AARHUS
UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Lodging score assessment using image processing and machine learning.

By
Emil Hilligsøe Lauritsen
202004154
Martin Michaelsen
202007433

Bachelor's Thesis In
Computer Engineering

Supervisor: Anders Krogh Mortensen
Co-Supervisor: Mads Dyrmann

Abstract

English version

When grass is exposed to extreme weather phenomena it causes the grass to bend, snap and permanently lay down which results in a diminish in the yield of the grass production. This snap and bending of crops are known as lodging. A way to avoid this is by producing more lodging-prone grass varieties, where a part of the process is manually assessing the degree of grass lodging. This is assessment a laborious and subjective process, which is prone to biased results and unclear conclusion. The purpose of this bachelor thesis is to develop an objective image-based model to assess fine-scale lodging severity of grass in fields. This thesis explores the use of histogram of oriented gradients (HOG) and Haralick features to train a support vector machine classifier (SVMC) and a support vector machine regressor (SVMR). In this thesis we find that training a SVMC with hog features yields the best accuracy of 58%. Furthermore, we find the same model used in a practical setting using data from 2019 as the validation set to have an accuracy 43%. This result leaves much room to improve through future research and studies.

Danish version

Når græs udsættes for ekstreme vejrforhold, får det græsset til at bøje, knække og lægge sig permanent ned, hvilket resulterer i en reduktion i udbyttet af græsproduktionen. Denne knækkende og bøjende tilstand kaldes lejesæd. En måde at undgå dette på er ved at producere græssorter, der er mere tilbøjelige til at knække, hvor en del af processen er manuel vurdering af graden af lejesæd i græs. Denne vurdering er en besværlig og subjektiv proces, der er tilbøjelig til forudindtaget resultater og uklare konklusioner. Formålet med dette bachelorprojekt er at udvikle en objektiv billedbaseret model til vurdering af den fint skaleret lejesædalvorlighedsgrad for græs på marker. Dette projekt udforsker brugen af histogram af orienterede gradienter (HOG) og Haralick-funktioner til at træne en support vector machine classifier (SVMC) og en support vector machine regressor (SVMR). I dette projekt finder vi, at træning af en SVMC med HOG-features giver den bedste nøjagtighed på 58%. Desuden finder vi, at samme model brugt i en praktisk indstilling ved hjælp af data fra 2019 som valideringssæt har en nøjagtighed på 43%. Dette resultat efterlader meget plads til forbedring gennem fremtidig forskning og studier.

Work distribution

Process/Name	Emil	Martin
Image preprocessing	P	P
Feature extraction	P	S
Machine learning	S	P
Hyperparameter optimization	S	P
Practical validation	P	S

P indicates primary and S indicates secondary.

1. Introduction

1.1. Background

Grass is a highly used crop in temperate climates, where it is used as a major food source for livestock. Furthermore, is it used for seed production for use on private and public lawns. It is therefore crucial to be able to maximize the yield of the grass, as to minimize the waste of space and resources used during the grass production. The current constant change in climate and weather patterns results in more extreme weather such as harsher winds, more rain, or droughts. When grass is exposed to these extreme weather phenomena it causes the grass to bend, snap and permanently lay down which results in a diminish in the yield of the grass production. Thus, causing a fall in the economic yield for the farmers. This snap and bending of crops are known as lodging. In the case of grass, the longer the grass the higher the risk of lodging occurring. One way to prevent lodging from occurring is to develop crops that are more resistant. This can be achieved by e.g., increasing the strength of the grass straws such that it becomes harder for the grass to bend and break in the first place. During this process, the action of assessing the lodging severity is essential to determine which grass variety has an actual effect on the lodging of the grass. This assessment is currently done manually by the grass seed breeders and is a manual and tedious task. Furthermore, the assessment is subjective to the individual assessor, which can lead to biased results and unclear conclusions on the different lodging resistances in grass varieties. There is therefore a clear need for an objective and automated process to perform this assessment leaving more time and resources for the grass seed breeders to perform other tasks.

1.2. Related work

This thesis is not the first to design methods in which can assess lodging, several studies and papers has already been published both for grass lodging specifically but also for lodging of other crops. Some of these papers have investigated UAV-based models. *Tan et al*[1] studied the use of HOG (Histogram of oriented gradients) features and canopy height features to classify lodging. However, this project managed only a limited classification range, either no lodging, medium lodging, or severe lodging. *Li et al*[2] tries to develop an UAV-framework to detect lodging in sugarcanes. They use three types of features, mean band values, visible band vegetation index and Haralick textual features. Both Li and Tan used supervised learning in form of a support vector machine to classify lodging. *Wilke et al*[3] used an UAV-based canopy height model which they used to classify a lodging percentage in barley.

