

# Automating Feature-Extraction for Camera Calibration Through Machine Learning and Computer Vision

---

ELIAS ÅKEBORG

MASTER'S THESIS

DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY

FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY



# Automating Feature-Extraction for Camera Calibration Through Machine Learning and Computer Vision

Author:  
Elias Åkeborg  
e16061ak-s@student.lu.se

Department of Electrical and Information Technology  
Lund University

Company:  
Spiideo

Principal Supervisor:  
Mattias Borg, mattias.borg@eit.lth.se

Supervisors:  
Carl Larsson, carl@spiideo.com  
Malin Velander, malin.velander@spiideo.com  
Karl Åström, karl.astrom@math.lth.se

Examiner:  
Erik Lind erik.lind@eit.lth.se

June 15, 2021



---

# Abstract

---

Machine learning as a field has expanded in an explosive manner, with more companies interested in using the technology. One of these companies, Spiideo, uses Machine learning to automatically stream and record sports, highlighting key events - all automatically without a cameraman. However, these cameras have to undergo a lengthy calibration process involving manual feature extraction. This work investigates the usage of machine learning and computer vision to automate this work. In particular, both U-Net and DeepLab v3+ networks were trained on sets of images and related data from previous feature extractions. From the ML detected features, ridge detection and sub-pixel optimization was used to remove outliers and for classification. The accuracy of the ML and computer vision combination was compared to the manual feature extraction, yielding similar results. The DeepLab v3+ network was found to very accurately extract the intended features, leading to high accuracy independent of camera position, camera angle or noise from the stadium.

Keywords: Machine learning, computer vision, feature extraction, classification, camera calibration, DeepLab, U-Net, ridge detection, sub-pixel optimization.



---

# Popular Science Summary

---

## **Teaching my computer to find the lines on a football field**

The interest in sports is gigantic everywhere in the world, and the demand for watching sports is increasing even more due to the Covid-19 pandemic. It does not matter if it is watching professional teams, friends, your children or grandchildren playing: watching games live is not always easy - mostly professionals can be seen on TV!

Spiideo has one solution to this: streaming sports online without a camera man by instead using “Artificial Intelligence”. This method teaches a computer to learn human-like tasks. In this case, Spiideo trains a computer to act like a camera man: e.g. following the ball during a football game and highlighting when goals are scored. Their cameras do however, require a lengthy calibration to work as intended. Currently, this calibration is done manually and is both quite tedious and time consuming. One of the main steps of this process is to click and save points on the outermost lines on the pitch (from an image) for each camera. One solution to this was explored in this project. In collaboration with Spiideo, I created a method using artificial intelligence, to find, mark and save lines on a football field - removing the need for manual work.

One problem during the project was how football fields and arenas can look very different: some have running tracks close by, some have advertisements in the arena and some even has extra lines on the pitch! One way to work around this problem was removing these "bad" points by a few so-called "computer vision algorithms". These are mathematical models which helped by finding points that seemed out of place!

The project eventually found a model that works: accurately finding and saving the lines on a football pitch after removing any outliers. There were some problems when lines were not clearly visible - in one case the entire pitch was covered in snow! However, in most cases the method was very accurate!



---

## Acknowledgments

---

This master's thesis was done as part of my Master of Science degree in Engineering Nanoscience at the Faculty of Engineering at Lund University. The project was made in collaboration with Spiideo, but executed at home due to the viral outbreak of Covid-19. When doing a project like this one, there are always many people involved in one way or another. Due to the unfortunate circumstances of the Covid-19 virus forcing the project to be done fully from home, much of the support came from long email chains, chats online and through moral support.

First of all, I would like to thank to my principal supervisor Mattias Borg for supporting me through the toughest part of the thesis project when progression was slow and for all the assistance around the practicalities that comes with a project of this size.

Secondly, I would like to express my sincere appreciation to my main supervisor on Spiideo, Malin Velandar for the weekly support, great chats and for forcing me to reflect on my progress.

I would also like to thank Carl Larsson for introducing me to the project and for the very interesting conversations.

Additionally, a large thank you to Karl Åström for the assistance with coding and for the many great discussions regarding different solutions and approaches. Also, a shout-out to Robin Gustavsson for helping me understand the current calibration process.

Finally, to everyone else that supported me for the duration of this master's thesis - thank you all very much. Particularly my family who is always there for me, listening to my rambling and frustration during more challenging periods of the project. Furthermore, thank you to my friends for letting me take my mind off things, pushing me through rough patches and cheering me on. Finally my team of amazing football players and staff, who has done so much to help me with the final years of my degree and this thesis.

*Elias Åkeborg, 2021*





---

# Table of Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Problem Description . . . . .	2
1.3	Aim . . . . .	2
1.4	Previous Work . . . . .	2
1.5	Thesis Outline and Disposition . . . . .	3
<b>2</b>	<b>Background and Theory</b>	<b>5</b>
2.1	Artificial Intelligence and Machine Learning . . . . .	5
2.1.1	Convolutional Neural Network	5
2.1.2	Autoencoder	7
2.1.3	U-Net	9
2.1.4	DeepLab v3+	9
2.2	Computer Vision . . . . .	9
2.2.1	Ridge Detection	11
2.2.2	Sub-Pixel Optimization	11
2.3	Current Camera Calibration . . . . .	11
2.4	Additional Background . . . . .	11
<b>3</b>	<b>Methods</b>	<b>13</b>
3.1	Data and Image Pre-Processing . . . . .	13
3.2	Line Detection Network . . . . .	14
3.2.1	U-Net	14
3.2.2	DeepLab v3+	15
3.3	Ridge Detection and Sub-Pixel Optimization . . . . .	15
3.4	Line Classification . . . . .	16
3.5	Generating Outputs . . . . .	16
3.6	Experimental Setup . . . . .	16
<b>4</b>	<b>Results</b>	<b>19</b>
4.1	Line Detection Through Machine Learning . . . . .	19
4.1.1	U-Net	19
4.1.2	DeepLab v3+	23
4.1.3	Generalization	27

4.2	Ridge Detection and Sub-Pixel Optimization . . . . .	32
4.3	Line Classification and Output Generation . . . . .	32
<b>5</b>	<b>Discussion</b> _____	<b>41</b>
5.1	ML Networks . . . . .	41
5.1.1	Generalization . . . . .	42
5.2	Computer Vision Aid . . . . .	42
5.3	Visual Effects . . . . .	43
5.4	Comparison to Manual Work . . . . .	44
5.5	Future Prospects . . . . .	45
<b>6</b>	<b>Conclusions</b> _____	<b>47</b>
6.1	From Experimental Data . . . . .	47
6.2	Network Analysis . . . . .	47
6.3	Usage of Computer Vision . . . . .	47
6.4	Observations . . . . .	48
6.5	Reflections . . . . .	48
	<b>References</b> _____	<b>49</b>
<b>A</b>	<b>Appendix A: Football Pitch Nomenclature</b> _____	<b>53</b>
<b>B</b>	<b>Appendix B: ML-Training Process</b> _____	<b>55</b>

---

## List of Figures

---

2.1	Figure showing the ReLU function. Note that the function is 0 for negative values and linear for positive values. Image taken from [14].	6
2.2	Figure showing an example of a CNN network. Note the convolutional, ReLU and pooling transformations of the input. Image taken from [9].	7
2.3	The general autoencoder structure. Note that the encoder stage is from input to the bottleneck (called latent vector in the image) and decoder stage goes from the bottleneck to output. Image taken from [14]. . . . .	8
2.4	The autoencoder structure (grey circles represent perceptrons and arrows the direction of data flow). Image taken from [14]. . . . .	8
2.5	The U-Net architecture. Note the connections between each pooling stage to the respective up-convolution. Image taken from [17]. . . .	10
2.6	The DeepLab v3+ model architecture. Note the atrous convolution upscaling filter. Also note the encoder and decoder stages. Image taken from [19] . . . . .	10
3.1	Three different image cutouts showing lines and grass from three different parts of the pitch. Left is from penalty box, center is from goal box, right is from goal line. Note the similarities between the images in regards to position and angle of the lines. . . . .	14
3.2	A (part of) text file from an old calibration process indicating the values saved from the first step of manual feature extraction. Note that only points and image size are saved. Furthermore, note how the amount of dashes prior to an integer value indicates if the value is x or y coordinate (one dash is for y and two for x). When three dashes are present, the x value is the start of a line, whereas four dashes indicates the first of the four saved lines. . . . .	17
3.3	An image plotting the points from one of the Spiideo data files. The white lines are between the points. Note how the points create the bounding lines of a pitch (two side lines, the goal line and the halfway line). . . . .	17

4.1	Three sets of image pairs with inputted image on the left and outputted line detection (white points) on the right. Note that all these images are of the right half of the pitch. . . . .	21
4.2	Three sets of image pairs with inputted image on the left and outputted line detection (white points) on the right. Note that all these images are of the left half of the pitch. . . . .	22
4.3	Three sets of image pairs with inputted image on the left and outputted line detection (white points) on the right. Note that all these images are of the right half of the pitch. . . . .	24
4.4	Three sets of image pairs with inputted image on the left and outputted line detection (white points) on the right. Note that all these images are of the left half of the pitch. . . . .	25
4.5	Four images depicting the DeepLab v3+ network response on one of the input cut-outs. Top-left displays the input cut-out. Bottom-left displays the label for the cut-out. Bottom-right shows the pure network response and top-right overlays the network response (red) on top of the input image (with a blue tint to clarify the network response further.) . . . . .	26
4.6	Six images displaying the points (white) from the U-Net (2nd row) and DeepLab v3+ (3rd row) network detections from input images (1st row) with different camera positions. Note that the U-Net does not find any lines on the pitch when the camera is positioned behind the goal while the DeepLab v3+ seems have similar accuracy independent on if the camera is positioned behind the goal or from the side. . . . .	28
4.7	Six images displaying the points (white) from the U-Net (2nd row) and DeepLab v3+ (3rd row) network detections from two different input images (1st row). Note that the DeepLab v3+ network has very few detections for both images while the U-Net network has more detections from both the pitch and from the background. . . . .	29
4.8	Three images displaying the input and detected lines for the U-Net and DeepLab v3+ network when there is snow on the pitch. Note the difference between finding many detections including lines and noise for the U-Net network to finding few detections from either lines or noise in the DeepLab v3+ network. . . . .	30
4.9	Images displaying ML network responses to the input image having additional lines on the pitch itself . . . . .	31
4.10	Two images showing the input(top) and ridge response (bottom) when running the input through all the elongated Gaussian filters. Note that the bottom image displays local maximums. . . . .	33
4.11	Two zoom-levels of the same image. The upper image shows a football field with blue markings from the sub-pixel optimizations plotted. The lower image is a zoomed in version on one of the lines in the upper image. Note the red point (which is the detection sub-pixel) and blue arrow (indicating the normal of the detection point). . . . .	34
4.12	Two images displaying remaining points after sub-pixel optimization (top) and outlier removal (bottom). Note the differences between the images. . . . .	35

4.13	Upper: points saved for each line (different colors for each line). Bottom: Polynomial fitting, 2nd deg, of the points in the upper image. .	36
4.14	Two images showing both the saved points from the project (top) and the manually saved points from the old calibration (bottom). Note the similarities here. . . . .	37
4.15	Three images showing the input image (top), the automatically generated output (middle) and manual calibration (bottom). Note the similarity of the two lower images. . . . .	38
4.16	Two text files both displaying image size and saved points for two different pitches. Left: generated file. Right: manually created file. Note the similarities. Also note that integers (for the points) start with one dash if its the y-value and two dashes if its the x-value. Additionally, three dashes indicate a new line and four lines are used for the first of all the lines . . . . .	39
A.1	Drawn schematic of a football field with used nomenclature indicated for sections and lines. . . . .	53
B.1	Picture from the training process of one the DeepLab v3+ networks, the blue line indicates the accuracy and orange loss. Black points are for validations. . . . .	56
B.2	Picture from the training process of one the U-Net networks, the blue line indicates the accuracy and orange loss. Black points are for validations. . . . .	57



---

## List of Tables

---

4.1	Table showing the changes of some key variables, amount of input images, training time and validation accuracy for the progression of every other U-Net network iteration. Note that LR is short for learning rate. For the amount of images used for training: note that the first number indicates the amount of cut-outs used in training, and the number within the parenthesis indicates the amount of different arenas these are taken from (e.g. 600 cut-outs from 6 different arenas = 100 cut-outs per arena) . . . . .	20
4.2	Table showing the validation accuracy and training time for the two different DeepLab v3+ model types. . . . .	23





---

## Abbreviations

---

ML - Machine Learning.

CNN - Convolutional neural network.

AI - Artificial Intelligence.

LR - Learning Rate.



# Introduction

---

*“Begin at the beginning,” the King said, very gravely, “and go on till you come to the end: then stop.”*

---

*Lewis Carroll,  
Alice in Wonderland*

## 1.1 Motivation

With the ever-growing field of machine learning becoming more apparent every year, more companies see the possibilities of implementing and using the technology. The advances in computing power have further advanced the practicability to use ML networks on large data sets and for more applications. Due to the Covid-19 pandemic, sports clubs are looking for new opportunities to work around the economic losses [1] and engage with their following [2] while supporters seek ways to follow their teams from home [3]. This applies from a professional level all the way down to watching your children or grandchildren play. Spiideo facilitates this process by streaming sports online through their services (both professional teams, amateur and even youth). The Spiideo cameras do however, require a thorough and manually executed calibration process. This thesis describes the development of an automatized model with the intent to simplify or even fully remove manual labour during this camera calibration process. The project was conducted in collaboration with Spiideo and executed from home due to the Covid-19 virus.

On a broad level, the Spiideo system is built around three main components; 4k video cameras, a cloud-based server and machine learning (ML) algorithms. Currently, ML is mostly used to find the ball, players, recognizing when a goal is scored or other events during a game. For teams, having recorded material has shown to be a major contribution to successful growth on both an individual and team level [4]. Furthermore, since the cameras record the entire pitch and uses ML to follow the play, there is no need for a cameraman which in turn makes the system easier to use.

## 1.2 Problem Description

Many of the AI-analysis features used in Spiideo's sport analysis require that both the exact position and orientation of the pitch in relation to the cameras are known. Thus, every new installation goes through a lengthy calibration process in which the positions of the cameras relative to the field are calculated based on known objects in the image (the bounding lines and the intersections between the halfway line and the central circle). Currently this calibration is done manually for each camera which requires both time and effort. The calibration process starts by manually extracting the features (selecting points on every line) and later changing the camera parameters in iterations until a acceptable result is reached. Changes in camera angle or position would require an entirely new calibration, starting with feature extraction by manually selecting new points. This thesis tries to automatize the feature extraction during the initial phases of the calibration process through a combination of ML and computer vision algorithms.

## 1.3 Aim

The primary objective of this thesis project is fully automating feature extraction for the camera calibration process of Spiideo cameras. However, in doing so, gaining understanding about ML and computer vision will be paramount for future improvements. Furthermore, this thesis project investigates the possibilities and issues when attempting to generalize the feature extraction process for different camera angles, camera positions or drastically different pitches in relation to weather, visibility, background noise etc.

## 1.4 Previous Work

For this project, previous work in each area was important. For the feature extraction, much previous work has been done through both ML and computer vision algorithms.

*Rethinking Atrous Convolution for Semantic Image Segmentation* by Chen et. al. [5] and *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs* by Chen et. al. [6] both explain the DeepLab networks. The two individual papers successfully prove how the DeepLab networks can accurately be used for semantic image segmentation. Additionally, the first DeepLab system was state-of-the art in 2017, with the three new versions performing even better. The papers were highly influential regarding choosing a ML model for the feature extraction during this thesis.

*Robust camera calibration for sport videos using court models* by Farin et. al. [7] discusses a robust calibration of camera parameters for different types of sport courts. Furthermore, the study discusses some ways to extract lines and a way to use combinatorial optimization from detected lines to find a sports court. The approach used in the study inspired some key parts of both line extraction, opti-

mization and classification for this thesis.

*End-to-End Camera Calibration for Broadcast Videos* by Sha et. al. [8] proposes another feature extraction approach in sports for camera calibration, in more challenging scenarios, using a different Semantic Segmentation technique combined with homography to extract the features in the image.

## 1.5 Thesis Outline and Disposition

This thesis mostly describes and discusses the ML networks used, how computer vision can be used to improve the feature-extraction accuracy and finally discusses some future modifications or additions to the network.

