm to extract proposal regions from the image from where these are given to a convolution neural network (CNN) which extracts a feature vector from the regions in question. Then these are given to a set of class specific support vector machines (SVM) where classification and offsets to boundary box regression is performed. Instead of using the selective search algorithm, the faster-RCNN makes a convolution feature map from the input image using a convolution network. A separate network is then used to find region proposals. This predicted region is reshaped using a region of interest pooling layer to classify the picture being within the proposed region and offsets values for the giving boundary boxes. **Mask-RCNN**, Figure 18a, extends to this idea by adding an additional branch being the objects mask. This enables the possibility to make instance segmentation to the found objects.[**FasterRCNN**] [**MaskRCNN**] [**prosConsDeepLearning**]



*(a) The faster-RCNN algorithm (b) The YOLO algorithm[**prosConsDeepLearning**] [**FasterRCNN**]*

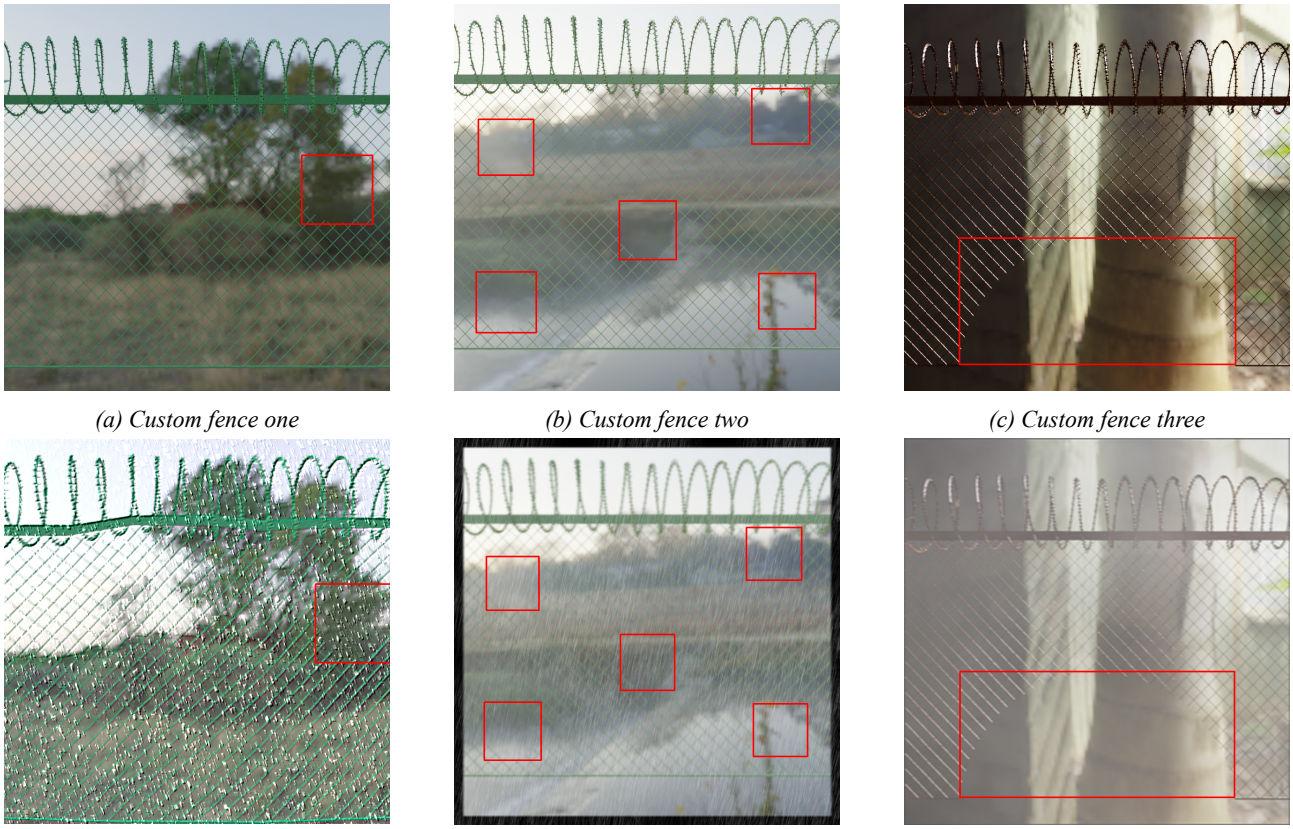
Figure 18: Illustrations of the considered algorithms to use in deep learning

In **YOLO** (you only look ones), Figure 18b, an image is split up into grids of size $N \times N$ where each grid consists of m bounding boxes. It uses a CNN to predict bounding boxes and class probabilities for each box. This algorithm is quite fast, but its comes with the expense of bad detection of small objects.