1.3. Objective and purpose

Even though previous studies achieve promising result they use a limited range of classification. (No lodging, medium lodging, severe lodging), this means that the model cannot properly substitute the manual assessment since makes it difficult to compare lodging resistance between different grass varieties. There is therefore a need for a model which achieves a more fine-scaled lodging assessment, which is what we will achieve in this thesis.

[1] (Tan, Mortensen, Ma, Boelt, & Gislum, 2021)
[2] (Li, 2020)
[3] (Wilke, 2019)

This bachelor thesis will develop an objective image-based model to assess fine-scale lodging severity of grass in fields. We will develop the algorithm based on image processing and machine learning. We are going to use histogram of oriented gradients (HOG) and a method proposed by Haralick to extract features. We will use these features to train a support vector machine classifier (SVMC) model to classify lodging. Since previous studies only tried to classify a limited range of lodging, we are going to try to achieve a wider range. This range is from 0-9 where each of the classes represent lodging score intervals of 10. We will also train a support vector machine regression (SVMR) model and compare it to the classification model. At last, we will optimize the hyperparameters for the models to obtain the optimal accuracy. The data for our supervised learning is supplied by the department of agriculture of Aarhus university. Our models will be evaluated against the manual score performed by the grass seed breeders. At last, we will perform an evaluation on a single year to mimic a practical use of our found model.

1.4. Git repository

The work performed in this thesis is developed and version controlled using the collaborative code sharing platform GitHub. The exact repository for this thesis can be accessed using the following hyper link: https://github.com/emil5654/Bachlor_project.git

2. Glossary

Abbreviation	
HOG	Histogram of oriented gradients
LS	Lodging Score
GLCM	Grey Level Co-Occurrence matrix
SVM	Support vector machine
SVMC	Support vector machine classification
SVMR	Support vector machine regression
UAV	Unmanned arial vehicle

Table 2.1: Glossary

3. Data

The data used to train and validate the model in this thesis are images of plots taken of two fields, which are located on Mindelundsvej and Bjaerub. This data has been supplied by The Department of Agroecology, Aarhus university, where they evaluate different grasses for seed production in field plot experiments. This data is gathered using an RGB camera mounted on an unmanned arial vehicle(UAV). Canopy height were also collected during this process but are not used in this thesis. Photogrammetry software was then applied to create orthophotos. Then each individual parcel was clipped from the orthophotos creating the individual plot. This was performed by a previous team, (Tan, Mortensen, Ma, Boelt, & Gislum, 2021), and we had no impact on the process. This part of the process can be seen on Figure 3.2 and the part of the process performed by mentioned team can be seen encased in a dashed square. These pictures of individual plots will henceforth be referred to as plot images, which contains both background and foreground pixels.

Each plot was scored manually with a Lodging Score (LS) which ranged between 0 and 100 in increments of 5. This data collection was performed for some fields weekly in May, June and July and has been performed in the span of the last 5 years which has accumulated to 4915 pictures of said

plots. As seen on Table 3.1 the process was not repeated for all fields consistently where Mindelundsvej were the only field pictured in 2016 and 2017. Further comments on the distribution can be seen in section 3.1. Note that the plots in Table 3.1 are not of the same plot during these years but shows the difference in our data both for the lodging but also for grain color, density and type of the crops.

The plot images contain a binary filter where each pixel will have a value of either 0 or 255. Whether it is 0 or 255 determines if the pixel is part of the plot or not. If they are part of the plot, meaning they have a filter value of 255, we call it a foreground pixel. If the pixel is not part of the plot, meaning it has a pixel value of 0, we call it a background pixel. As seen on Figure 3.1 the plot image is surrounded by white pixels. These are the background pixels. The green part of the image plot are the foreground pixels. We are only interested in the foreground pixels which will be accommodated in section 4.2.

As seen on Figure 3.1 the plots within the plot images are not aligned horizontally, which is caused by the cutting from the orthophotos which were aligned to cardinal directions. Thus, making the top of the plot images facing north and the bottom facing south. This has caused the plots to be skewed in two directions left and right. From this we can essentially divide the plot images into two cases. A case where the plot is skewed to the right and a case where it is skewed to the left. Henceforth will we call these cases respectively Case A and Case B, which are also marked in the figure.