The first step of the thesis project was to collect and label the data. The 4k-resolution images were captured from old calibrations from different stadiums. These images then required some pre-processing to run in a ML network.

The second step was to implement and train a ML network to find the lines on the pitch. Note that only lines on the pitch are sought after, whereas detections from the stadium or arena, background or from advertisements in proximity to the pitch could interfere and lower the accuracy of the network.

The third step was using computer vision algorithms to remove outliers from the ML detections by using ridge detection and sub-pixel optimization. When observing the currently manual calibrations, not many points from each line seem to be required for the calibration to succeed. However, the points need to be on (or very close to) the line and therefore cannot have any significant outliers.

The final step was to use the detected points and classify them depending on which line they were a part of and save these classified points into data files to later be used to choose the camera parameters (files with the same appearance as the ones from manual feature extraction).

The approach used in this thesis project was using certain neural networks based on the convolutional neural network (CNN): namely U-Net and DeepLab v3+. These networks were chosen due to their capabilities to save information between convolutions (U-Net) or accuracy in semantic segmentation (DeepLab v3+), both helping decipher the intended features during this project [11].

**Disposition:**

**Chapter 2** introduces the reader to some of the core theory and concepts for the ML networks and the different computer vision algorithms used. Additionally, a short explanation of Spiideo's current camera calibration process and feature extraction is described.

**Chapter 3** explains the methods used in each step to tackle the problem at hand and gives an overview of the experimental setup for the thesis project.

**Chapter 4** displays the results for different iterations of the ML networks, the improvements made by using computer vision algorithms, and two examples of the classification and output generation.

**Chapter 5** discusses the difference between the ML network results, the addition of computer vision algorithms to remove outliers and the point classification for generation of output files. Furthermore, the chapter discusses future modifications, additions and changes to the approach.

**Chapter 6** concludes the thesis project with reflections from the entire work process.

**Appendix A:** gives an overview of a football field to explain the used nomenclature.

**Appendix B:** displays two large images from the ML training and validation process.

---

# Background and Theory

---

*“Never trust anything that can think for itself if you can’t see where it keeps its brain”*

---

*J.K. Rowling,  
Harry Potter and the  
Chamber of Secrets*

## 2.1 Artificial Intelligence and Machine Learning

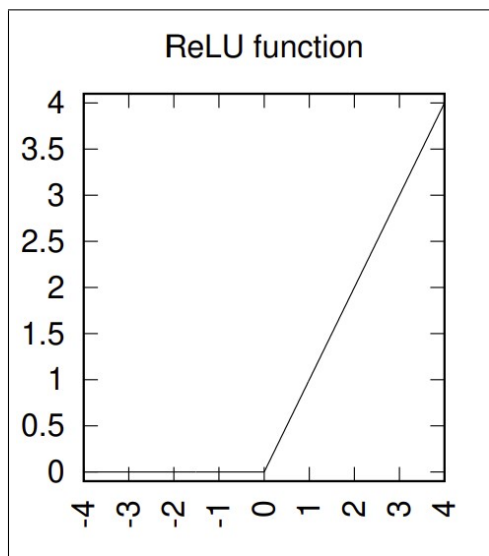
For this thesis project, the reader is assumed to have some prior basic knowledge about AI and neural networks. Therefore, only the core of the Machine learning (ML) networks used for the project will be introduced in this report. However, the report will be written in a way, where even readers without much prior understanding can follow the results and discussion. In this section, the two main neural networks (U-Net and DeepLab v3+) are explained after a brief introduction to the Convolutional Neural Network and Autoencoder on which the two networks are built upon.

### 2.1.1 Convolutional Neural Network

**The Convolutional Neural Network (CNN)**, is one of the main network architectures used in ML. The CNN network typically consists of an input layer, an amount of hidden layers and an output layer. However, what separates the CNN from other ML techniques is the usage of **convolution layers**. These layers act as filters, finding certain specific features from the input (e.g. an image). By using a larger amount of hidden layers and filters per layer, many different features can be found and for different resolutions in the input. The filters can feature e.g. brightness, color or edges, but also more complex features like an eye, a tire or a cat’s tail [9]. Furthermore, the network architecture typically uses a so called **"Rectified linear unit" (ReLU)** and a **pooling layer** between each convolution. ReLU is a non-linear function that is typically used to simplify the values in a network by removing negative values (set to zero) while keeping positives. See Figure 2.1 for



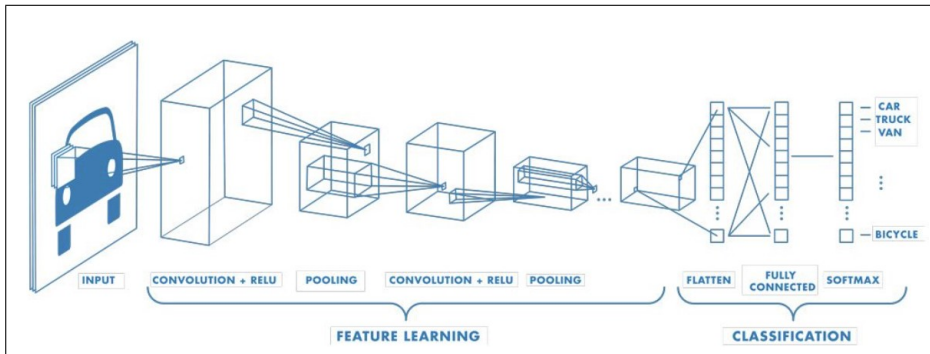
the appearance of the ReLU function. The ReLU function is typically referred to as an activation function, due to its nature of "activating" only positive features, saving these to the next layer.



**Figure 2.1:** Figure showing the ReLU function. Note that the function is 0 for negative values and linear for positive values. Image taken from [14].

**Pooling** is a function used to reduce the data complexity by combining data from the output of a convolutional layer. Thus reducing the amount of parameters for the next hidden layer stage. When using a 2D matrix input (e.g. an image), pooling can use either the maximum from the outputs (max-pooling) or the average of outputs from a rectangular neighborhood of perceptrons. This forces the outputs of the next stage to become less sensitive to small changes and typically also down-samples the network to smaller dimensions (e.g. 256x256 to 64x64). For a full example of a CNN network, see Figure 2.2 and note the input stage, the hidden layers (convolution, ReLU and pooling) and output stage(s) [9], [10].

The training of a CNN typically starts by pre-processing input data to fit the network, extracting features on which the network is supposed to find and finally training on a large data set. Due to the nature of the CNN, the architecture is very suitable for visual object detection, segmentation or classification [9], [12]. For different CNNs, different **parameters** will have different optimal values. The two main parameters typically noted are **Learning Rate (LR)** and **Batch Size**. LR controls the step size during training of a network and is typically found from a trial-and-error approach. A smaller step size requires more updates before the network reaches its ideal point whereas a large step size goes quicker with the added risk of drastic updates leading to divergent or periodical behaviours. A



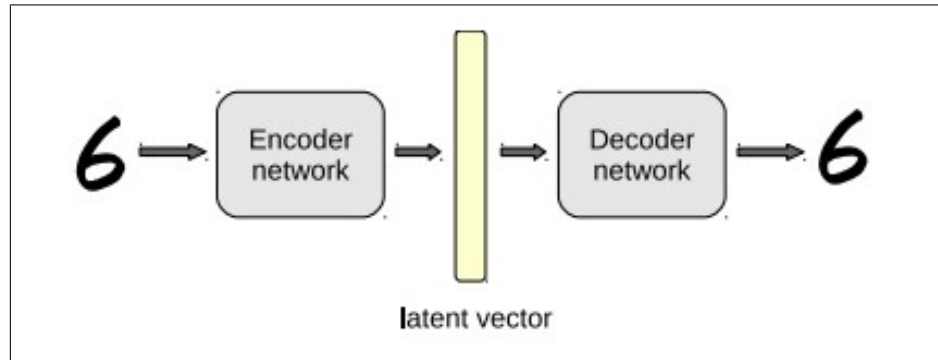
**Figure 2.2:** Figure showing an example of a CNN network. Note the convolutional, ReLU and pooling transformations of the input. Image taken from [9].

smaller step size also has the additional risk of the network getting stuck in a local minimum (for loss) and therefore not finding the global minimum. Depending on the CNN model the LR will differ, thus requiring the trial-and-error approach to find the optimal value (LR value typically around 0.1-0.0001). **The batch size** of a network describes the division of a data set into smaller "batches". The network is then trained on one batch at a time in iterations. The batch size is typically correlated to the optimal learning rate, but there is no simple way to optimally choose this parameter. Depending on the batch size, the network will be trained on a different amount of iterations. This parameter impacts the network in both a regularizational and computational way. A smaller batch size requires more iterations and more computational power, whilst having additional regularization effects to prevent *overfitting*, where the data performs well on the training data but worse for any new data. Smaller batches therefore mean that the gradient of the cost function will be cheaper to calculate, but with the consequence of being less accurate. [11], [13], [14].

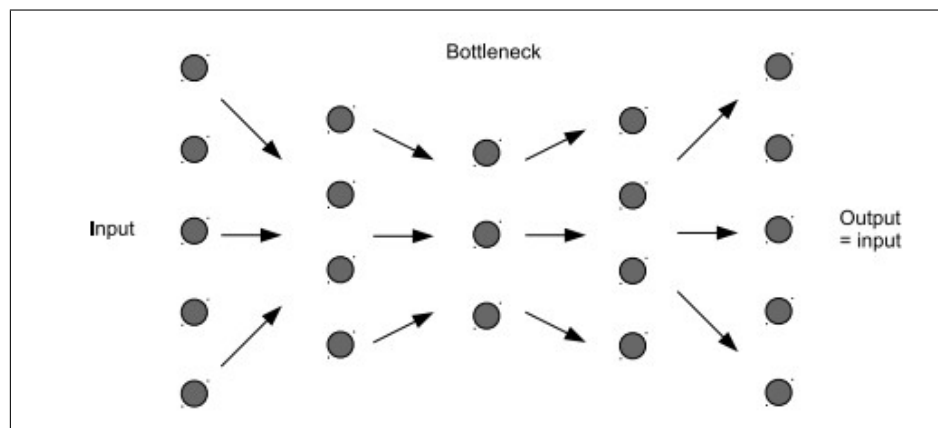
### 2.1.2 Autoencoder

The **autoencoder** is specific type of neural network that attempts to reproduce the input after a reduction and reconstruction process. See Figure 2.3 which displays a simplified image of an autoencoder architecture and Figure 2.4 which displays an example of an autoencoder with 3 hidden layers.

Note that typically an autoencoder has more than one hidden layer. The middle hidden layer (often called "code" or "bottleneck") always has fewer nodes (perceptrons) than the input layer, whereas the output has the same amount of nodes as the input (remember that the main idea is to replicate the input). The first half of the network "encodes" the input to the bottleneck layer and the second half "decodes" the information from the bottleneck layer to the output. Also note that generally, autoencoders follow a symmetrical pattern in terms of the hidden nodes in the encoder and decoder - sometimes called a "butterfly design". Similar



**Figure 2.3:** The general autoencoder structure. Note that the encoder stage is from input to the bottleneck (called latent vector in the image) and decoder stage goes from the bottleneck to output. Image taken from [14].



**Figure 2.4:** The autoencoder structure (grey circles represent perceptrons and arrows the direction of data flow). Image taken from [14].

to most ML networks, more complex data requires more complex hidden layers [11], [15].

Using the autoencoder structure in combination with the CNN architecture, the "Convolutional Autoencoder" is created. This architecture encodes the input through convolutional steps, similar to the normal CNN, then decodes using "deconvolution", which acts somewhat like the opposite of convolution, trying to recreate the input [16].

### 2.1.3 U-Net

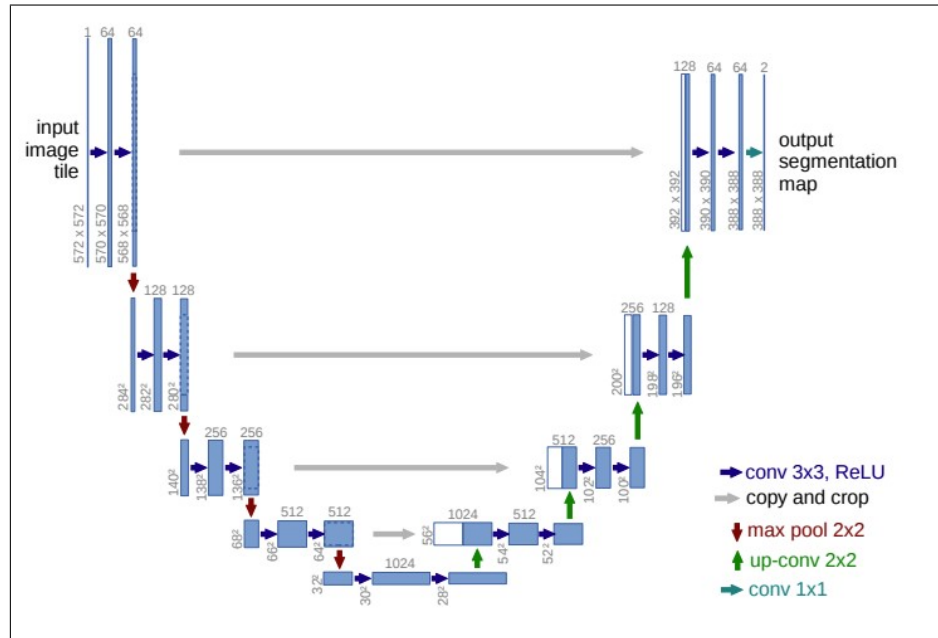
The U-Net network architecture, created to work with smaller training set, is a type of CNN. The architecture takes an input image and (using convolutional and pooling layers with ReLU as an activation function) runs both sub-sampling and up-sampling in stages, with information being retained between down/up-sampling pairs (skip connections), see Figure 2.5. The architecture looks quite similar to the autoencoder, see Figure 2.4, but there are a few important differences. Mainly, note that for typical autoencoders, there is a straight front-to-back direction from input to output by first encoding then decoding the input. The U-Net instead retains connections between each stage. In Figure 2.5, note these connections marked as grey arrows. One consequence of these connections is that one cannot use only half of the architecture (compare to only using the encoder or decoder stage of an autoencoder) [17], [18].

### 2.1.4 DeepLab v3+

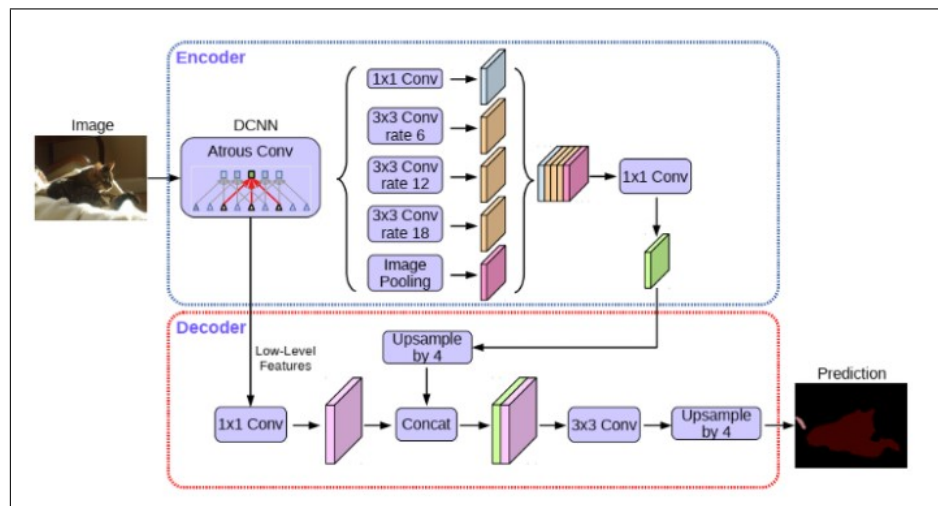
DeepLab is another architecture based on CNN. Instead of the typical convolution layering, the architecture uses a method called "atrous convolution", which is a type of convolution using upsampled filters. By combining the atrous convolution with "bilinear interpolation" (a re-sampling technique in three dimensional images to filter or map points), a full resolution feature map is recovered. The architecture has been updated a few times, adding on improvements over the preceding version. Version 3+ is the latest iterations of the architecture, using an added "decoding" module to refine results further, making the architecture somewhat similar to the autoencoder approach (having an encoder and a decoder stage). The image in Figure 2.6 displays a simplified version of the DeepLab v3+ architecture and the atrous convolution filter [5], [6], [19].