Given these methods it has been decided to move on with the Mask-RCNN because the precision is preferred instead of speed and the possibility of using the instance segmentation for estimating the size of the breach in the given image.

Data augmentation

In order to train the network a lot of data have to be collected in order to make a robust solution. Shared data from group 171 in the experts in teams course has been used. This data represents a lot of images where a 3D-model of a fence with artificial breaches has been placed in front of different backgrounds as seen in Figure 19a, 19b and 19c. This includes almost 500 images with fences in different configurations. However, more images are needed, so data augmentation is used. The augmented data takes into account different conditions like weather, lightening, blur and noise in the image. This has been done automatically in python where 15 new images have been made from each original image. Every image has been scaled and blurred differently each time. Moreover, rain, snow, fog, Gaussian noise and perspective transformations has randomly been given to each of these images. The result of three of them can be seen in Figure 19d, 19e and 19f.



(a) Custom fence one *(b) Custom fence two* *(c) Custom fence three*

(d) Custom snow *(e) Custom rain* *(f) Custom fog*

Figure 19: Illustration of some of the custom made fences with different backgrounds. The original form can be seen in Figure 19a, 19b, 19c and its augmented parts in Figure 19d, 19e and 19f respectively

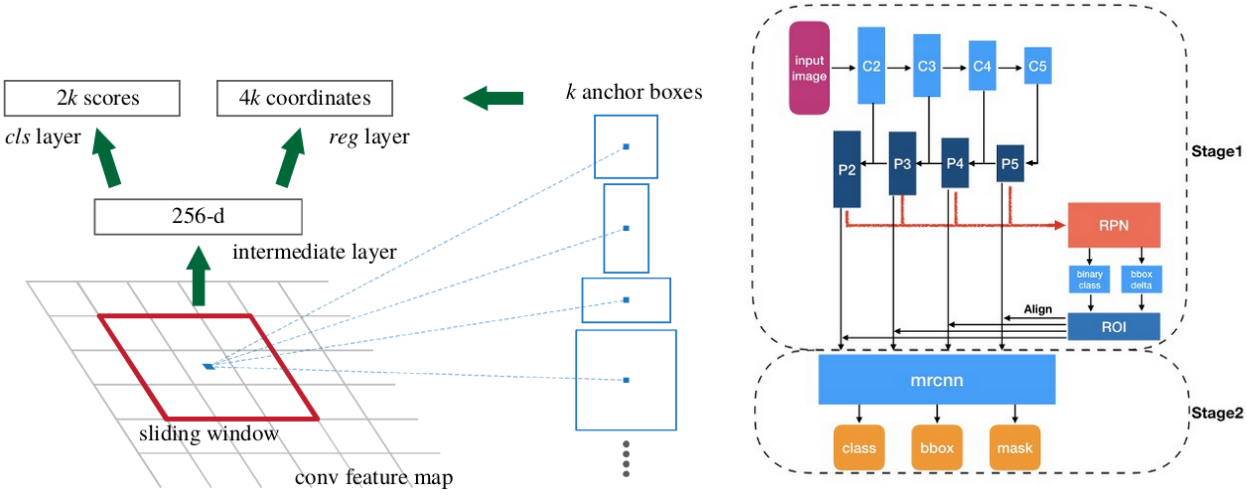
The data augmentation resulted in almost 7000 new images which is going to be used for training the deep neural network (DNN). 80% of the images will be used for training and 20% as validation/test images. The augmentation with snow, rain and fog has been used to take into account real life scenarios when the drone has to fly in different whether conditions and still make a robust estimation of possible breaches in the fences. [**imgaug**]

Mask-RCNN

Mask-RCNN consist of two stages. The first stage begins with the bottom up path where a computation of the backbone (ResNet101 in this implementation) to compute a hierarchy of features at different scales with a scaling factor of two. This is denoted as *C2* (conv2), *C3* (conc3), *C4* (conv4) and *C5* (conv5) in Figure 20b. This illustrates the features activation outputs from the residual block by each last stage. The conv1 is excluded from the pyramid because of the size (memory). The top down path generates feature maps associated to each stage which is denoted *P2*, *P3*, *P4* and *P5* which is called a feature pyramid network (FPN). The use of FPN is due to its maintenance of strong semantic features at various resolution scales which makes the network scale-invariant for object detection.[**FeaturePyramid**]