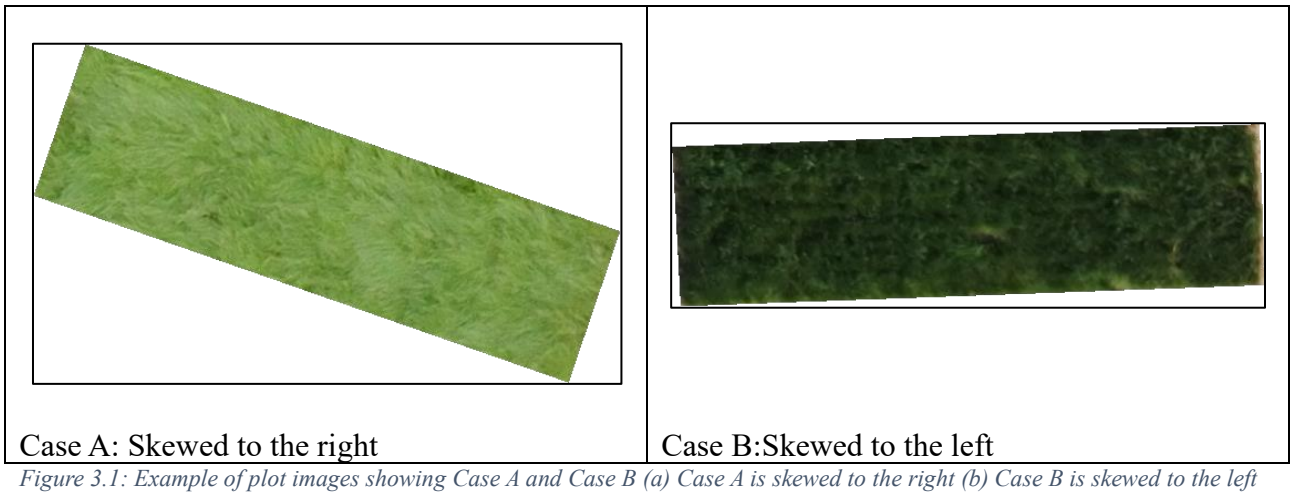

Figure 3.1: Example of plot images showing Case A and Case B (a) Case A is skewed to the right (b) Case B is skewed to the left

Year	Mindelundsvej	Bjaerub
May: 2018	 180529_Min_451	 180529_Bj_306
May: 2019	 190528_Min_94	 190514_Bj_98
May: 2020	 200528_Min_8	None this year
May: 2021	 210531_Min_91	None this year
June: 2017	 160607_Min_61	None this year
June: 2018	 180625_Min_449	 180608_Bj_306
June: 2019	 190628_Min_99	 190621_Bj_281
June: 2020	 200612_Min_97	 200612_Bj_580
June: 2021	 210629_Min_93	None this year
July: 2016	 170707_Min_96	None this year
July: 2018	None this year	 180705_Bj_305