## 2.2 Computer Vision

The following section very briefly introduces the reader to the core concepts of the two Computer Vision techniques used in the project: ridge detection and sub-pixel optimization.



**Figure 2.5:** The U-Net architecture. Note the connections between each pooling stage to the respective up-convolution. Image taken from [17].



**Figure 2.6:** The DeepLab v3+ model architecture. Note the atrous convolution upscaling filter. Also note the encoder and decoder stages. Image taken from [19]

### 2.2.1 Ridge Detection

The first Computer vision technique used in this thesis project is based on ridge detection. This method tries to automatically detect one dimensional features in feature detection from an image. In mathematics, a ridge is defined as the point(s) where a line or curvature has a local maximum. One of the ways to find ridges are by running a grey scale image through Gaussian filters and then finding and saving the local maximums [20], [21].

### 2.2.2 Sub-Pixel Optimization

Sub-pixel optimization is a mathematical model used to increase the resolution of points in an image. The method is based on finding both the sub-pixel point and its normal for chosen pixels. By comparing the normal and sub-pixel point-coordinates of a set of points on a ridge response image, one could determine if the points are from the same one ridge [22] - [26].

## 2.3 Current Camera Calibration

The current calibration of Spiideo cameras is done through manual steps. First, an image from each camera is saved. Sequentially, for each image, feature extraction of points for the bounding lines of the pitch (or using halfway line if camera only sees half of the pitch) are marked manually and saved into a data file. Combining these points, with knowledge about the size of the pitch, a recursive trial-and-error approach is done to optimize the different camera parameters from an approximated starting point. Specifically, removal of distortion is optimized by straightening each line (from the saved points). Once the parameters seem sufficient enough, the camera parameters are saved and the calibration is finished.

## 2.4 Additional Background

The reader is assumed to know the core nomenclature of a football field, but Appendix A does give an overview of the main nomenclature used to describe the different areas and lines on a football field in this thesis.



*“Don’t always complain the way  
isn’t there. If you can’t find the  
way, create it.”*

---

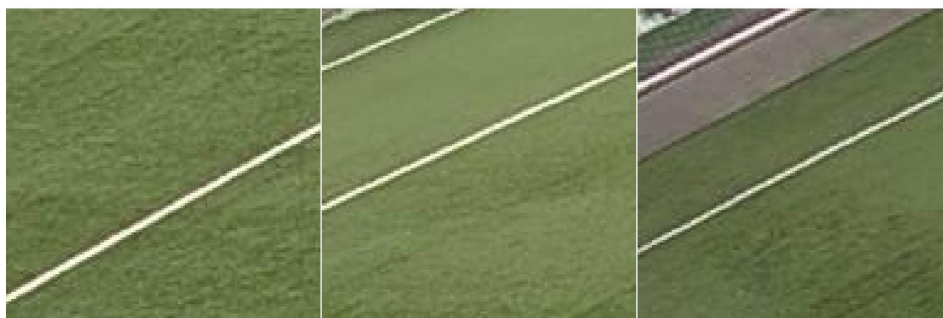
*Israelmore Ayivor*

### 3.1 Data and Image Pre-Processing

Prior to training the ML network, some pre-processing of the available data was done. The first ML network, described in the next section, used images as input for the training. These images were extracted from old calibrations of the Spideo cameras. The native resolution of these images was in 4k, but to reduce the complexity and standardize images for training - each image was cut into smaller images of equal size (128x128, 256x256 or 512x512 px).

Additionally, for each calibration image, text files with lists of the manually extracted features (points for each line on the pitch) were used to create labeled images for the ML training. This labeling was made by making a grid of zeros with the same size as the native image, then placing ones in the positions of (and a set radius around) the lines created by connecting the points on each field-line. Note that the halfway line is used as a bounding line during this thesis project due to the cameras typically only seeing about half of the pitch. Furthermore, points for the other lines in the image were manually extracted and combined with the data file. The main reason behind this was forcing the network to find all the lines on the pitch, and the data provided excluded lines from the penalty box, goal box and central circle. Note that the two images of size 128x128 can be close to identical even if one image is from the penalty box and one is from another line in the same direction. This can be observed in Figure 3.1 where the left image is from the penalty box, the center image is from the goal box and the right is from the goal line. If this manual step was not made, the network could be trained on close-to identical inputs, with very different labels.





**Figure 3.1:** Three different image cutouts showing lines and grass from three different parts of the pitch. Left is from penalty box, center is from goal box, right is from goal line. Note the similarities between the images in regards to position and angle of the lines.

## 3.2 Line Detection Network

The first step of automatizing the calibration process was removing the manual labour of feature extraction for every camera. Therefore, an initial ML network was created with the intent of finding and separating the lines on the pitch (the features) from everything else in the image. Following the pre-processing in the preceding section, directories for training images, validation images, training labels and validation images were created.

### 3.2.1 U-Net

The initial networks were based on the U-Net architecture. Initially, one image from one stadium was cut into 50 different 128x128px images. These cut-outs were then used to both train, validate and test the network. All the cut-outs used in this initial network included lines from the pitch, forcing the network to recognize the white lines from the grass. From this first network, some of the training parameters were experimented with to create a working environment with correctly made labeling.

Sequentially, the U-Net network was expanded with more calibration images from different stadiums or fields. The network was trained for a larger amount of epochs and evaluated for different training parameters. Furthermore, cut-outs from the parts of the calibration images with no appearing lines (e.g. the sky and the stadium or stand) were introduced into the network to reduce the outputted false positives (noise) from the stadium, the grass and other noise which the network predicted as a line.

The parameters for each iteration of the U-Net network were noted down. To test and compare the results, a new full 4k image, from a different stadium than

the ones used for training and validation was tested on. This test read the 4k image, cut it into the same size used in the network training, ran each image through the network and saved the predicted output. By combining the saved image predictions, a fully sized 4k-resolution image was reproduced. By comparing this predicted image between each iteration, the progression between U-Net iterations was recorded. The final iterations used larger image cut-outs (256x256px).

For the final iterations of the U-Net networks, a more concrete and rigorous testing was done by using a set of 10 different 4k-resolution images and comparing the resulting prediction to the label. This comparison resulted in a more reliable prediction accuracy for the networks.

### 3.2.2 DeepLab v3+

Following the U-Net network experiments, new ML networks with the DeepLab v3+ architecture were created following a similar approach. Due to the directories of image cut-outs already existing from the previous steps, training of the first DeepLab v3+ networks could start almost instantly by just changing the network type in the code.

The initial DeepLab v3+ networks were trained on the same set of images as the final U-Net networks and evaluated with the same testing set of images. Afterwards, the training parameters were adjusted and more images were introduced due to the faster run time for the DeepLab v3+ network. Subsequently, new DeepLab v3+ networks were trained for even larger cut-outs (512x512px). Finally, a few 5-hour training and validation sessions of a 512x512px DeepLab v3+ network were performed and evaluated.

## 3.3 Ridge Detection and Sub-Pixel Optimization

From the ML predicted line detection, computer vision algorithms were introduced to further increase the accuracy of the predicted image.

First, all elongated Gaussian filters were run on a native 4k image from an old calibration (this image was not used for training or validation for any of the ML networks). After running the image through these filters, the detected ridges from local maximums were extracted and observed. The same input image was then run through the ML network. Afterwards, the detections from the ML network were combined with the ridge detection to form a matrix, saving points the ML network predicted with a high chance being on a line and simultaneously being part of the set of detected ridges.

These chosen and filtered detections were then run through a sub-pixel algorithm, finding more accurately exactly the center of each line segment and each detection's normal. Following this, some outliers were removed by only saving sets of points with both a similar normal and position in the image.

### 3.4 Line Classification

Following the line-prediction and computer vision aid, a classification of the different lines was quickly demonstrated. This classification tried to break down the detection into vectors for each of the different bounding lines of the pitch, similar to the manual data from old calibrations. Due to lack of time, a ML network was not finalized. Instead, the resulting image from sub-pixel optimization was broken down into quadrants and more outliers were removed from the detections by comparing the numerical values of each point in the quadrants (for their normal and coordinates). For each quadrant, the remaining points were saved into a vector. Note how this created four vectors: one for each bounding line in the image (in the ideal case). To find the intersections between the lines, all vectors then underwent polynomial fitting. Afterwards, new points were automatically and evenly placed along the four polynomials using the intersections with the other polynomials as starting and end points. These new points were finally saved into four new vectors: one for each line.

### 3.5 Generating Outputs

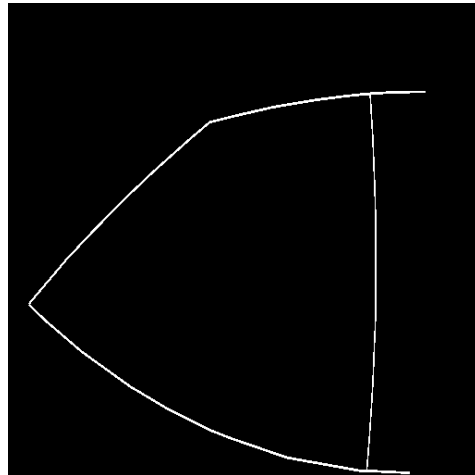
During creation of the labeling sets, the Spiideo data files were observed to follow a specific pattern in respect to order of parameters. Figure 3.2 shows a short version of such a file. Note that after the initial information about image size, the rest of the file consists of points. Furthermore, the amount of dashes (-) indicates if the value is the x or y coordinate. Values starting with 3 or 4 dashes indicate the start of a new line. Figure 3.3 displays the realization of plotting the points from such a data file. By using the vectors from the line classification step, data files were generated to identically replicate the Spiideo data files. Hence, an input image to the system outputs a new file containing evenly spaced points along each of the four bounding lines of the pitch. Remember that the halfway line is used as a bounding line during this project due to the cameras typically only seeing about half of the pitch.

### 3.6 Experimental Setup

Due to the Covid-19 virus, the entire thesis project was executed from home, using an Nvidia Geforce GTX 1050 Ti GPU. The code was written and executed in MatLab.

```
|---  
image_size:  
  width: 3840  
  height: 2160  
points:  
- - - - 26  
  - 254  
  - 136  
  - 195  
  - 383  
  - 73  
  - 298  
  - 112  
- - - 3697  
  - 1326  
  - 3481  
  - 1163  
  - 2565  
  - 570
```

**Figure 3.2:** A (part of) text file from an old calibration process indicating the values saved from the first step of manual feature extraction. Note that only points and image size are saved. Furthermore, note how the amount of dashes prior to an integer value indicates if the value is an x or y coordinate (one dash is for y and two for x). When three dashes are present, the x value is the start of a line, whereas four dashes indicates the first of the four saved lines.



**Figure 3.3:** An image plotting the points from one of the Spiideo data files. The white lines are between the points. Note how the points create the bounding lines of a pitch (two side lines, the goal line and the halfway line).



*“However beautiful the strategy,  
you should occasionally look at  
the results.”*

---

*Winston Churchill*

## 4.1 Line Detection Through Machine Learning

The following section displays the resulting images and the network accuracy during training from different iterations of the ML networks. Additionally, the test accuracy is displayed for the final iterations. Appendix A displays two images saved from the training and validation process.

### 4.1.1 U-Net

#### Model, Validation and Testing accuracy

From Table 4.1, every other iteration of the U-Net network training is displayed. Note the amount of images used for training, the training time and validation accuracy. Additionally, observe the slight changes in parameters between the iterations. Finally, U-Net 7 is the final iteration using 128x128px sized input images and U-Net 9 is the final iteration using 256x256px input images. The validation accuracy for U-Net 7 is 94.88% and for U-Net 9 it is 96.24%.

The total test accuracy for the final U-Net (U-Net 7 in Table 4.1) using 128x128 sized input image cut-outs on the test set was recorded at 91.85% .

The total test accuracy for the final U-Net (U-Net 9 in Table 4.1) using 256x256 sized input image cut-outs on the test set was recorded at 94.67%

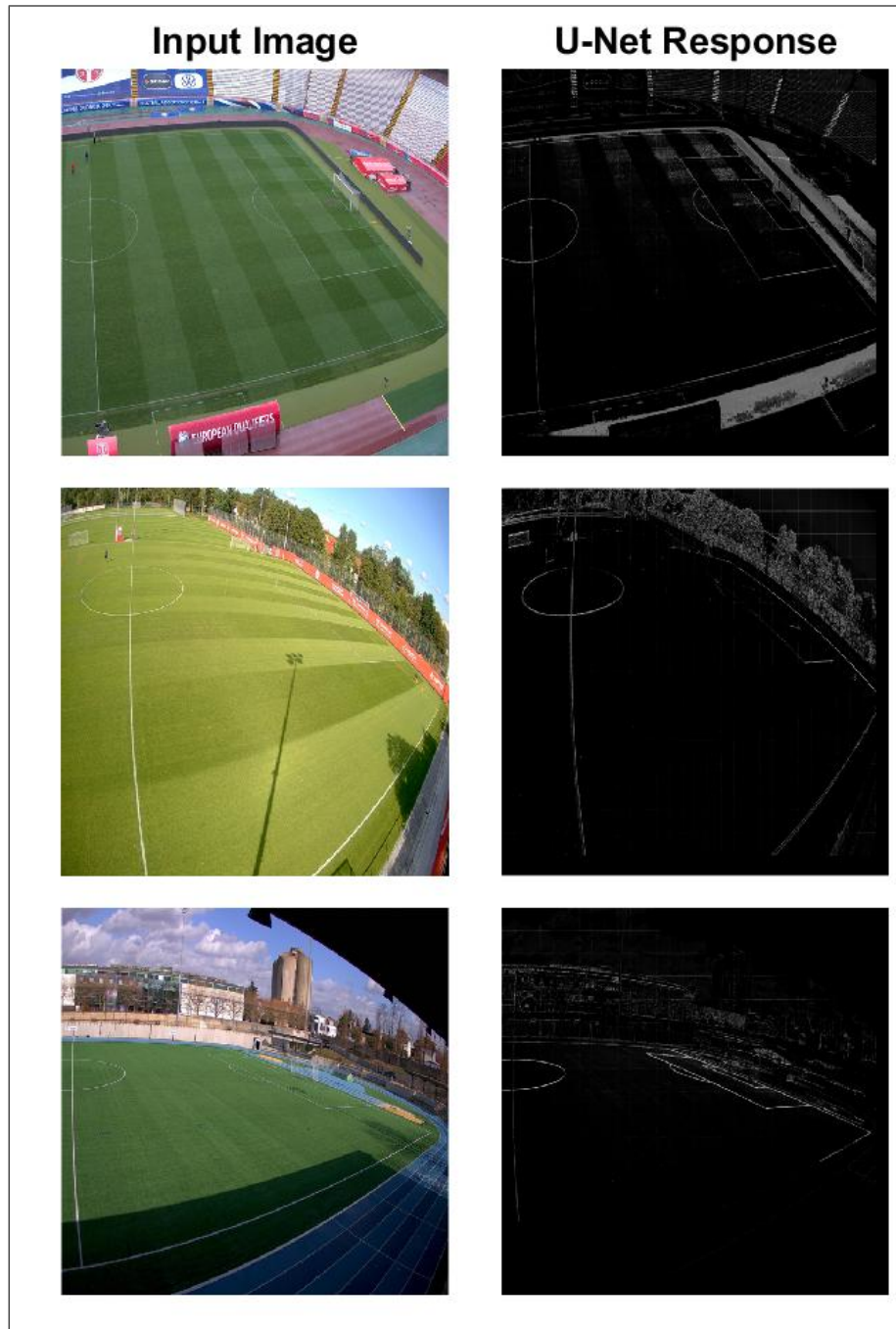
Note that these test results are different to the validation accuracy in both cases.