Now the top down pathway gets scanned for features using a lightweight neural network called region proposal network (RPN) in a sliding window fashion. The features is bound to raw image locations using anchors. An illustration of this can be seen in Figure 20a. The regions which are taking into account are the anchors. At each location of the sliding window, multiple region proposals are predicted according to the number of anchors k. Here the anchor is predicted to belonging to the foreground (possible object) or background. This is done by computing the intersection over region (IoU) between the anchor boxes and ground truth box locations. If the $IoU > 0.5$, it will be considered positive and being an object. If the $IoU < 0.5$, it will be considered negative and being the background. Moreover, the center coordinates (x,y) along with its width and height are predicted yielding $4k$ coordinates. The scores and coordinates are giving to the classification and regression layer respectively. If several of these anchors are overlapping each other, the anchor with the

highest foreground score will be kept using non-max suppression.



*(a) Regional proposal network [***FasterRCNN***]*

*(b) Illustration of the mask-RCNN structure [***MaskRCNNStructure***]*

Figure 20: Illustrations of the RPN and associated anchors in Figure 20a and high level visualisation of the mask-rcnn in Figure 20b

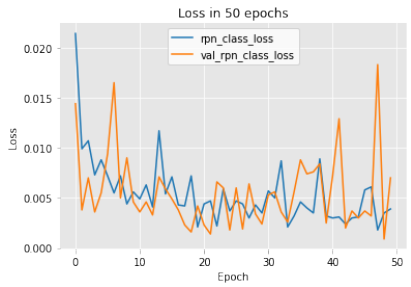
In the second stage the region of interest (RoI) is taken into account. These are the anchor boxes got from the predicted objects from stage one using RPN. However, unlike the RPN which only predicted two classes (foreground/background), the network will now label possible objects to classes or disregard them as belonging to the background. Boundary box refinement will now take the RoIs into consideration and refine the box to encapsulate the possible objects of interest. Now RoI pooling is performed using RoIAlign. This is done to extract a small feature map of the RoI without any quantisation loss. This pooling is necessary because the fully convolution layers (FC) expects a fixed input dimension vector to be used in the last step. Now the classification, boundary box regression and instance segmentation will be evaluated. [**MaskRCNNStructure**] [**InstanceSegmentation**]

The implementation of the mask-RCNN is based upon open source software [**matterportMaskrcnn2017**] which is coded in python. This yields almost an exact implementation of the algorithm based on the original paper [**MaskRCNN**]. The training of the network has been done using Googles online platform, Colab, in order to increase the speed of the training process. The data sets have been uploaded to Dropbox from where the images where fetched to Colab. Every image is associated with an xml file which includes image id, image width and height and the boundary box of the custom made breaches. This is done so that the data can be properly loaded with the information to be used in Tensorflow and Keras. A number of parameters have been set when training the model. The training have been done using the following hyperparameters namely a learning rate of 0.001, momentum of 0.9 and weight decay of 0.0001 within the defined configuration model. Both the configuration and prediction model will be used with a *detection minimum confidence* of 0.7, meaning that the score of a given object prediction must be above a probability of 0.7 in order to be recognised as belonging to a certain class e.g breach in the fence. Furthermore, transfer learning has been used where pretrained COCO weights for the backbone ResNet-101 was implemented. This enables faster training using the custom made dataset [**cocoWeights**]. This implementation of mask-RCNN gives the opportunity of using ResNet50 and ResNet101 which are 50 and 101 layers deep respectively. However, the ResNet101 is used because it have proved to have a better performance. This will come at the expense of a longer training time due to the dept of the network. [**trainClassifier**]

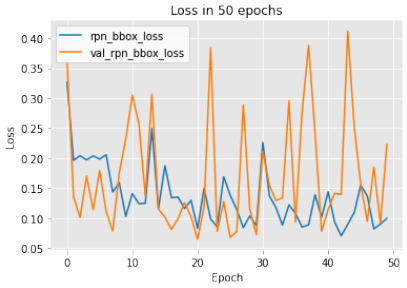
Analysis

The network will be trained with two different datasets namely the original one with 465 images and the augmented with 6975 images. The smaller dataset is given a batch size of 100 and the bigger 200 and both of 50 validation steps per epoch

respectively. The learned weights of the network will be saved for later use. The results of training with the augmented dataset can be seen in Figure 21 and 22.