Table 3.1: Examples of plot images for May, June and July for both fields

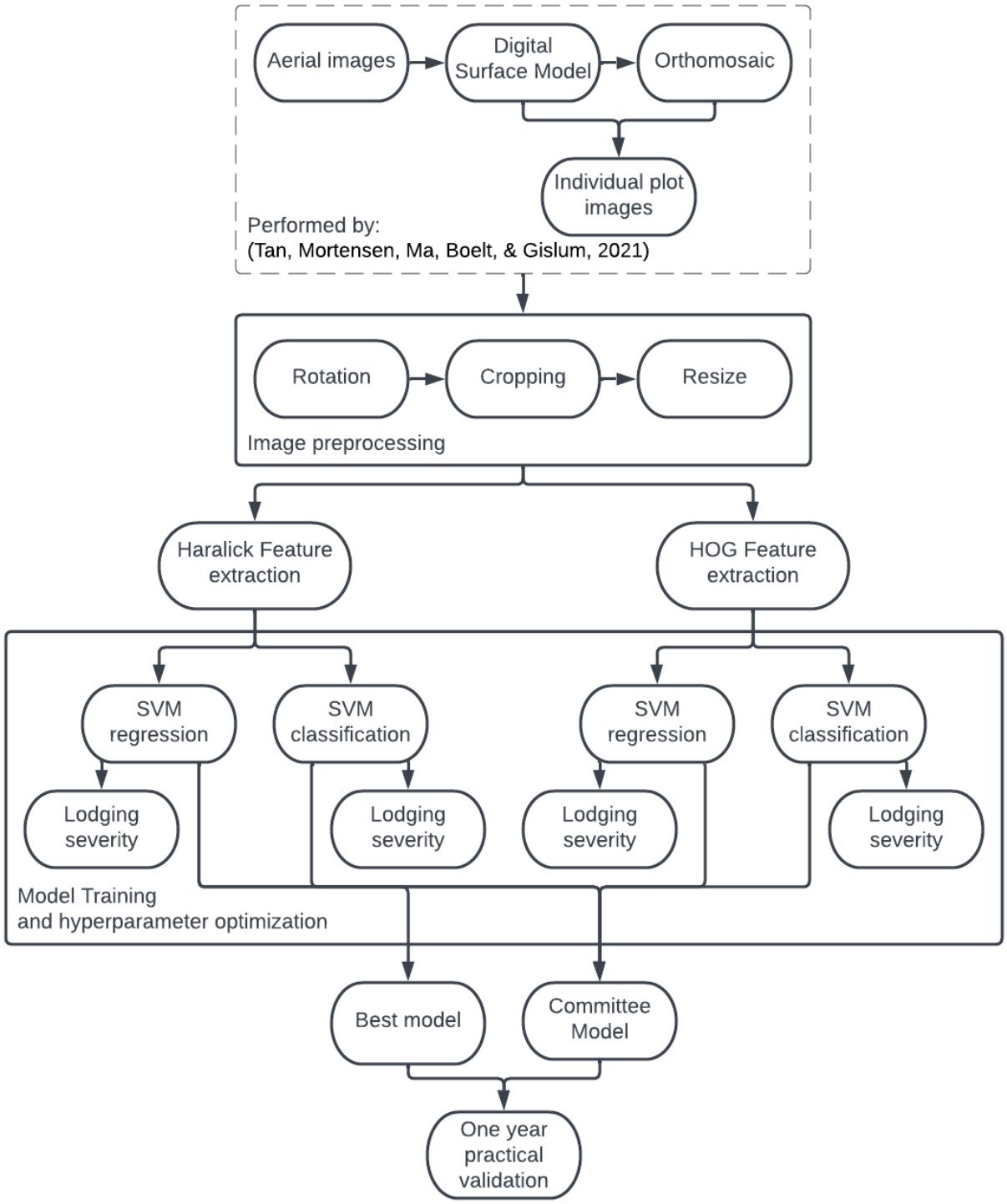

Figure 3.2: Flowchart of the process of this thesis for performing lodging severity extraction

3.1. Distribution of data

The following section will describe how the data is distributed. The distribution will be showed in terms of date and of how many instances of each LS there is.

In Figure 3.3 we see a boxplot of the LS for each month. Here we see an increase in lodging from May to June. We see that the average lodging is low for May and high for June and July. As previously mentioned, when the grass continues to grow the risk of lodging increases as well. However, it is

worth noting a small decrease in the LS from June till July. This does not correspond with what we would expect. We would instead expect an equal or increase from June to July. It is also worth noticing that there are 11 outliers in May and that the distribution of scores of June range all the way from 0 to 90. In appendix 9.1.2 we can see the standard deviation for each lodging month. They are as expected high with June reaching 27. This tells us something about the subjectiveness of our data. This will be further discussed in section 6.2.

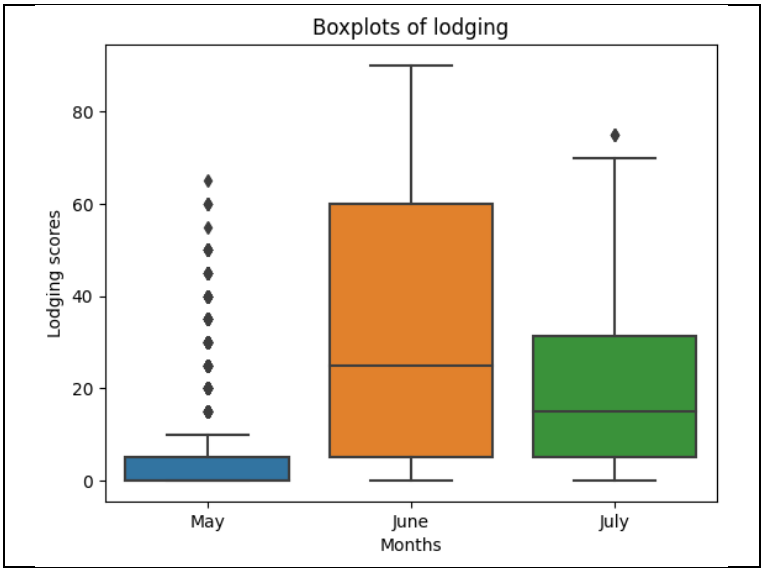

Figure 3.3: Average LS of each month.