#### Detections

Figure 4.1 and 4.2 display the network response to strictly using input images where the camera is positioned from the side (which is the same camera position-

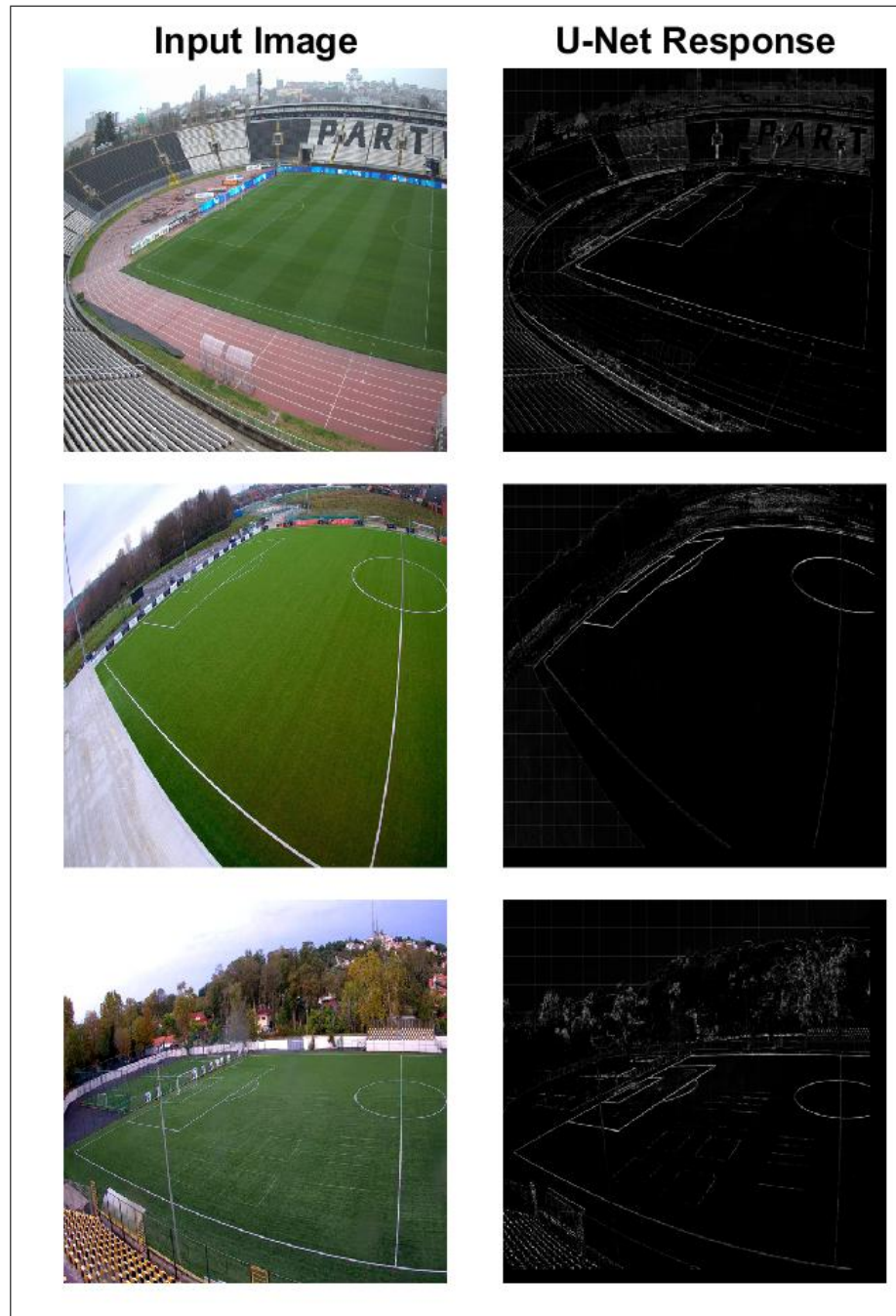
<u>Iteration</u>	<u>Parameters</u>	<b>Number of training imgs (no. of arenas)</b>	<b>Time (min)</b>	<b>Validation accuracy</b>
U-Net 1	Img 128x128px LR: 2e-3 batch size: 128	Training: 40(1) Validation: 4 (1)	25	78.42%
U-Net 3	Img 128x128px LR: 2e-3 batch size: 16	Training: 150(1) Validation: 40(1)	48	86.88%
U-Net 5	Img 128x128px LR: 1e-3 batch size: 16	Training: 125(3) Validation: 40(2)	73	91.28%
U-Net 7	Img 128x128px LR: 1e-3 batch size: 16	Training: 600(6) Validation: 120(2)	168	94.88%
U-Net 9	Img 256x256px LR: 1e-3 batch size: 32	Training: 600(6) Validation: 120(2)	346	96.24%

**Table 4.1:** Table showing the changes of some key variables, amount of input images, training time and validation accuracy for the progression of every other U-Net network iteration. Note that LR is short for learning rate. For the amount of images used for training: note that the first number indicates the amount of cut-outs used in training, and the number within the parenthesis indicates the amount of different arenas these are taken from (e.g. 600 cut-outs from 6 different arenas = 100 cut-outs per arena)



**Figure 4.1:** Three sets of image pairs with inputted image on the left and outputted line detection (white points) on the right. Note that all these images are of the right half of the pitch.





**Figure 4.2:** Three sets of image pairs with inputted image on the left and outputted line detection (white points) on the right. Note that all these images are of the left half of the pitch.

Network	Network type	Training time (min)	Validation accuracy
DeepLab v3+, 256x256 px	resnet-18	140	98.04%
DeepLab v3+, 256x256 px	resnet-50	203	98.22%
DeepLab v3+, 512x512 px	resnet-50	425	98.34%

**Table 4.2:** Table showing the validation accuracy and training time for the two different DeepLab v3+ model types.

ing that the network was trained with).

Note in the two figures how most of the detections from the pitch itself are from the lines. Also note the amount of detections from noise (e.g. the lines detected from the stadium in the top right image in Figure 4.2.)

#### 4.1.2 DeepLab v3+

##### Model, Validation and Testing accuracy

From Table 4.2, The two final DeepLab v3+ networks are displayed. These networks were trained on the same amount of images as the final U-Net networks, while later iterations instead used 512x512px sized cut-outs. Note the difference in time and accuracy between the models and to the U-Net network.

The total test accuracy when using the resnet-18 network on the test set was recorded at 98.04 % .

The total test accuracy when using the resnet-50 network on the test set was recorded at 98.34 % .

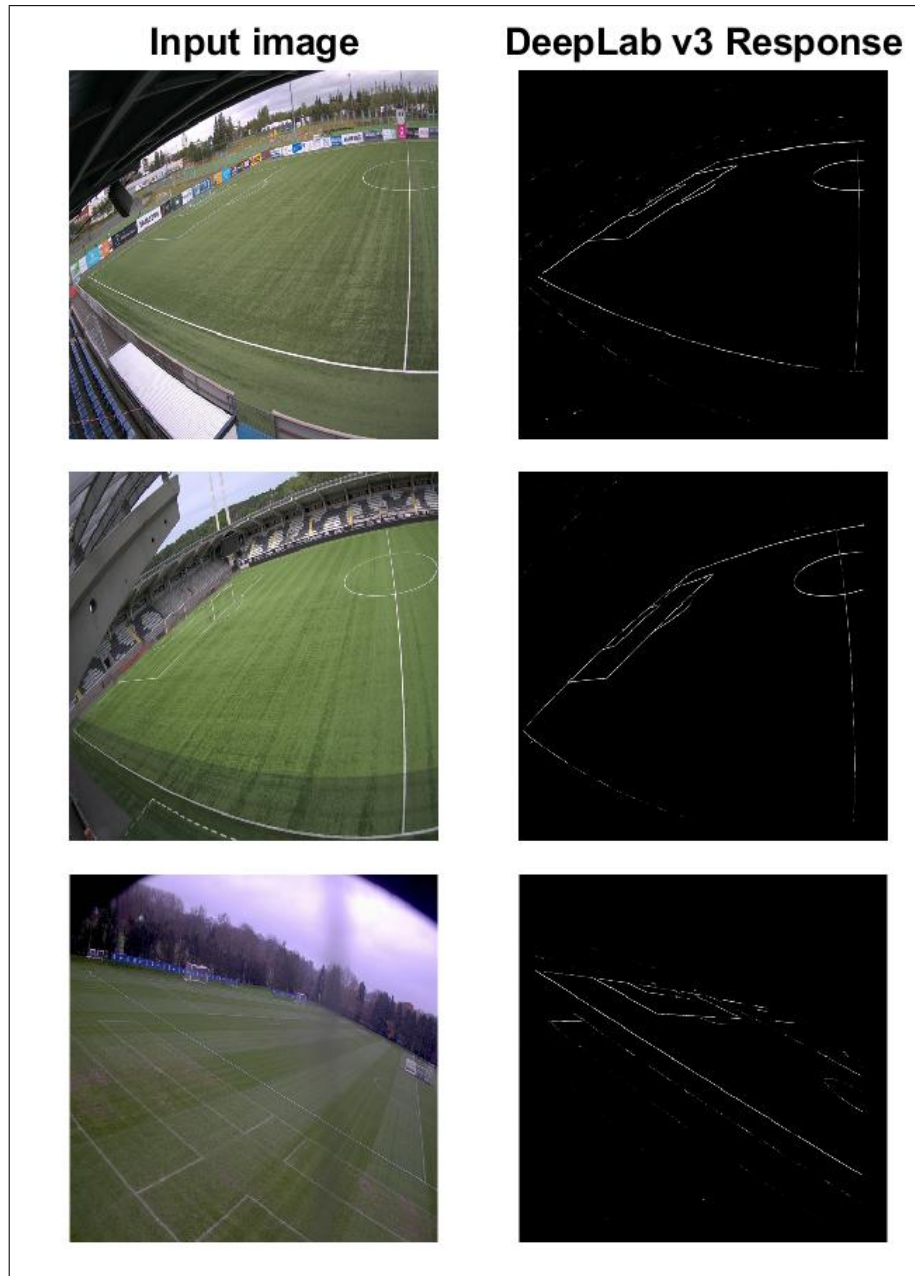
The test set used for these networks was the same as for the U-Net networks.

##### Detections

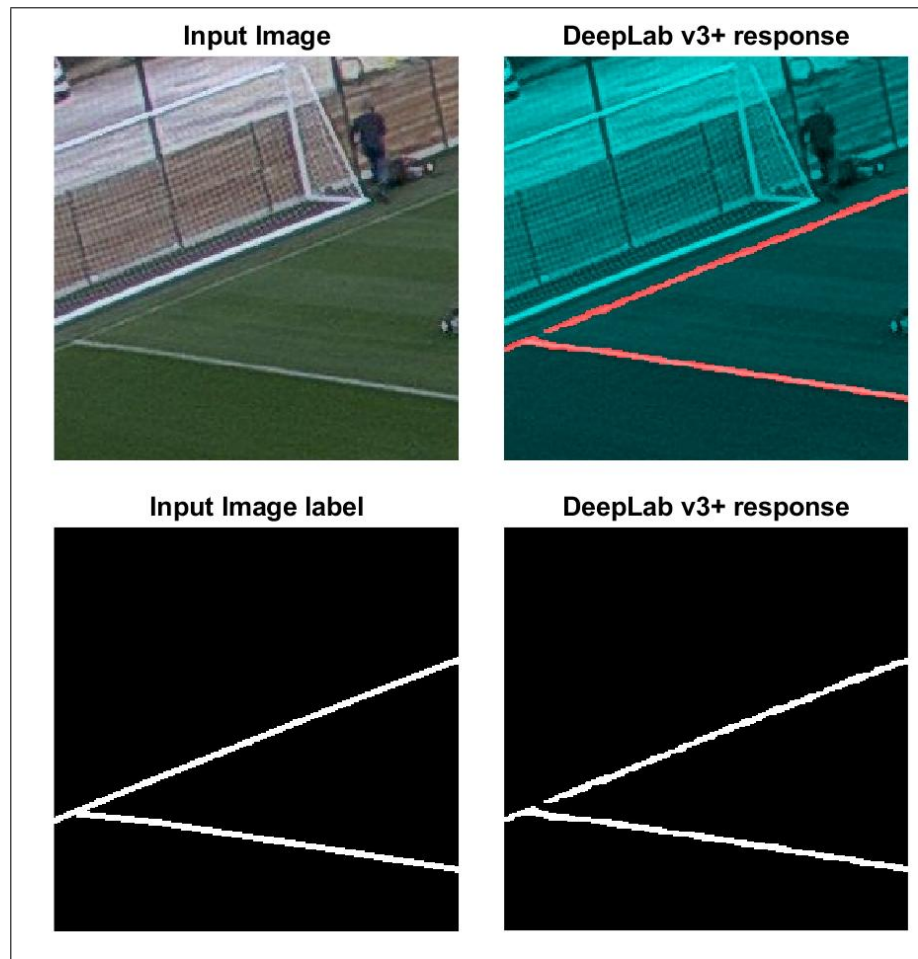
Figure 4.3 and 4.4 display line detection from different pitches, strictly using input images where the camera is positioned from the side (which is the same camera positioning that the network was trained with). Note that the camera angle differs slightly between the images. Figure 4.5 displays a more zoomed-in result from the DeepLab v3+ network: the result from one of the cut-outs during training. Note the accuracy and similarity to the label.



**Figure 4.3:** Three sets of image pairs with inputted image on the left and outputted line detection (white points) on the right. Note that all these images are of the right half of the pitch.



**Figure 4.4:** Three sets of image pairs with inputted image on the left and outputted line detection (white points) on the right. Note that all these images are of the left half of the pitch.



**Figure 4.5:** Four images depicting the DeepLab v3+ network response on one of the input cut-outs. Top-left displays the input cut-out. Bottom-left displays the label for the cut-out. Bottom-right shows the pure network response and top-right overlays the network response (red) on top of the input image (with a blue tint to clarify the network response further.)

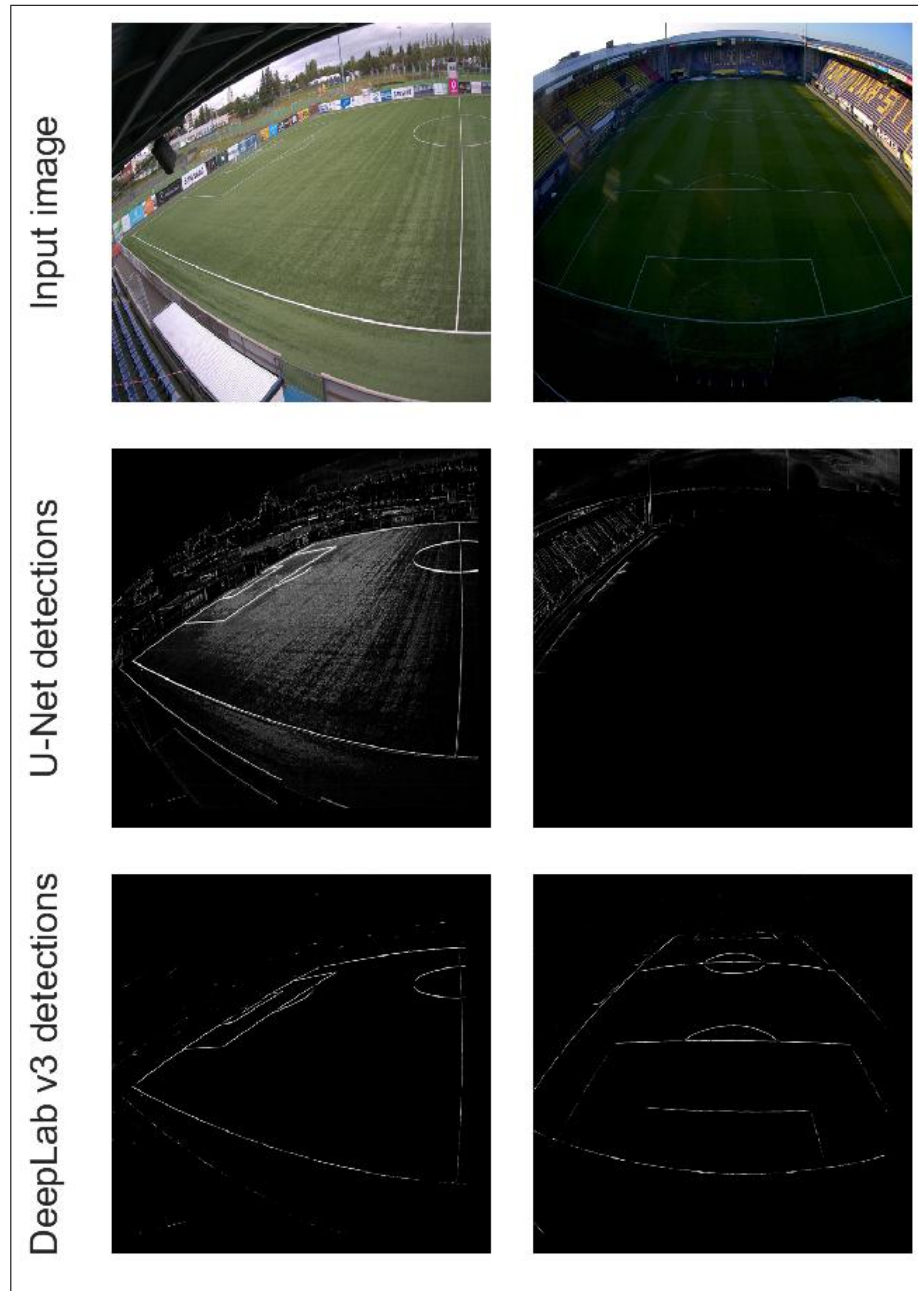
### 4.1.3 Generalization

From the final U-Net and DeepLab v3+ networks, some images with vastly different camera positions, noise and line visibility were tested. In Figure 4.6 two different camera positions are tested: from the side and from behind the goal. Note how the DeepLab v3+ network has a much higher accuracy than the U-Net network for both input images - with the U-Net network having very low accuracy for the image with the camera positioned behind the goal (Figure 4.6, mid right).