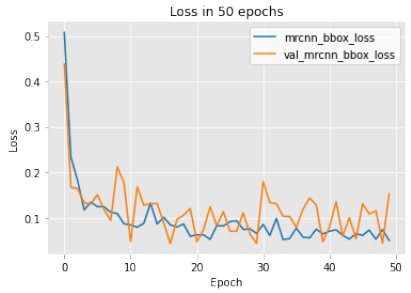
(a) RPN class loss



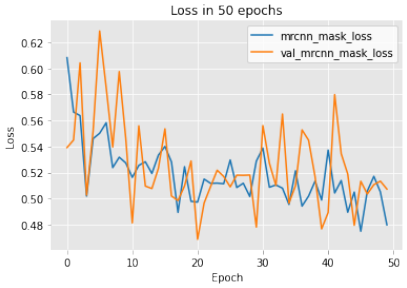
(b) RPN boundary box loss

(c) Mask-rcnn class loss

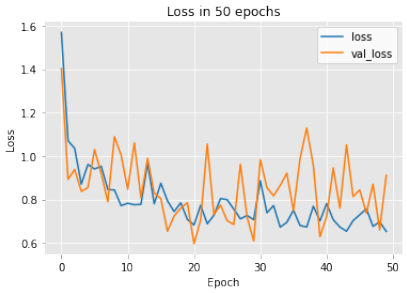
Figure 21: Illustrations of the loss during training in 50 epochs using 6975 images where 80% is used for training and 20% as validation/test



(a) Mask-rcnn boundary box loss



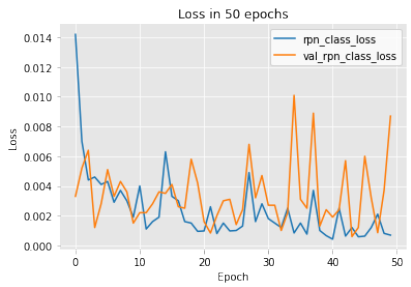
(b) Mask-rcnn mask loss



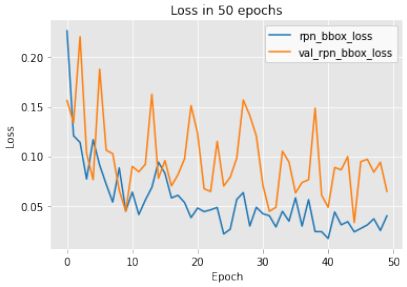
(c) Total loss

Figure 22: Illustrations of the loss during training in 50 epochs using almost 6975 images where 80% is used for training and 20% as validation/test

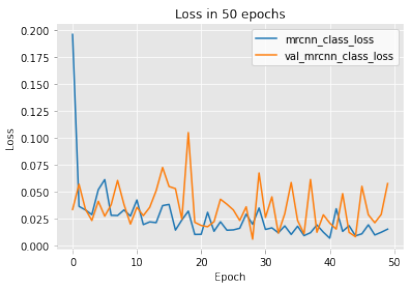
In Figure 22c the total loss can be seen. The training loss is seem to converge. However, the validation loss seems to fluctuate quite a lot. By studying Figure 21b and 22b, it is seen that the mean contributors to the total loss is the RPN boundary box loss and the mask-rcnn mask loss. The reason for the mask loss could be that no mask is given in the dataset yielding no training data for the mask besides the area on the boundary box which is not precise enough to properly train the network in regard to the instance segmentation. It is seen in Figure 21c and 22a that the mask-rcnn class and boundary box loss are converging as inspected with only minor fluctuations.