In Figure 3.4(a) we have illustrated how the data we have been given is distributed, which were given in increments of 5. In bin 0, there will be visual scores from 0-5, in bin 1 scores from 6-10 and so forth. It is worth noticing that 38% of our data consist of 0-5 visual scores. A mentioned in section 1 we have chosen the fine-scaled model should consist of classes 0-9 where each class contains 10% lodging. This distribution can be seen on Figure 3.4(b). Here we see that around 45% of the data now lies within class 0.

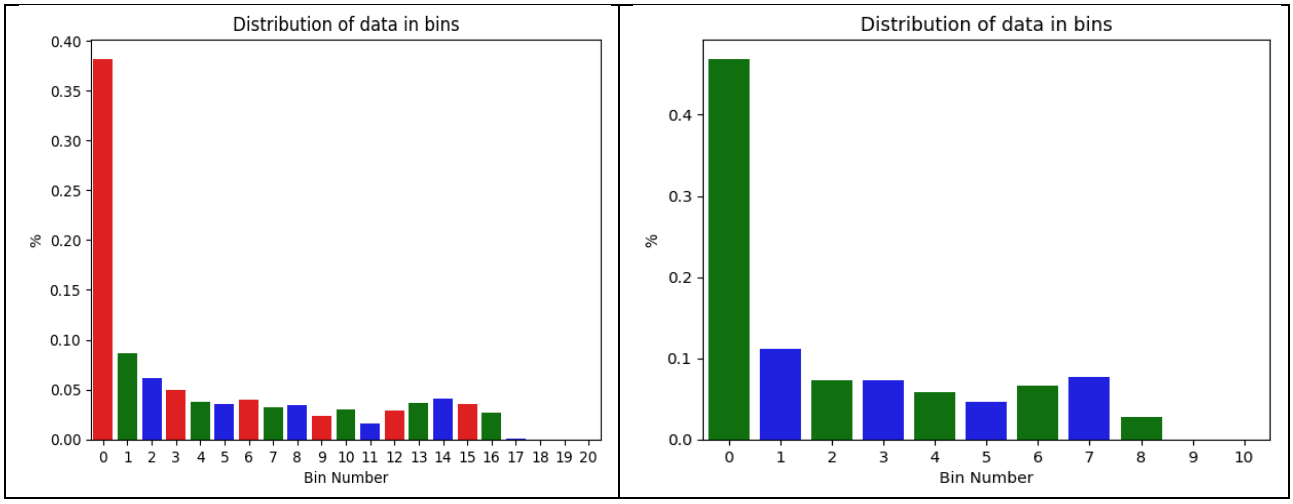

Figure 3.4: Bar plot of data distribution. (a) is for bin size 5 (b) is for bin size 10.

3.2. Data augmentation.

Data augmentation is a technique which can be used in machine learning. It works by creating new variation of the previous data thus increasing the number of training data. By increasing the amount of training data, we increase the model's potential to capture general pattern in the data. By using data augmentation, we hope to increase the models' overall performance, and achieve some variation in the classes it picks. A potential drawback to this is that bias in the original data will also be present in the new augmented data.

For our thesis we chose to do a geometric transformation of our data by mirroring the data horizontally and vertically. These mirrors can be seen on Figure 3.5. We can see the patch of lodging on the not mirrored version is mirrored on the four pictures. The effect of this will be that our model will have four versions of the same picture with the same LS, though in different parts of the picture, to use for its training. Thus, increasing the number of plot images four-fold.