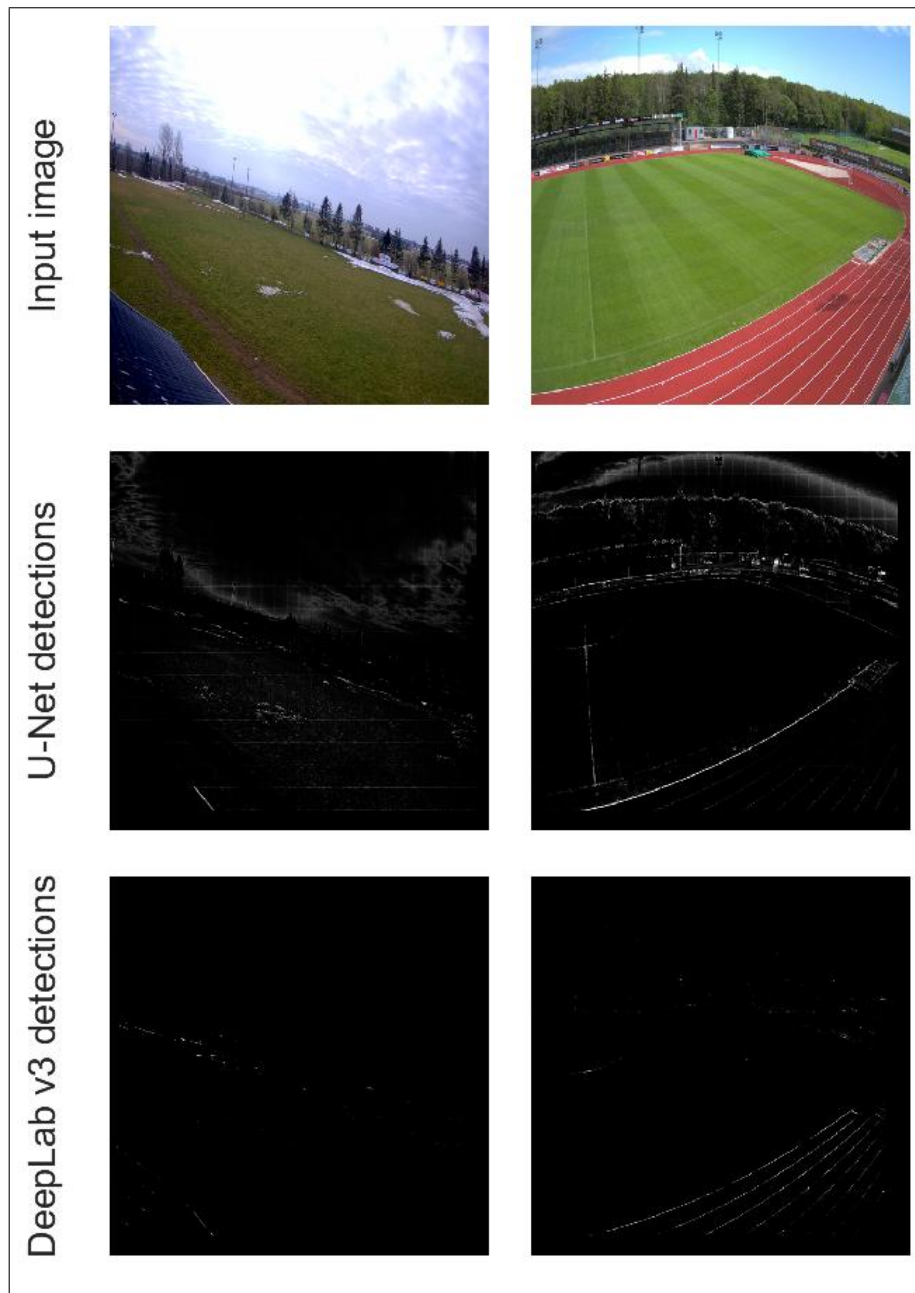
In Figure 4.7, two different inputted images with (left) poor line visibility and (right) white lines (noise) outside the pitch. Observe how the U-Net network finds substantially more detections on the pitch for both images, while also finding many detections from the background. Additionally, for the right input image, the U-Net networks has much fewer detections from the running track outside the pitch.

In Figure 4.8, the network responses to having much noise on the pitch itself (snow) are displayed. Note the difference between the amount of detections of both lines and noise. In Figure 4.9, note the impact of additional lines on the pitch.



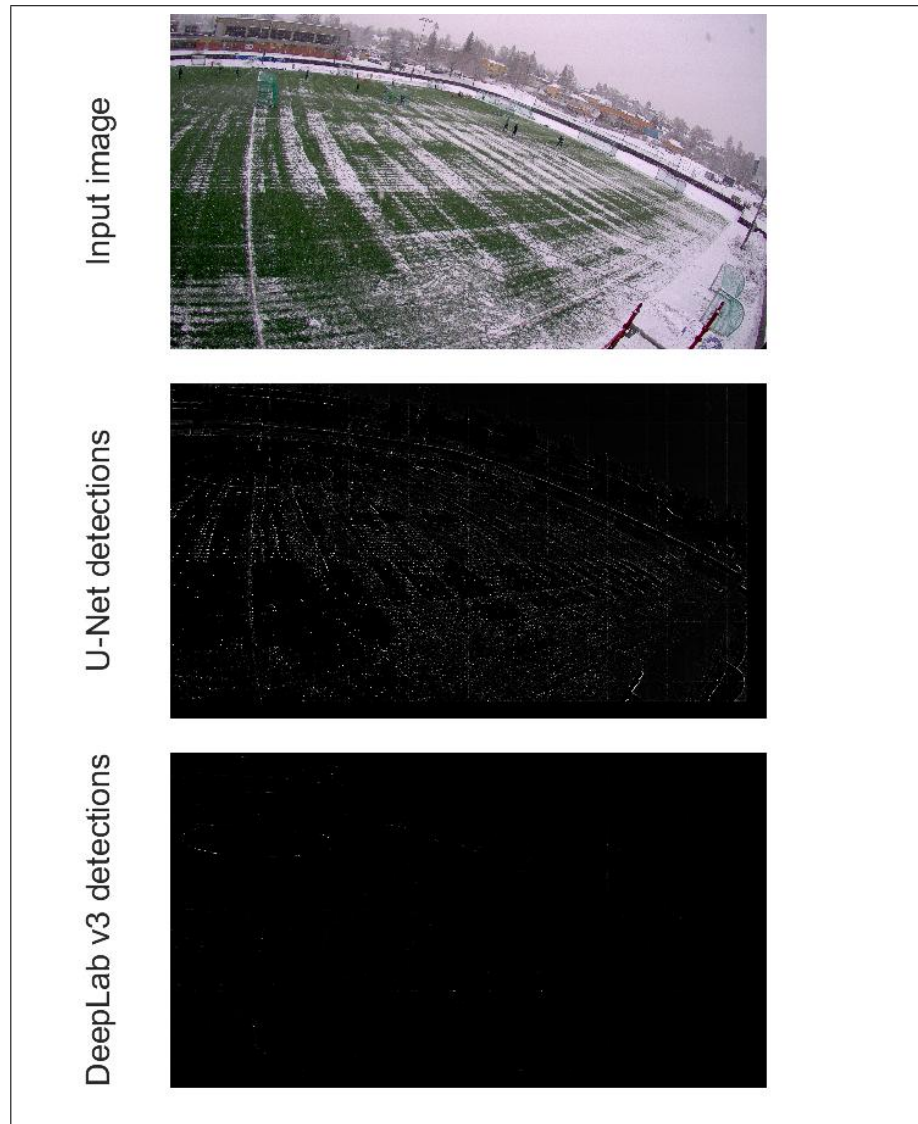


**Figure 4.6:** Six images displaying the points (white) from the U-Net (2nd row) and DeepLab v3+ (3rd row) network detections from input images (1st row) with different camera positions. Note that the U-Net does not find any lines on the pitch when the camera is positioned behind the goal while the DeepLab v3+ seems have similar accuracy independent on if the camera is positioned behind the goal or from the side.

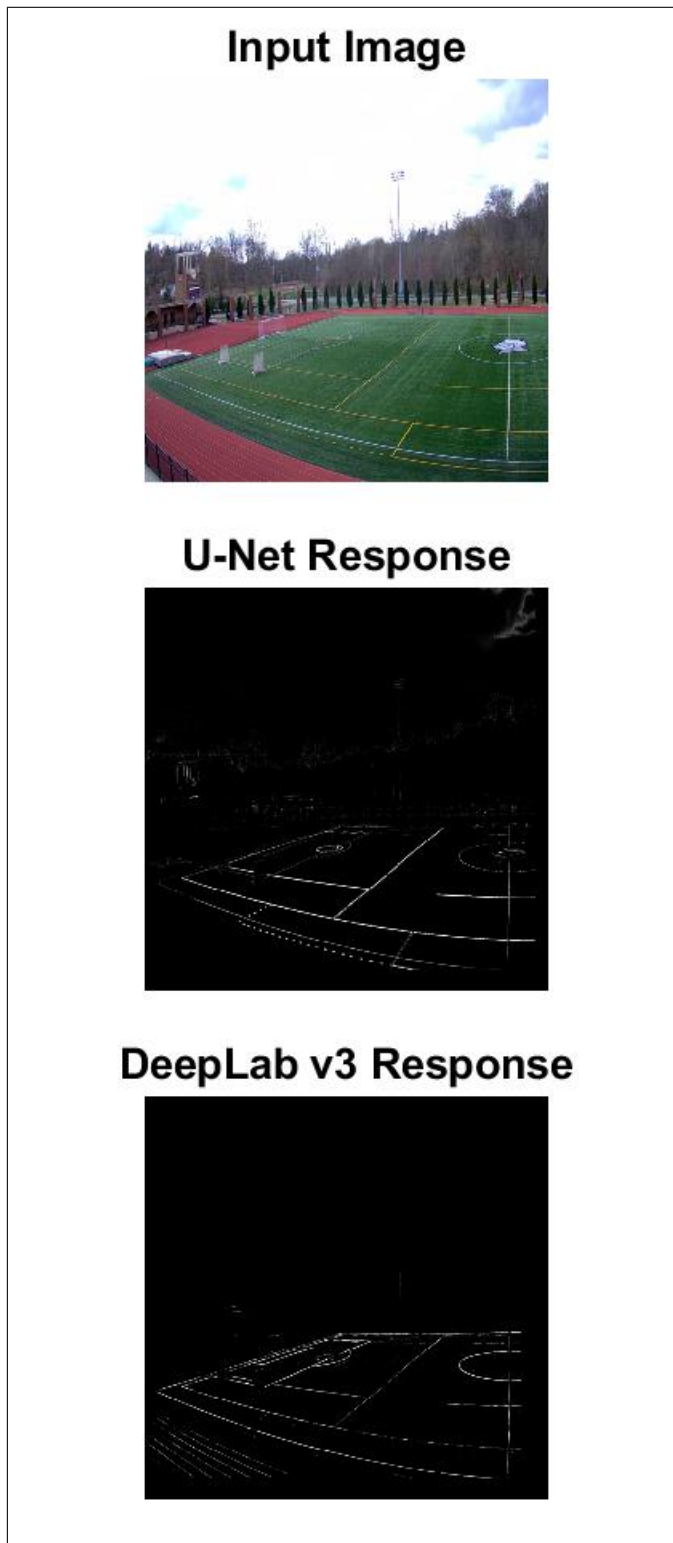


**Figure 4.7:** Six images displaying the points (white) from the U-Net (2nd row) and DeepLab v3+ (3rd row) network detections from two different input images (1st row). Note that the DeepLab v3+ network has very few detections for both images while the U-Net network has more detections from both the pitch and from the background.





**Figure 4.8:** Three images displaying the input and detected lines for the U-Net and DeepLab v3+ network when there is snow on the pitch. Note the difference between finding many detections including lines and noise for the U-Net network to finding few detections from either lines or noise in the DeepLab v3+ network.



**Figure 4.9:** Images displaying ML network responses to the input image having additional lines on the pitch itself

## 4.2 Ridge Detection and Sub-Pixel Optimization

The following section displays a few images for both of the computer vision techniques used to mainly remove outliers and save more accurate data.

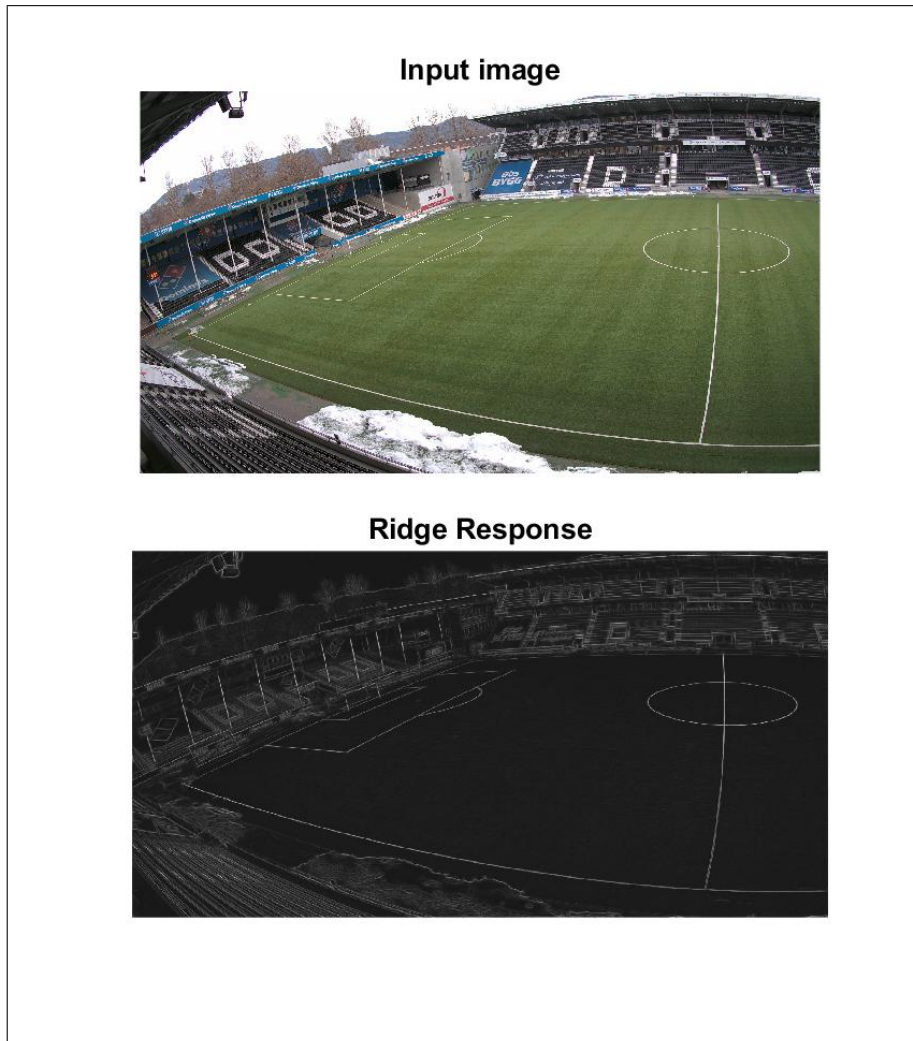
An image and the resulting ridge detection are displayed in Figure 4.10. The upper image in the figure displays the input and the lower image the ridge response. Note that the algorithm is run on a "normal" input image, not an image from the ML network.