(a) RPN class loss



(b) RPN boundary box loss



(c) Mask-rcnn class loss

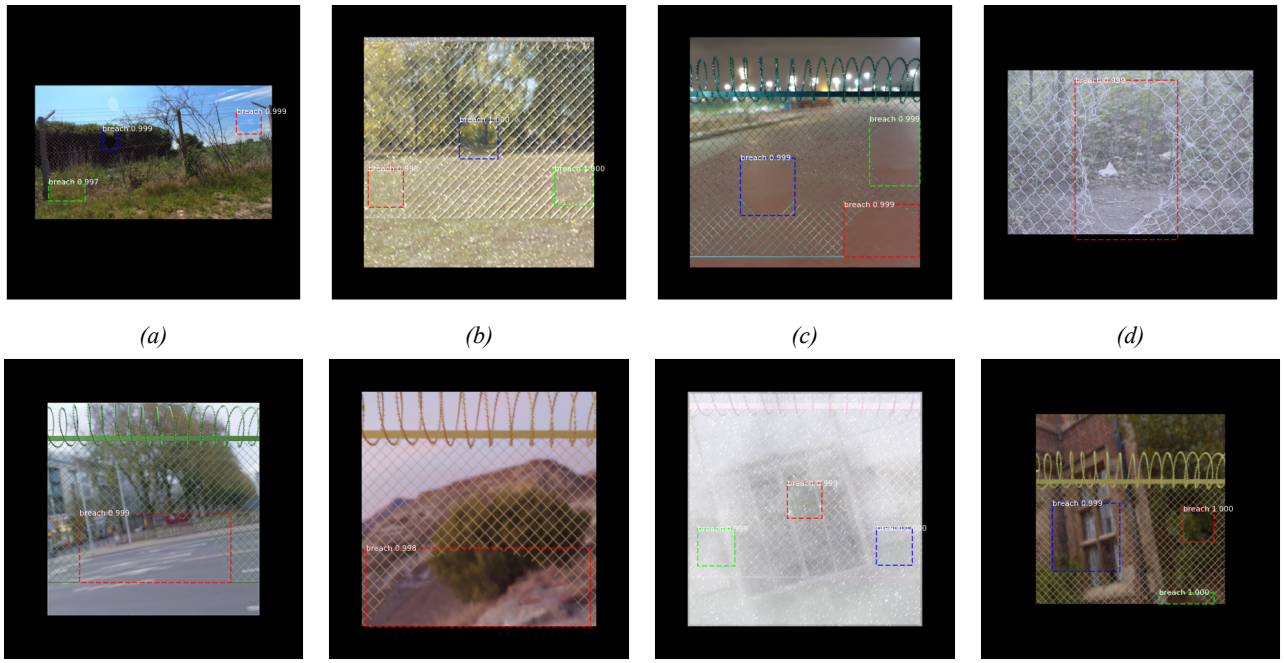
Figure 23: Illustrations of the loss during training in 50 epochs using 465 images where 80% is used for training and 20% as validation/test

(a) Mask-rcnn boundary box loss *(b) Mask-rcnn mask loss* *(c) Total loss*

Figure 24: Illustrations of the loss during training in 50 epochs using almost 465 images where 80% is used for training and 20% as validation/test

In Figure 23 and 24 similar conclusions can be made. Here the network has been trained using the small original dataset consisting of 465 images. It may the notised that the mask-rcnn class and boundary box loss is converging as seen in Figure 23c and 24a.

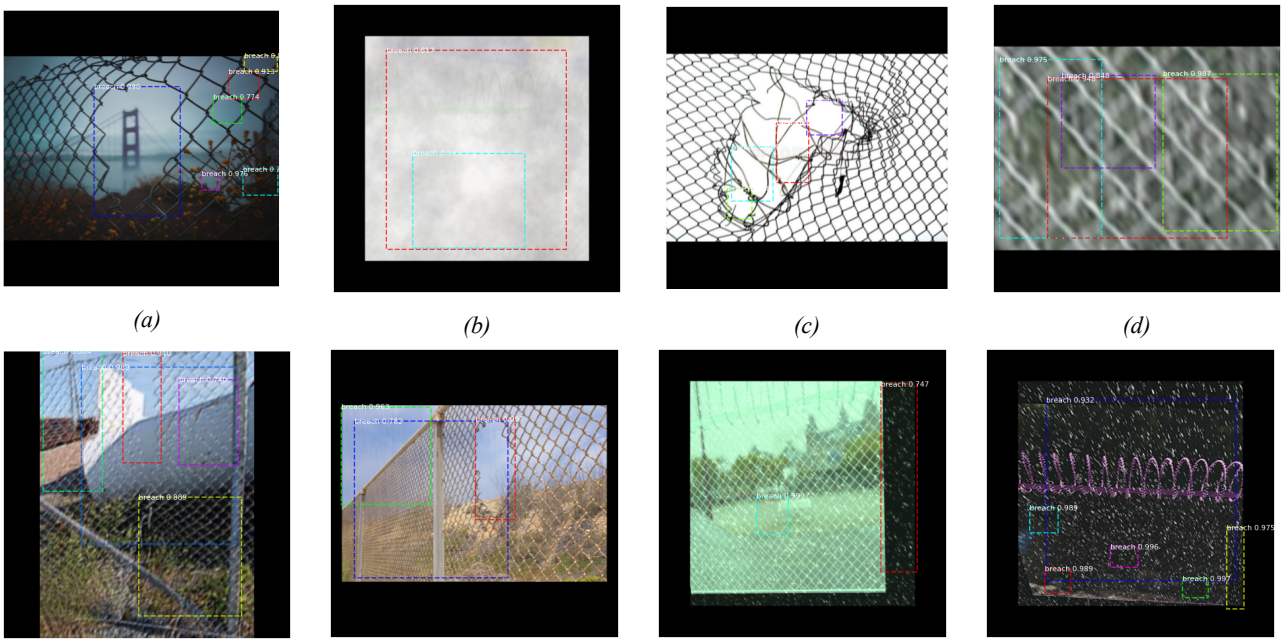
A subset of the augmented validation set along with the predicted breaches in each image can be seen in Figure 25. The breaches have all been found even in difficult conditions. This illustrates that the mask-rcnn performs quite well in estimating the boundary box and the image classification. This has been done using weights gained from training the network with the augmented dataset.