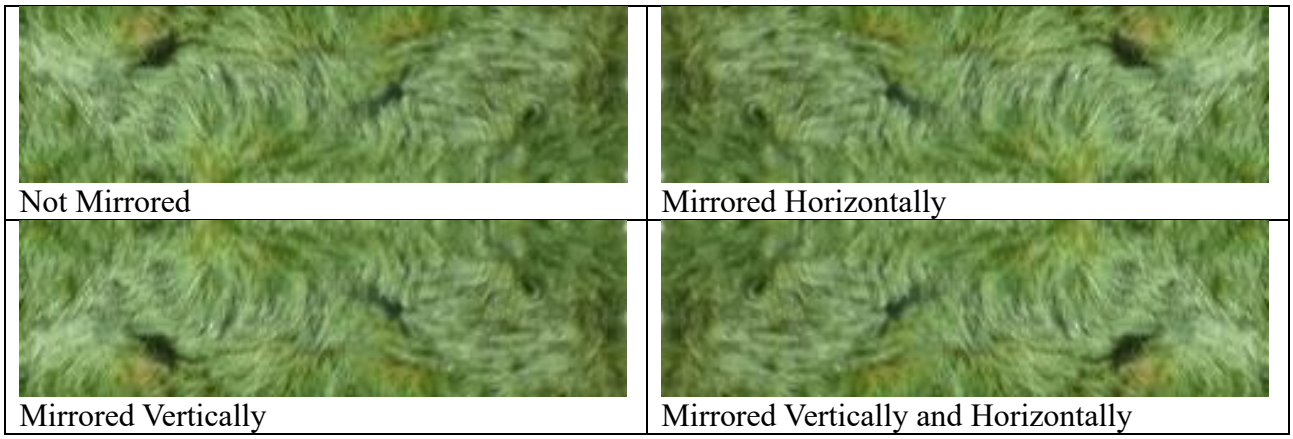

Figure 3.5: Example of plot image mirrored.

3.3. Data balancing

As seen on Figure 3.4(b) the data is heavily concentrated at class 0. This means that our supervised learning model will have much more data available to learn the characteristics of low lodging plots. This also means that it will be more likely to predict low LS. We chose to examine the effect of changing the data distribution in three ways. Two for the mirrored plot images and one for the non-mirrored plot images. For the non-mirrored we chose to select 500 random images with LS of 0 to use in our dataset, which will create a more uniformly distributed dataset. For the first method for the mirrored plot images, we pick 500 random plot images with a LS of 0 to mirror and ignore all the rest when training the model. The second were to choose 2000 random plot images from the mirrored plot images with a LS of 0 to use in our dataset. In both cases the data will be more uniformly distributed, but the first will be trained against even distributed plot images rotated in each direction where the second has a random distribution in the mirrored orientations. The effect and results of this can be seen in section 5.4.

4. Method

4.1. Python Packages in use

In this section we will list and reference the used python libraries used to perform the methods described in section 4. Some of them will be described in following sections and some of them are for basic functionality and will therefore not be mentioned further.

Package	Version	Link to website
Python	3.9.12	https://www.python.org/
Scikit-learn	1.0.2	http://scikit-learn.org
Scikit-image	0.19.2	https://scikit-image.org
mahotas	1.4.13	http://luispedro.org/software/mahotas
imutils	0.5.4	https://github.com/jrosebr1/imutils
cv2	4.6.0	https://pypi.org/project/opencv-python/
numpy	1.21.5	https://www.numpy.org
pandas	1.4.2	https://pandas.pydata.org
seaborn	0.11.2	https://seaborn.pydata.org
matplotlib	3.5.1	https://matplotlib.org

Table 4.1: Python packages in use. Conatining links to home-site and version numbers.

4.2. Image Preprocessing

Before we can do any kind of training and prediction for our data, we must process the pictures to extract some kind of features which the model can use. We will describe how these features are extracted in section 4.3. But first, as seen on Figure 3.2, we preprocess the pictures to make them uniform, such that the pixel density and image orientation has as small an effect on the result as possible. For the preprocessing, the intention is to make the pictures the same orientation and size. As we can see on the example pictures on Figure 3.1 the pictures are skewed. The first step was to find a universal method to rotate these pictures to make them straight horizontally. The second step is then to crop the images such that they only contain foreground pixels. The last step is then to resize the plots to the same size. These steps can also be seen on Figure 3.2.

4.2.1. Rotate pictures:

To rotate the pictures we used the rotate function from the imutils package which rotates the picture with a given angle. The angle can be negative to symbolize a counterclockwise rotation and positive to symbolize a clockwise rotation. As seen in mentioned in section 3 our data can be skewed in two ways, therefore we need to account for this difference when finding the rotation angle. One method for Case A and one method for Case B.