In Figure 4.11 and 4.12, the sub-pixel optimization is shown. The bottom image in Figure 4.11 displays how the optimization works. Note that the sub-pixel algorithm uses a combination of the ML output and local maximums from the ridge response. Figure 4.12 displays the usage of the sub-pixel optimization to remove the worst outliers.

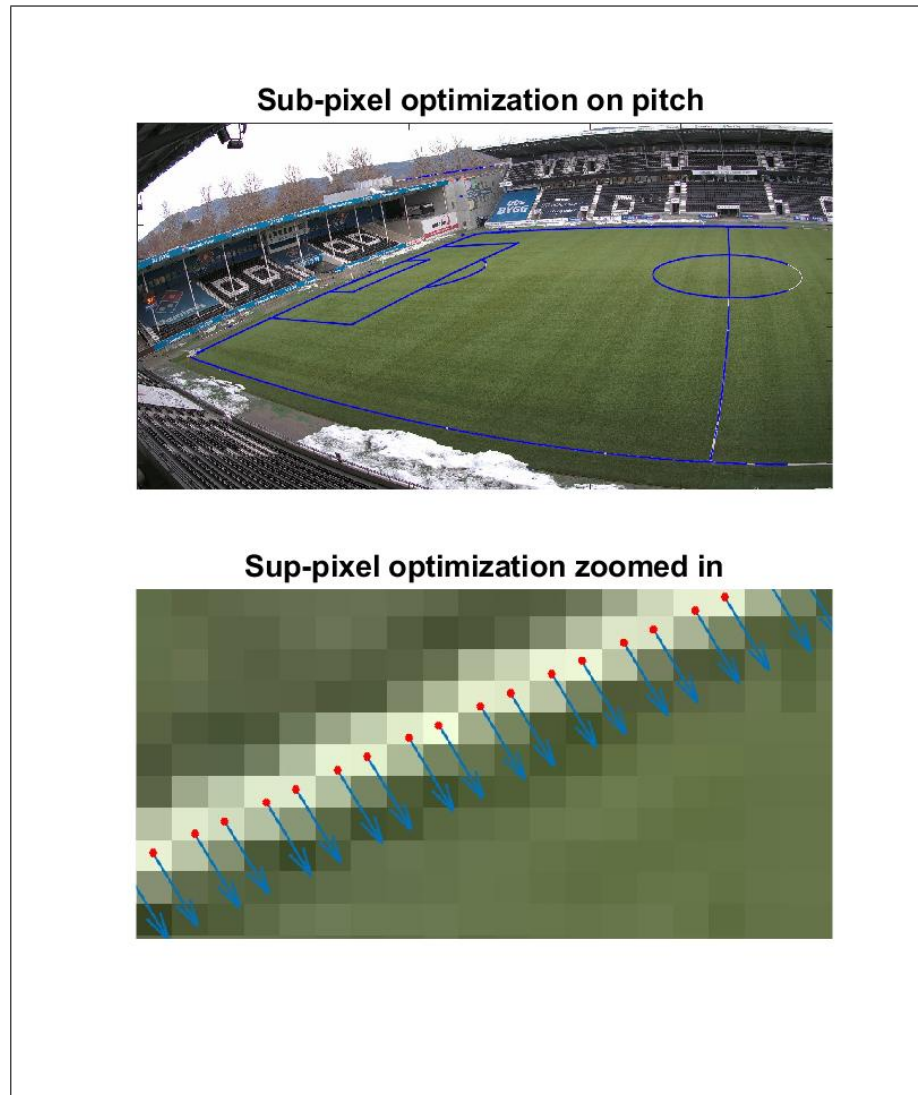
## 4.3 Line Classification and Output Generation

Figure 4.13 displays the line classification points (top) and the polynomial fitting of these points (bottom). The different colors indicate that the points are from different lines on the initial football field.

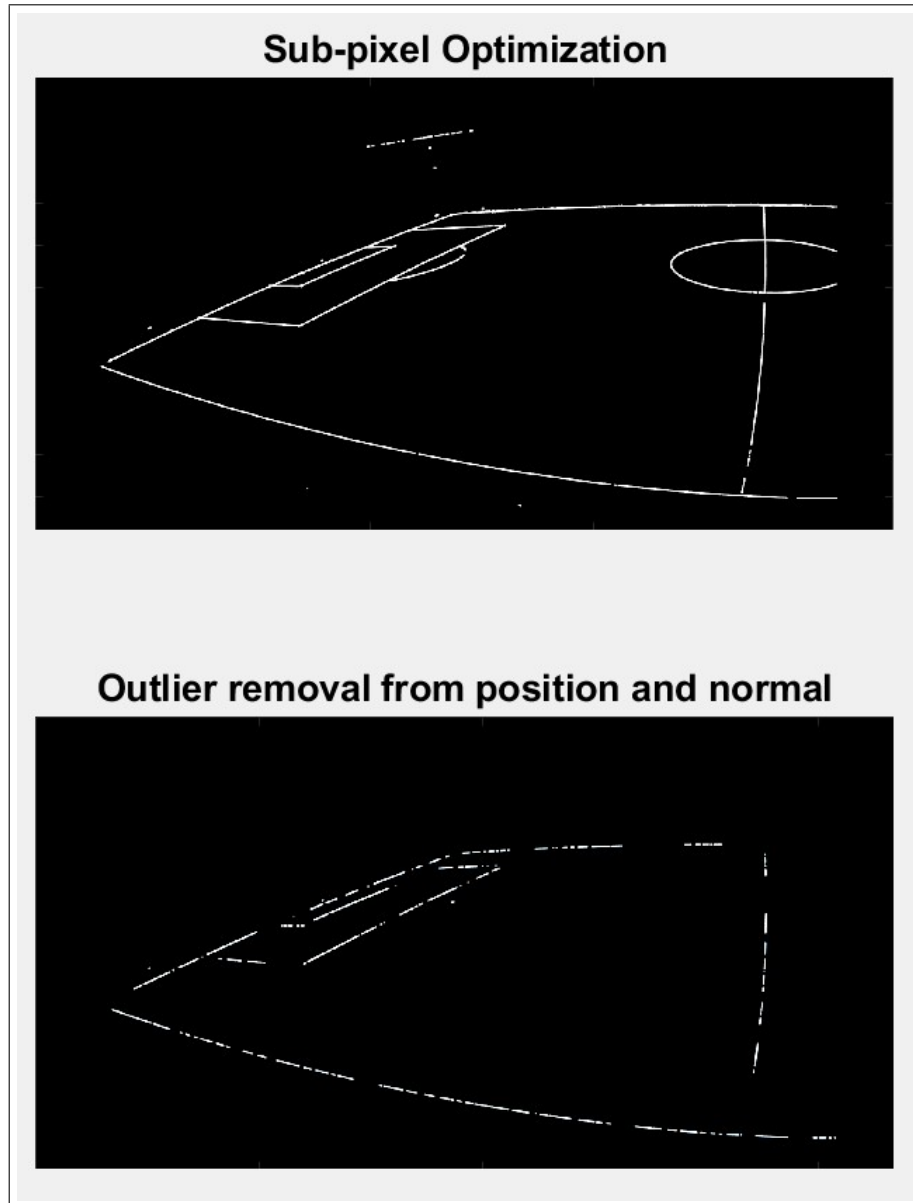
In Figure 4.14, 4.15 and 4.16, saved points from the earlier polynomial fitting are displayed. Note in Figure 4.14 that the upper image shows the generated points while the lower image is from an old calibration of the same football field. Figure 4.15 displays another iteration of the full project model with the inputted image (top), generated points (middle) and old manually extracted points (bottom). Note that points for each line are saved in different vectors. Finally, Figure 4.16 displays the data file generated from the project and the old manually created file for the same pitch.



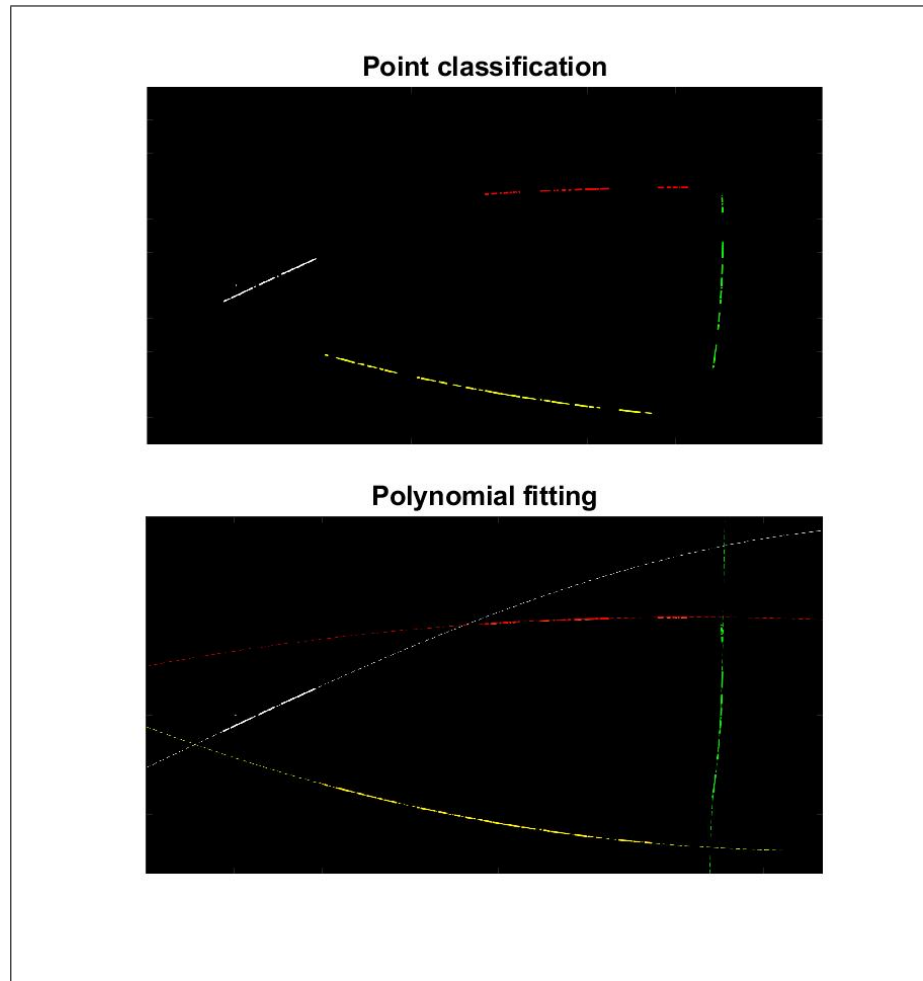
**Figure 4.10:** Two images showing the input(top) and ridge response (bottom) when running the input through all the elongated Gaussian filters. Note that the bottom image displays local maximums.



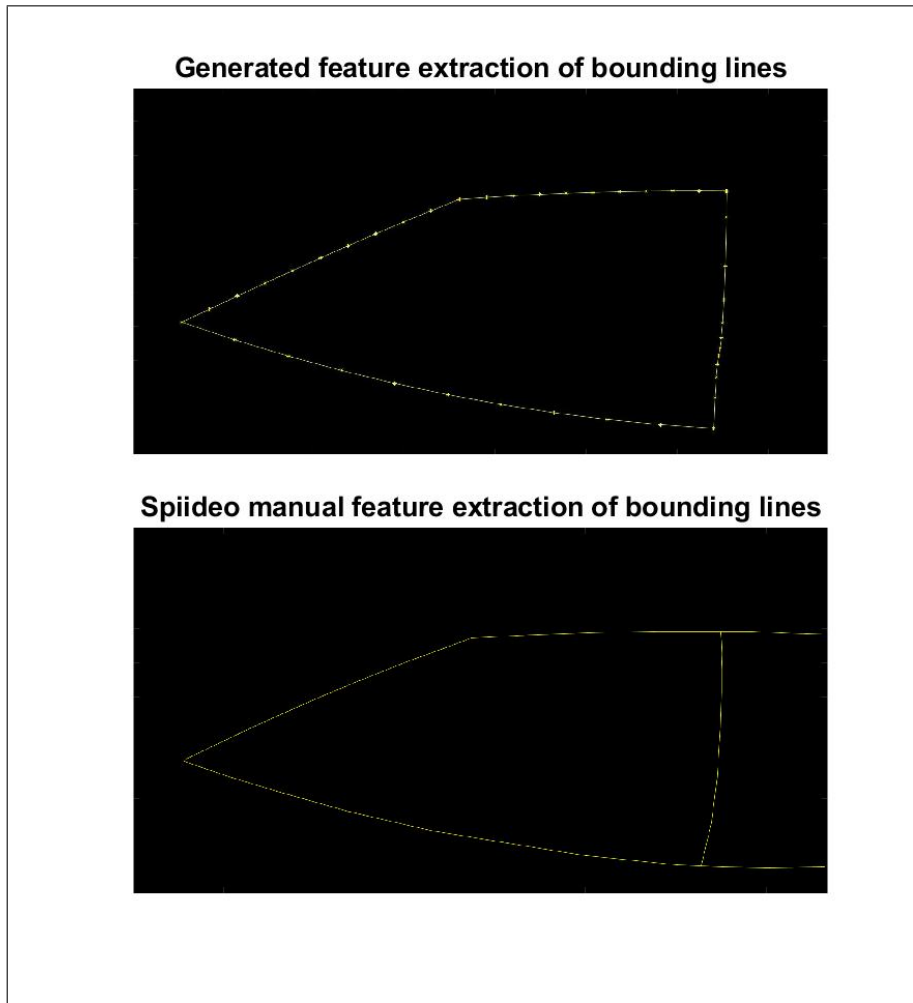
**Figure 4.11:** Two zoom-levels of the same image. The upper image shows a football field with blue markings from the sub-pixel optimizations plotted. The lower image is a zoomed in version on one of the lines in the upper image. Note the red point (which is the detection sub-pixel) and blue arrow (indicating the normal of the detection point).



**Figure 4.12:** Two images displaying remaining points after sub-pixel optimization (top) and outlier removal (bottom). Note the differences between the images.

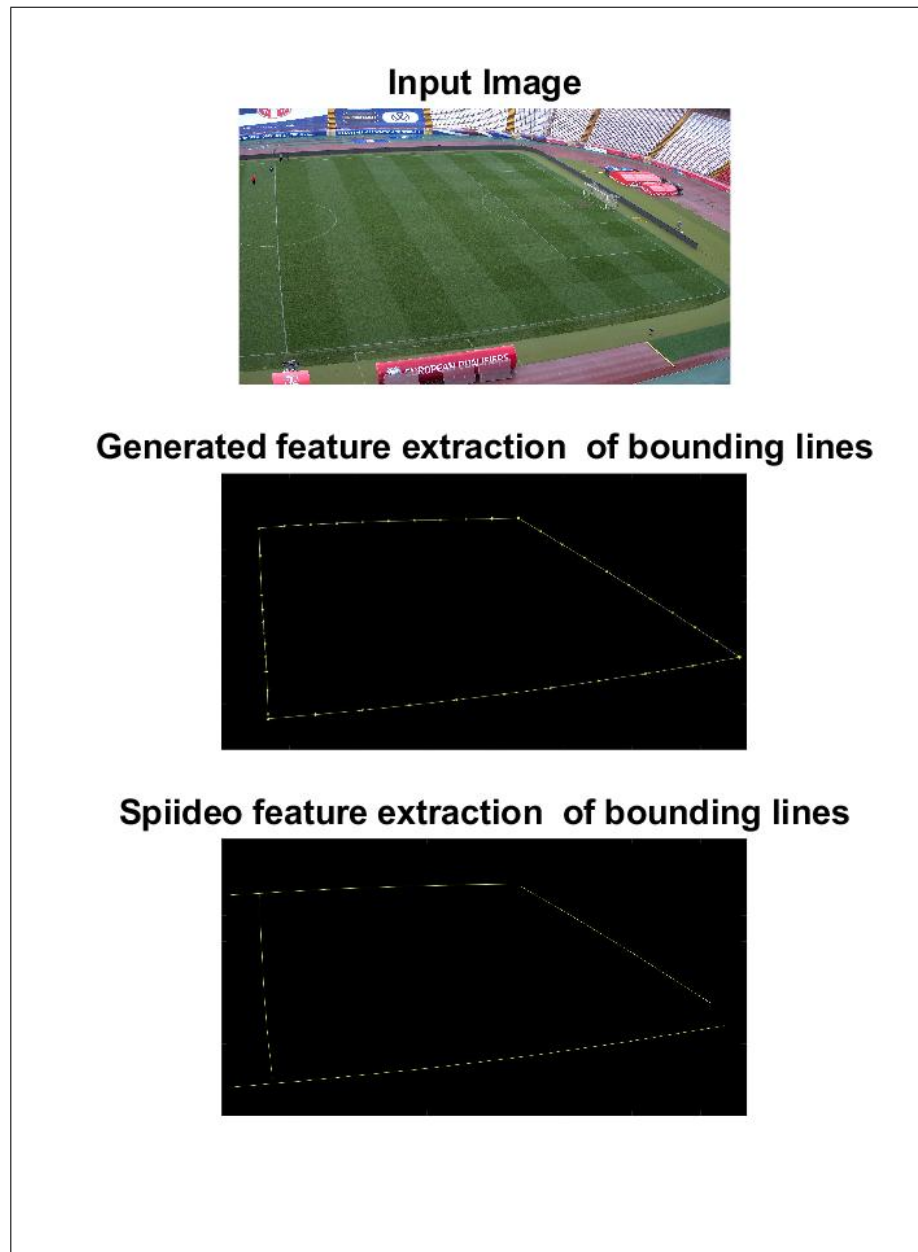


**Figure 4.13:** Upper: points saved for each line (different colors for each line). Bottom: Polynomial fitting, 2nd deg, of the points in the upper image.