(a) *(b)* *(c)* *(d)*

(e) *(f)* *(g)* *(h)*

Figure 25: Illustration of the mask-rcnn performing well in finding some of the breaches in the validation set. Each breach is encapsulated within a bounding box with a probability score going from zero to one

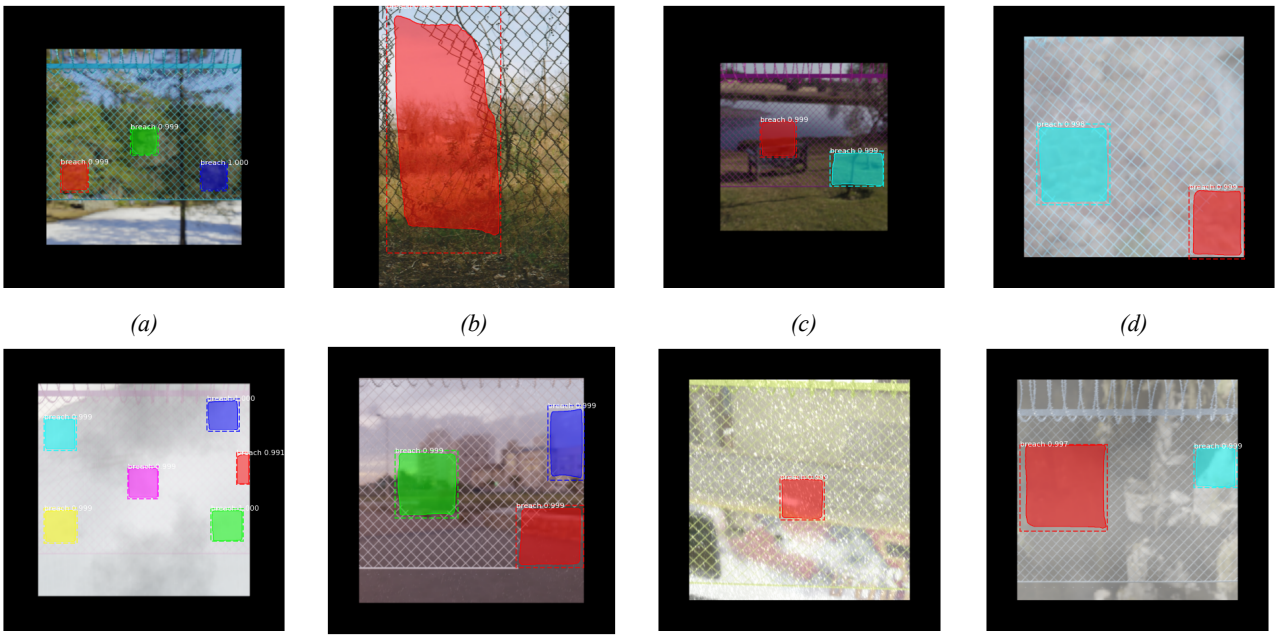


(a) *(b)* *(c)* *(d)*

(e) *(f)* *(g)* *(h)*

Figure 26: Illustration of the mask-rcnn performing bad in finding some of the breaches in the validation set. Each breach is encapsulated within a bounding box with a probability score going from zero to one