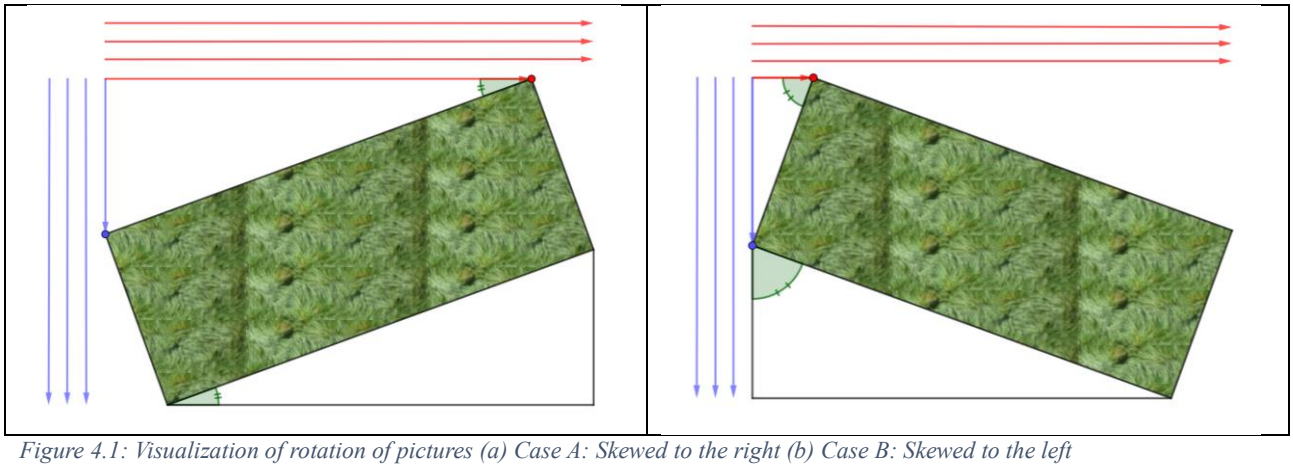

Figure 4.1: Visualization of rotation of pictures (a) Case A: Skewed to the right (b) Case B: Skewed to the left

Rotate pictures: Case A

The way we found the angle for Case A, was by first finding the two top corners of the picture, which can be seen on Figure 4.1: Case A as a red and blue dot. The corners are found using two function *"find_left_corner"* and *"find_right_corner"*

In Case A *"find_left_corner"* locates the column and row of the top left corner pixel shown in Figure 4.1 as a blue dot, and *"find_right_corner"* locates the column and row of the top right corner pixel shown as a red dot. As mentioned in section 3, the plot images have a binary filter showing which pixels are part of the background with a value of 0 and which pixels are part of foreground with a value of 255. The way the function finds the blue corner is by traversing all rows in each column until it finds the first pixel which is a foreground pixel and saves the row and column number of this pixel. This traversal is shown in the Figure 4.1 as blue arrows. In effect this goes from top to bottom until it finds the first foreground pixel. The function finds the red corner by traversing all columns in one row until it finds the first foreground pixel and saves the row and column number of this pixel. This traversal is shown by the red arrows. In effect this goes from left to right until it finds the first foreground pixel.

The angle is then found by finding the length in pixels of the blue and red arrow shown on the figure connecting the blue and red dot. Then through basic trigonometry the angle is found as follows:

$$\angle = \arctan\frac{a}{c}$$

Where a is the length of the blue arrow and c is the length of the red arrow. The angle can also be seen in Figure 4.1(a). Note that the two angles shown are identical.

Rotate pictures: Case B

The way we found the angle for Case B, was by finding the two left corners of the picture, which can be seen on Figure 4.1(b) as a red and blue dot. The corners are found using the same functions as for Case A. In Case B *"find_left_corner"* locates the column and row of the top left corner pixel shown in Figure 4.1(b) as a blue dot, and *"find_right_corner"* locates the column and row of the bottom right corner pixel shown as a red dot. The traversal is the same for the two functions for both Case A and Case B. The difference is the corners which are found by the traversal through the pixels. Where the red dot is the top right corner for Case A and bottom left corner for Case B.

The angle for Case B is found the same way as the angle for Case A. The difference for Case B is that the found angle will rotate the plot image to stand vertically, but this is then corrected in the code by subtracting an additional 90°. The actual rotation is done using the same code as that for Case A, with the difference of subtracting the 90°. The code decides whether the plot image is of Case A or B by checking whether the column of the blue corner is below or above 200 making it respectively either Case B or Case A. Note 200 is a value chosen arbitrarily as a boundary to decide whether the point is at the top or at the bottom of the plot image. This value might need adjustment if the pictures are initially skewed at higher angles.

Filter adjustment

Now that the pictures have been rotated there are still background pixels surrounding the plot. During this rotate, some pixels did not fit to an exact pixel after the rotate. This is dealt with by taking the mean value from surrounding pixels. This did not have a serious effect on the pixels themselves, but

the binary filter became non-binary by doing this, which is why we made a function to fix the filter by changing all pixels, having a filter value below 255, to 0. This might lead to some loss of pixels but at such a low level that it should not have any major effect on the final result. The background pixels are though still there which brings us to next stage which is the cropping of the pictures.