**Figure 4.14:** Two images showing both the saved points from the project (top) and the manually saved points from the old calibration (bottom). Note the similarities here.





**Figure 4.15:** Three images showing the input image (top), the automatically generated output (middle) and manual calibration (bottom). Note the similarity of the two lower images.

Generated feature data-file	Spiideo old feature data-file
<pre>--- image_size: width: 3840 height: 2160 points: - - - - 278.7271 - 1377.0727 - - 430.7271 - 1300.5126 - - 582.7271 - 1224.5172 - - 734.7271 - 1149.2745 - - 886.7271 - 1074.9729 - - 1038.7271 - 1001.8006 - - 1190.7271 - 929.9456 - - 1342.7271 - 859.5963 - - 1494.7271 - 790.9409 - - 1646.7271 - 724.1675 - - 1798.7271 - 659.4644</pre>	<pre>--- image_size: width: 3840 height: 2160 points: - - - - 2577 - - 2135 - - 2509 - - 2118 - - 1926 - - 1941 - - 1105 - - 1595 - - 487 - - 1261 - - 141 - - 1029 - - 3254 - - 1493 - - 3025 - - 1449 - - 2811 - - 1407 - - 2784 - - 1399 - - 2767 - - 1394</pre>

**Figure 4.16:** Two text files both displaying image size and saved points for two different pitches. Left: generated file. Right: manually created file. Note the similarities. Also note that integers (for the points) start with one dash if its the y-value and two dashes if its the x-value. Additionally, three dashes indicate a new line and four lines are used for the first of all the lines



*“The aim of argument, or of discussion, should not be victory, but progress.”*

---

*Joseph Joubert*

## 5.1 ML Networks

When observing Table 4.1, note how the accuracy increases steadily with a larger data set. However, during training, a balance between down sampling the images and computing time had to be found. Either information would be lost due to the down sampling (giving more faulty detections) or the network would require larger amounts of computational resources. This can also be noted in Table 4.1 when comparing the training time between the larger-sized network to the smaller-sized ones. Additionally, consider Figures B.1 and B.2 in Appendix B, where the accuracy increases with more epochs (longer training). In theory, the network would reach a point where validation accuracy starts to decrease due to overtraining, however that is not observable in these results from the training processes. Hence, with more computational resources the network could be trained for longer (more epochs) and with larger-scale images to lose less information, likely leading to higher accuracy.

From Figures 4.1, 4.2 4.3 and 4.4, note that the DeepLab v3+ network outputted better results than the U-Net network when trained the same data set (Trained on 600 cut-outs of size 128x128px from 6 different arenas and Validated on 120 cut-outs from 2 different arenas.) Highest test accuracy for DeepLab v3+ was 98.22% whereas the test accuracy for U-Net was 94.67%. Additionally, comparing Tables 4.1 and 4.2, training for the DeepLab v3+ network also ran faster and required less computational power. Thus, larger images could be used for training, resulting in the higher test accuracy of 98.34% when training the network on 512x512 sized inputs.

When observing Figures 4.6, 4.7 and 4.8, note how images with more "noise"

from e.g. snow, different camera positions, fences and other lines in close proximity to the pitch reduces the accuracy of the networks. Particularly, the DeepLab v3+ network is having dramatically reduced accuracy when the lines in the image are not clearly visible to the camera, see Figure 4.6. The U-Net network instead is having difficulties with most noise, especially from the stadium, and has close to zero true detections when the camera is positioned behind the goal, see Figures 4.6. However, the U-Net network does seem to find more true points on the figures with poor line visibility than the DeepLab v3+ network, see Figure 4.7.

### 5.1.1 Generalization

Looking at Figure 4.2, 4.4 and 4.6, note how the line detection seems to work with a similar accuracy for all images when using the DeepLab v3+ network. The accuracy of the network appears to rely less on the camera angle and more on how visible the lines in the image are, compare to Figure 4.7 where the angles are similar to before, however the resulting accuracy is much lower. This likely means that the DeepLab v3+ network works well for most camera angles and positions as long as the lines are clearly visible. Even better results for the images from behind the goal (and with very different camera angles from the used training data) could be acquired by using more of these images during training. A larger data set with more variation in terms of camera angles, camera positions and weather conditions would increase the general accuracy for the network. For the U-Net network, the resulting image seems to depend heavily on the camera position and amount of noise. This could be due to the fact that the network could not handle very large input images for training and much information was lost. Regarding noise, the U-Net network seems to find more false positives from the background, see Figure 4.6. However, for images with poor line visibility and running tracks, the network more accurately finds detections on the pitch (but also more noise) than the DeepLab v3+ network, see Figure 4.7. This could be due to the apparent lower threshold for detections in the U-Net. Finding more points while also finding more noise could be okay if false positives could be removed in a later stage. For a better performance, larger parts of the 4k-resolution images (either by down-scaling or using larger cuts) have to be trained on to retain more of the U-Net theoretical capabilities of retaining information between convolutions. Furthermore, using more variety of images would increase the generalization of the network for different angles and camera positions.

## 5.2 Computer Vision Aid

In Figure 4.10, the local ridge-maximums after running the native image through all elongated Gaussian filters are shown for a given input image. By using these ridges in combination with the ML networks' detected points from the lines; some of the worst outliers could be removed. This can be observed in Figure 4.12 where the saved points are combining results from the ML network and local maximums of the ridges.

By observing Figure 4.11, showing the resulting image after running the sub-pixel optimization (blue) on an ML and ridge detection combination, one might first not see anything special (top image). However, looking at the zoomed-in image (bottom), the sub-pixel optimization algorithm is more clearly showing both the normal and sub-pixel point. Figure 4.12 shows the same sub-pixel optimized image, but before (top) and after (bottom) removal of all the smallest sequences of points with a similar normal and coordinates. Using the normal and positions of detected points seems to be a working strategy, however when removing more of the sequences, some true positives were also lost. Therefore, a balance had to be found where most of the outliers from noise were removed while still retaining enough detection points for each of the four bounding lines (including the halfway line). When observing the manual calibration data files (see Figure 3.2), only around 5-10 points per line were saved. Therefore, choosing to remove more noise with the consequence of losing detection points on the lines appears reasonable as long as enough points remain to recreate the entire line. Figures 4.13 and 4.14 show the progression from the removal of outliers to final vectorization of the lines. Note how only few segments of points (around 50 points in relatively close proximity with a similar normal) are enough to almost fully recreate the bounding lines through polynomial fitting.

The key thing from the ridge detections was removing some of the outliers, however this step seems unnecessary in comparison to how much more outliers the later methods remove. The method does help in regards to reduce time of the sub-pixel optimization. Since the optimization is run on every detected point saved from the combination of the ML network and ridge detection, removing points early significantly reduces the run time and computing power required. However, using the sub-pixel optimization more outliers are removed (including most of the outliers that would be removed from the ridge detection).

### 5.3 Visual Effects

First, the visibility of the lines in the original image seems to be the most impactful for the accuracy of the ML models (specifically the more accurate DeepLab v3+ network) as previously discussed (Figure 4.6 and 4.7). Furthermore, since the detected points from the ML model are the basis for the rest of the models, the visibility of the lines are vital for the entire process to succeed. If the ML model outputs very inaccurate results, there is no way to reproduce enough information for successful classification of the different lines.

The main noise resulting in false detections being saved seems to come from either the arena itself, the pitch, advertisements or the background. This can be seen clearly in Figure 4.1 and 4.3, where notably all the extra detections are due to these factors. Additionally, due to how the training was performed for the networks, false positives from pitches with additional lines around or on the pitch are typically saved, see the right column of Figure 4.6 where the running track is detected and Figure 4.9 where false positives from both the pitch and running

track are saved.

Observing the visual impact of sub-pixel optimization and line classifications from section 4.2 and 4.3, note how the image after sub-pixel generation (upper image of Figure 4.12) still retains a very complete image of the pitch, with some outliers. When using the sub-pixel optimized points to classify the lines however, a lot of true-positive data points are lost as well (Figure 4.13). A different approach (maybe using a second ML model) to classify more of the true positive points would more accurately depict the polynomial fitting - hence leading to more accurate results for later calibration stages.

Comparing the generated data files from the system to the manually created files acquired from the company, the similarities are striking (Figure 4.16). When separating the images into four parts, some manual labour was done to extract the correct line. A more robust approach would be required here for the classification step to be fully automatized independent on the input image. As it stands, manual observations have to be made to separate the lines correctly. For an image with more noise from lines, the classification becomes even harder due to these lines fitting the criteria of having similar coordinates and normal.

## 5.4 Comparison to Manual Work

Looking at the bottom part of Figure 4.13 and 4.15 compared to the ML outputs (Figure 4.1, 4.2, 4.3 and 4.4), note how the network finds more detections than the amount of points chosen during manual calibration. Therefore, a more strict threshold of the ML model can be used to only choose points which the network is very certain is on a line. Furthermore, another way this can be done is by changing how strict the removal of points from sub-pixel optimization is, see Figure 4.12. Removing a larger percentage of points (only saving e.g. longest segment for each direction or saving points which are even closer in terms of normal and coordinates) would lead to more outliers being removed at the cost of saving less true positives. As discussed however, this might not be a problem if enough true points remain to recreate the lines.

When observing the ML predicted outputs, one should note the difference between false negatives (not detecting a line) and false positives (detecting unwanted lines), and how these affect the outcome of the entire project. With false negatives, line segments are not detected by the network. However, for false positives, additional points from other sources than the lines are detected and saved. As previously stated, the manually saved points for previous calibrations do not typically have more than 20 points per line. Therefore, the occurrence of false negatives might not be concerning as long as there are other points saved on the line (preferably spread out). This is following the same thought process as the previous discussion. In the case of false positives, where additional points are saved from "noise", the problem is far greater. These outlying points, if saved, will obviously create problems when trying to estimate polynomials for each line during later stages of

the calibration. One interesting remark arising from this train of thought is "how many points are needed to get enough information on a line for the calibration?". If the image can retain enough information for each line when removing all false positives, the current calibration techniques following these results will suffice.

Compare the manually created data files (and their plotted points) with the generated ones in Figures 4.14, 4.15 and 4.16. Note resemblance between the generated one and the manually created one, where the generated one even has more decimal places, using the sub-pixel points. It does not, however, include the additional intersections between the halfway line and the central circle which are saved in most manually created data files used in old calibrations.

Spiideo could use the created process flow to automatically generate the informational data files of feature extraction (the bounding lines on a new pitch), hence removing some of the currently manual labour at the start of the calibration process for the cameras.

## 5.5 Future Prospects

For a project of this scale, time is often the limiting factor from testing everything of interest. In this section some of these ideas for future improvements and additions are stated.

- **Automating the extraction of camera parameters.** The natural next step is using the results of this project to automatically extract the camera parameters, hence making the entire calibration process automatized.
- **Server-side coding.** The code is currently written in MatLab locally on a computer. An improvement to this would be writing the code to a server (e.g. Amazon Sagemaker and Amazon AWS) in python, and letting the user use the model from any computer, without having the code or ML network downloaded locally.
- **More robust classification of each line.** To use the full method on a large data set or any new data, a more robust and accurate classification is vital for the final output generation.
- **Using a larger ML network for line detection.** There is a lot of old calibrations that can be used to train a much larger ML network, which possibly would increase the accuracy even further. Thus making the subsequent steps more accurate as well.
- **User interface.** One could create a more interactive calibration where the user can e.g. observe the detected points (and change them manually if needed), choose the threshold of outlier removal, line classification etc.
- **In depth analysis of the removal of outliers.** Understanding the effects of outliers more accurately, and being able to remove enough of them while retaining enough information about the lines would increase the model accuracy for all the output steps in this project. An interesting question here



is the balance of using more or less points from both the true detections and noise - what is the optimal approach?

- **Generalization of other sports.** Adding on to the project results, one could use more inputted training data from different sports to create a more generalized network which works independently on which sport is observed.
- **Camera calibration revamp.** In this project, the models output true detections from all of the real lines on a pitch (even penalty box, goal box). However, currently the later camera calibration only uses the outermost bounding lines. It would be of great interest to me personally to see if there could be improvements made to the final parameter calibration by using more of the detected lines from the other parts of the pitch as well. Typically the penalty box has set dimensions while the length and width of a football field can vary. Therefore, a camera calibration using the penalty box would not require the additional information about pitch-size which is currently required.

*“It is good to have an end to  
journey toward; but it is the  
journey that matters, in the  
end.”*

---

*Ursula K. Le Guin,  
The Left Hand of Darkness*

## 6.1 From Experimental Data

The experimental data (4k images with correlated information from old calibrations) was vital to create the training set for the ML networks. Having an even larger data set would most likely increase the accuracy of the networks. However, due to the fact that the manually created calibration files do not include some of the lines (penalty box, center circle etc), one would have to manually create a more complete label for more files - which would take a lot of time. Thus, another approach could be to train a network to only find the outer lines for the pitch only.

## 6.2 Network Analysis

The ML networks work well for images with conditions where the lines are clearly visible and with few disturbances. However, for less clear images where the lines are more obstructed or unclear, the network typically outputs poor results with non-salvageable detections. The DeepLab v3+ network ran faster than the U-Net network and could therefore handle larger data sets, a larger amount of epochs and even bigger input images. Consequently, the DeepLab v3+ network outputted better results when finding the lines of the pitch.

## 6.3 Usage of Computer Vision

The usage of computer vision techniques were highly dependent on how well the ML network predicted the lines. A poor prediction lead to many false positives

from noise being saved for future calibrations. When the ML network performed well however, the ridge detection and sub-pixel optimization helped reduce outliers by comparing the positions and normal of nearby points. More work has to be made to fully optimize the removal of outliers and more rigidly classify the point detections depending on their respective line.

## 6.4 Observations

The ML network finds more lines (penalty box, goal box and center circle) than what the manually created calibrations have used. This additional information is lost during the final few stages to replicate the old data files. Using the additional lines from e.g. the penalty box could be beneficial to the calibration of the camera parameters in later stages.

## 6.5 Reflections

Understanding the problem was one of the key parts of this thesis project. However, due to the different approaches used - the problem was continuously evolving. In the end, more time should have been put into classification of the individual lines. When looking more closely at the manual calibration files, not many points for each lines seem to be needed to fully calibrate the camera. For this thesis project, most of the time was used to create and improve on the ML network to extract all the lines from a pitch. From the detected points, classifying which points are on each line would be vital to generate similar output files as for the manual calibrations.

One of the main problems in doing the project from home was the lack of computing power, making training very slow. Much of the frustration during the project was due to long training sessions crashing midway through. However, executing this project was in many ways very interesting with much problem solving at every turn and the core concepts used and the discussions on the way have been vital to my future understanding of the field.