In Figure 26 the mask-rcnn do not perform quite as well. It can be seen in the figure that multiple wrong objects have been predicted as belonging to the breach class. However, it must be mentioned that the found breaches in Figure 26a and Figure 26c is taking very close to the fence which will not happen in real life using the drone. The algorithm is actually able to find the breaches, but with many wrong detections as a site note. Most of these could effectively have been sorted out by setting the limit of accepted objects to 0.9 or 0.9995 probability which will be done in **??**. Furthermore, by using a training set which includes many instances of close up fences, the algorithm would properly perform better in these cases.



(a) *(b)* *(c)* *(d)*

(e) *(f)* *(g)* *(h)*

Figure 27: Illustration of the instance segmentation using mask-rcnn. Each color encapsulated within a bounding box represents the area associated to the found breach

The results of instance segmentation can be seen in Figure 27. The area which belongs to the breaches have been nicely separated even in difficult conditions like the one in Figure 27b. However, it must be mentioned that the estimation of some instance segmentations have been seen to fluctuate quite a lot if the algorithm has trouble of finding the breach. But in most cases in works very good. Now by knowing the the distance to the fence, an approximation to the size of the breach can be found by using the instance segmentation if wanted. In Table 1, the different weights of the two different trained networks are shown, where the ”Small dataset weights” are trained from a pool of 465 images, and the ”Big dataset weights” are trained on the augmented dataset of 5580 images.

Table 1: Description of the mean average precision (mAP) using the original and augmented dataset

Size of dataset	mAP50
Original dataset *(372 train images)*	
Small dataset weights	0.839
Big dataset weights	0.959
Original dataset *(93 test images)*	
Small dataset weights	0.859
Big dataset weights	0.932
Augmented dataset *(1395 test images*	
Small dataset weights	0.660
Big dataset weights	0.874

The mean average precision (mAP) can be seen in Table 1. It can be seen that both networks trained with the original small dataset and the big augmented one performs quite well. However, the network trained with the augmented dataset performs better on every dataset. This has been done in only 50 epochs of training. A longer training time could possibly increase the mAP for both networks. However, due to time constraints, this has not been executed.

1.4 Part conclusion

During the development phase a few different vision algorithm were developed and analysed based on how they perform in segmenting an image and the possibility of taking a method further to actually distinguish the fence from its environment. A lot of time was spend making the Fourier Transformation, seen in subsection 1.2, as well as a screening method for this approach, see subsubsection 1.3.1. It has been shown that this method can be used to find the grid structure in an image and detect a breach in the fence. However, this method proved very unstable with different input images, so each new environment (e.g. different light conditions) has to be very specific fine tuned for it. Therefore this method was not tested and developed further upon, since it is very hard to make it versatile for every possible environment in which a solution might be required.

The problem with the Fourier Transformation method was that when a more bright input image of the fence was analysed, it performed quite poor on segmenting the image, which is a hard requirement for the screening approach. Therefore two other segmentation approaches were developed, a Canny edge detector and U-Net deep-learning segmentation algorithm. The Canny edge detector and convolutional based segmentation, see subsubsection 1.2.3 & subsubsection 1.2.4, had the same issues as the Fourier, making it work well in multiple different environment proved challenging, therefore an idea of using a deep learning to segment the image arose seen in subsubsection 1.2.2. However, to train this network, the method needs to know where the fence is in an image. Which kind of defeats the purpose of this algorithm, because if the fence location was known, then the segmentation of the fence would be complete and could be screened for breaches. Based on this knowledge we discuss that using another deep-learning algorithm could prove a success if chosen correctly. Therefore the final method developed in this project was a Masked Regional Convolutional Neural Network (Masked RCNN), see subsubsection 1.3.2. Where a region of interest is specified when training the network, which is desired, because we generate the training data ourselves with artificial breaches. The location of these artificial breaches is known, thus making it easy to put a region around, which the network will train and learn from.

Furthermore, using such an method, the network can be trained using multiple different environments, since the solution should be applicable in everyone of these. Artificial data was utilised to mimic snow, rain, cloudy day, sunny day and image with different blurs, thus making it very stable and versatile for different environments without having to fine tune it to each new environment.