Comparing the results to the aim of the thesis, the feature extraction was successfully automated in terms of finding the lines on a football field. However, this thesis did not completely succeed in classifying each of the found lines in a generalized way and therefore some additional work has to be made to completely automate the entire feature extraction-to-file process. Hopefully this project can be a good starting point for future endeavours. The other objectives were to gain understanding of the ML field, computer vision techniques and also trying to generalize the feature extraction. In regards to the generalization, I concluded that with more training on different angles and camera positions, the line detection could work as good independent on where the camera is placed for new calibrations. Additionally, I concluded that there already is some generalization for the DeepLab v3+ network when the image is clear (visible lines) with good conditions (lower amount of noise).

---

## References

---

- [1] J. Hammerschmidt, S. Durst, S. Kraus, K. Puumalainen, *Professional football clubs and empirical evidence from the COVID-19 crisis: Time for sport entrepreneurship?*, Technological Forecasting and Social Change, Volume 165, 2021, <https://doi.org/10.1016/j.techfore.2021.120572>
- [2] D. Edgar, S. Edgar, *Maintaining Football Club Identity During COVID-19: The Case of the English Premier League.*, <http://doi:10.4018/978-1-7998-6780-7.ch002>, Impacts and Implications for the Sports Industry in the Post-COVID-19 Era (pp. 13-39), IGI Global, 2021
- [3] D. Purpura, *COVID-19 pandemic: Online broadcasts keep fans connected to favorite teams.*, URL: <https://bit.ly/2Rn0pk0>, ThisWeekNews, (accessed: 2021-05-16).
- [4] M. A. Gomez-Ruano, S. José Ibáñez, A. S. Leicht, *Editorial: Performance Analysis in Sport*, 10.3389/fpsyg.2020.611634, Front Psychol, 2020
- [5] L. C. Chen, G. Papandreou, F. Schroff, H. Adam, *Rethinking Atrous Convolution for Semantic Image Segmentation*, <https://arxiv.org/pdf/1706.05587.pdf>, CoRR, 2017
- [6] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A. L. Yuille, *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*, <https://arxiv.org/pdf/1606.00915.pdf>, CoRR, 2017
- [7] D. S. Farin, S. Krabbe, P. H. N. de With, W. W. J. Effelsberg, *Robust camera calibration for sport videos using court models*, <https://doi.org/10.1117/12.526813>, Proc. SPIE 5307, Storage and Retrieval Methods and Applications for Multimedia, 2004
- [8] L. Sha, J. Hobbs, P. Felsen, X. Wei, P. Lucey and S. Ganguly, *End-to-End Camera Calibration for Broadcast Videos*, 10.1109/CVPR42600.2020.01364., 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp 13624-13633), 2020
- [9] The MathWorks *Convolutional Neural Network*, URL: <https://se.mathworks.com/discovery/convolutional-neural-network-matlab.html>, (accessed: 2021-03-19).

- 
- [10] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, <https://ieeexplore.ieee.org/document/726791>, Proceedings of the IEEE, 1998
- [11] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, <http://www.deeplearningbook.org>, MIT Press, 2016
- [12] D.C. Ciresan, U. Meier, J. Maşci, L. M. Gambardella, J. Schmidhuber, *High-Performance Neural Networks for Visual Object Classification*, <http://arxiv.org/abs/1102.0183>, CoRR, 2011
- [13] L. N. Smith, *A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay*, <http://arxiv.org/abs/1803.09820>, CoRR, 2018
- [14] M. Ohlsson, P. Edén, *Lecture Notes on Introduction to Artificial Neural Networks and Deep Learning (FYTN14/EXTQ40/NTF005F)*, Department of Astronomy and Theoretical Physics, Lund University, 2020
- [15] A. Ng *CS294A Lecture notes: Sparse autoencoder*, URL: [http://ailab.chonbuk.ac.kr/seminar\\_board/pds1\\_files/sparseAutoencoder.pdf](http://ailab.chonbuk.ac.kr/seminar_board/pds1_files/sparseAutoencoder.pdf), Stanford University (accessed: 2021-04-09)
- [16] M. Chen, X. Shi, Y. Zhang, D. Wu, M. Guizani, *Deep Features Learning for Medical Image Analysis with Convolutional Autoencoder Neural Network*, 10.1109/TBDATA.2017.2717439 IEEE Transactions on Big Data, 2017
- [17] O. Ronneberger, P. Fischer, T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, <http://arxiv.org/abs/1505.04597> CoRR, 2018
- [18] K. B. Chen, Y. Xuan, A. J. Lin, S. H. Guo, *Lung computed tomography image segmentation based on U-Net network fused with dilated convolution*, <https://doi.org/10.1016/j.cmpb.2021.106170> Computer Methods and Programs in Biomedicine, Volume 207, 2021
- [19] L. C Chen, Y. Zhu, G. Papandreou, F. Schroff, H. Adam, *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*, <http://arxiv.org/abs/1802.02611>, CoRR, 2018
- [20] T. Lindeberg, *Edge Detection and Ridge Detection with Automatic Scale Selection*, <https://doi.org/10.1023/A:1008097225773>, International Journal of Computer Vision, issue 30 (pages 117–156), 1998
- [21] J. Damon, *Properties of Ridges and Cores for Two-Dimensional Images*, <https://doi.org/10.1023/A:1008379107611>, Journal of Mathematical Imaging and Vision, issue 10 (pages 163–174), 1999
- [22] L. C Chen, Y. Zhu, G. Papandreou, F. Schroff, H. Adam, *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*, <http://arxiv.org/abs/1802.02611>, CoRR, 2018

- 
- [23] K. Astrom, A. Heyden, *Stochastic modelling and analysis of sub-pixel edge detection*, <https://doi.org/10.1109/ICPR.1996.546729>, Proceedings of 13th International Conference on Pattern Recognition, Volume 2 (pages: 86-90), IEEE, 1996
- [24] C. Steger, *Subpixel-precise extraction of lines and edges*, [https://www.isprs.org/proceedings/XXXIII/congress/part3/141\\_XXXIII-part3.pdf](https://www.isprs.org/proceedings/XXXIII/congress/part3/141_XXXIII-part3.pdf), International archives of photogrammetry and remote sensing, Volume 33, Number 3 (pages (141-156), 1999
- [25] K. Astrom, A. Heyden, *Stochastic analysis of image acquisition, interpolation and scale-space smoothing*, Advances in applied probability (pages 855-894), JSTOR, 1999
- [26] F. Nagase, *Analysis of structure and movement of neural growth cone*, <http://lup.lub.lu.se/student-papers/record/4934064>, Master's Theses in Mathematical Sciences, 2014



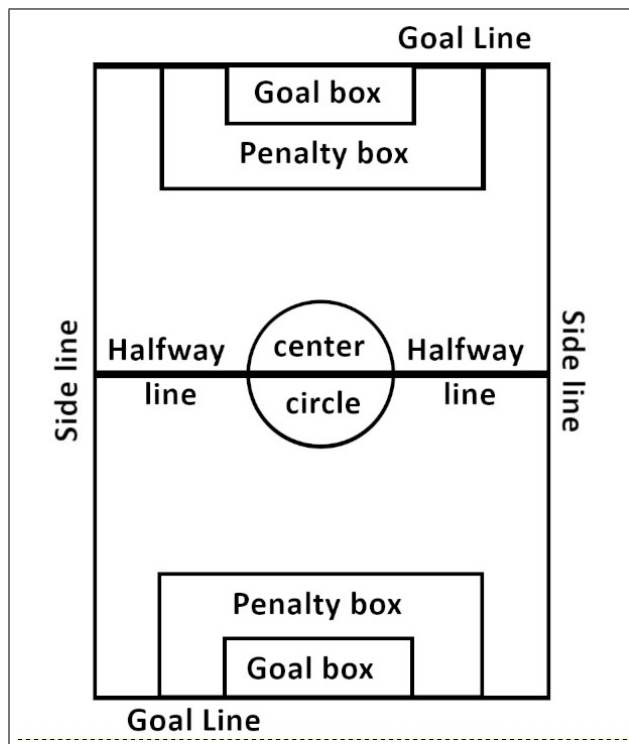
---

## Appendix A: Football Pitch Nomenclature

---

The following image in Figure A.1 displays an overview of the used nomenclature of the different lines and parts of a football field.

The final nomenclature used is "bounding lines", which is a term describing the outermost lines around the field (sidelines and goal line). Note that if only half of the pitch is visible, one of the goal lines are not visible and the halfway line is then used as a bounding line for that half of the pitch.



**Figure A.1:** Drawn schematic of a football field with used nomenclature indicated for sections and lines.

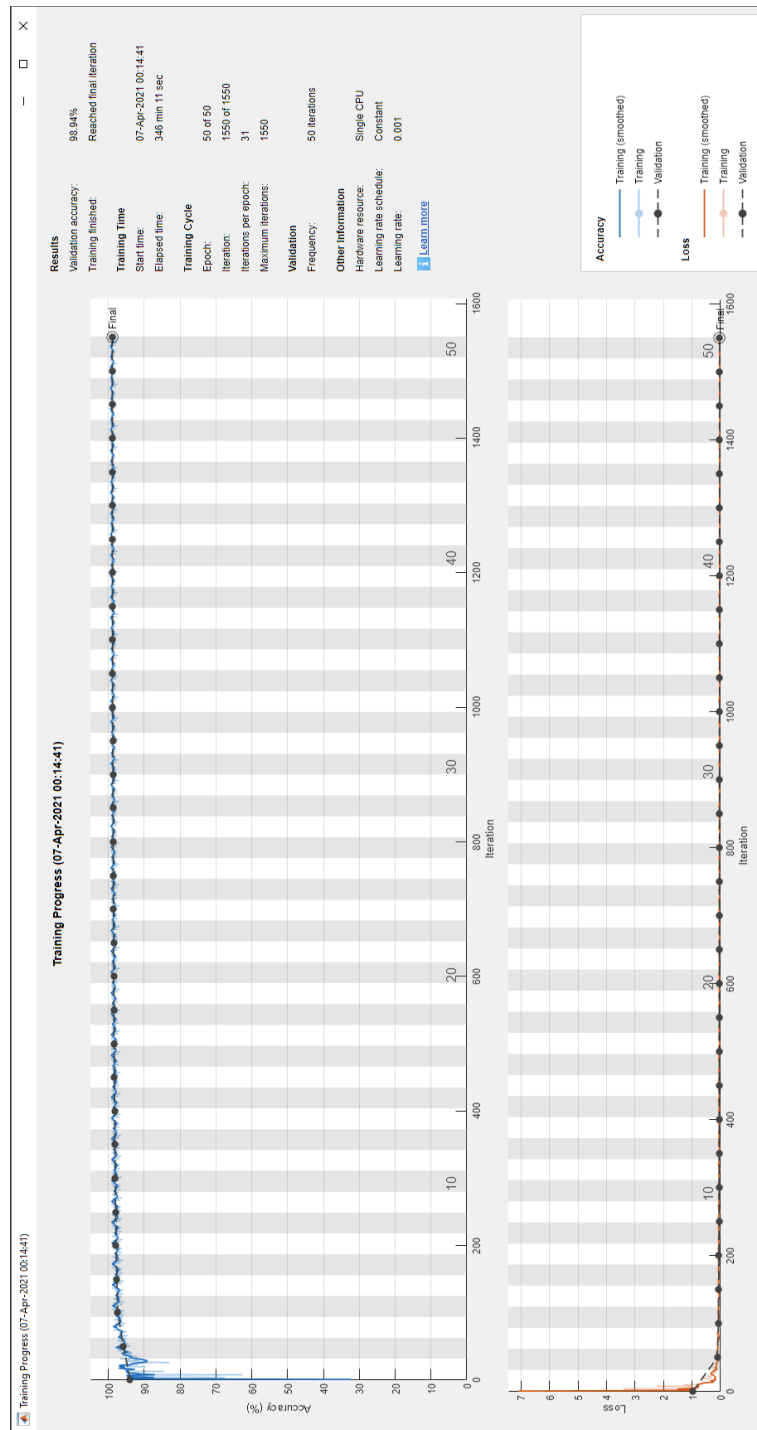




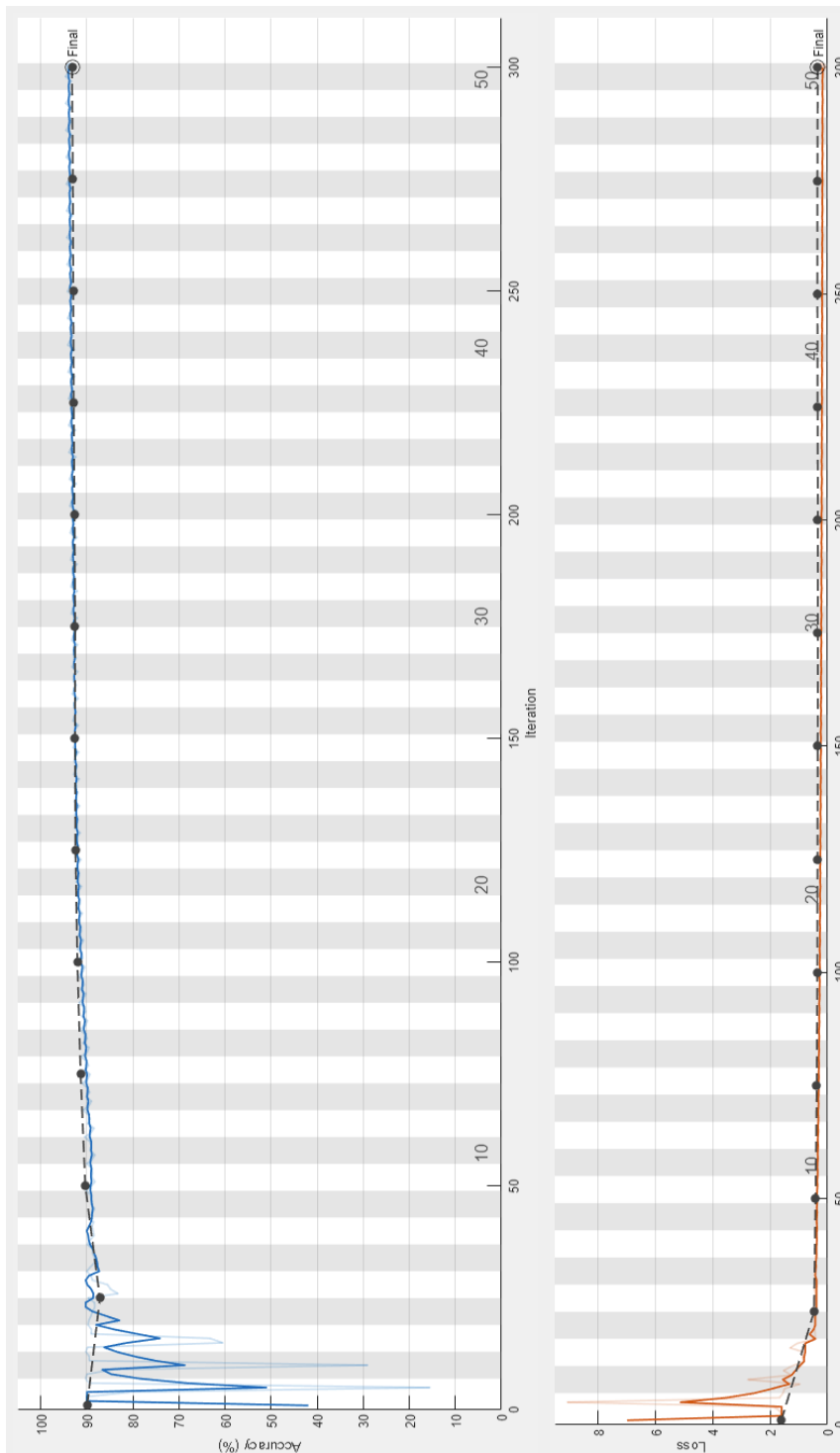
## Appendix B: ML-Training Process

---

Two typical examples of the training process is shown in Figures B.1 and B.2, displaying the progression of training and validation accuracy/loss. Note that the text is quite small from MatLab and is unfortunately not very readable in the printed version. However, since mostly the appearance of the curves were interesting for the thesis, one does not miss out on any vital information due to the smaller text size.



**Figure B.1:** Picture from the training process of one the DeepLab v3+ networks, the blue line indicates the accuracy and orange loss. Black points are for validations.



**Figure B.2:** Picture from the training process of one of the U-Net networks, the blue line indicates the accuracy and orange loss. Black points are for validations.



**LUND**  
UNIVERSITY

Series of Master's theses  
Department of Electrical and Information Technology  
LU/LTH-EIT 2021-824  
<http://www.eit.lth.se>