4.2.2. Crop pictures:

The objective at this phase is to find the biggest image we can create while only containing foreground pixels. This image can be seen on Figure 4.2(a) as the red rectangle. This is done by finding the pixel closest to the center of the picture for respectively the top, the bottom, the left and the right side of the picture. Considering the case of finding the top pixel, we iterate through the rows, going from the top until we reach a pixel having a filter value of 255. This is repeated for each column and at last the highest row number is returned. This process is represented on Figure 4.2(b) as the blue arrows. Here one problem is that the pictures are not perfect rectangles since some sides may be sloped and crooked. This can also be seen on Figure 4.2. Note that the example is an extreme example of a plot image since the sides on the data were often relatively straight. This was fixed by introducing an ignore value n which ignores the n^{th} right and left most pixels while finding these inner pixels which could be used to construct the red rectangle. This ignore value can also be seen on Figure 4.2. For this ignore value we found that 20 will allow us to get through all the pictures. This is then repeated for all sides, where for the bottom we iterate through the rows while starting from the bottom, and for the left and right side we iterate through the columns and respectively start from the left and the right.

These pixel values are then used for the cropping of the picture, which is done by overwriting the picture where the ranges for the rows and columns are the found values for the red rectangle. We can see this in pseudo code shown in formular (4.1).

$$image_i = image_i[top{:}\,bottom, left{:}\,right] \tag{4.1}$$

Where top and bottom are the rows of the closest foreground pixel to the center for respectively the top and bottom. Left and right are then the columns of the closest foreground pixel to the center for respectively the right and left.

One problem which arose from this process where if the closest pixel to the center were in the ignore values range, then some background pixels where still in the corners of the picture. This where though easily fixed by running the code again using an ignore value of 0. Whether or not this was needed where found by checking if there still existed a 0 in the filter for the cropped picture. This change resulted in all pictures being cropped successfully.

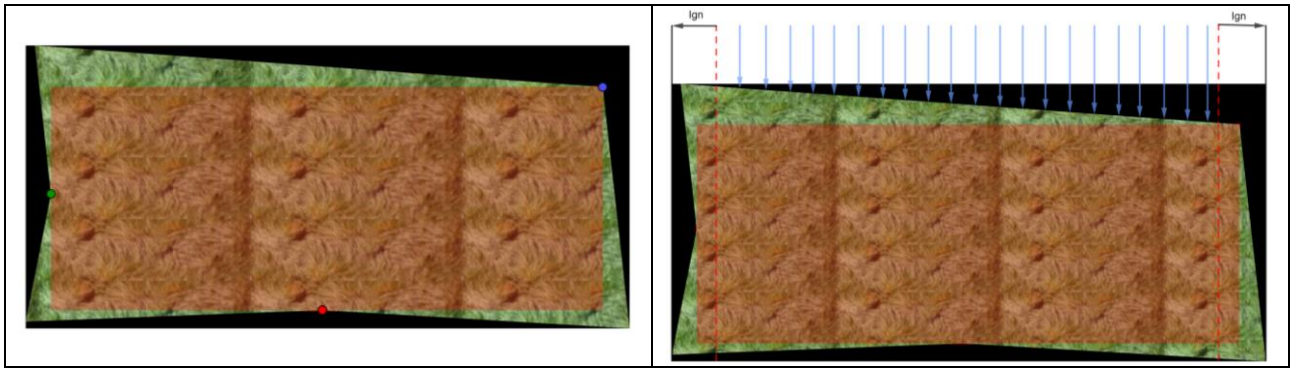

Figure 4.2: (a) Crude example of picture before cropping (Green is the individual plot image. Red is the cropped image area. Black is the background pixels.). (b) Process of finding the top pixel used for the cropping of the plot image. (Blue arrows indicate the process of the function.)

4.2.3. Reshape pictures:

After the cropping, the step before achieving uniform pictures is to take the different pixel densities into consideration. This is, as mentioned earlier, caused by the different flight heights when taking the drone pictures. For the reshaping we chose to use the reshape function in the cv2 package. This takes the desired image dimension and resizes the image to these dimensions.

We wanted to check whether the size had an impact on the accuracy of our model. Therefore, we chose three different sizes for the picture. The first size is the mean size of all the images, the second is the size of the smallest picture and the third is the size of the largest picture. The sizes and number of pixels can be seen in Table 4.2.

	Min	Mean	Max
Img size	71x343 pixels	144x496 pixels	169x534 pixels
Number of pixels	24353	71424	90246

Table 4.2: Picture sizes

4.3. Features extraction.

As previously mentioned, we need to extract features from the images that our supervised learning model can learn from. Here we have selected two different methods of features extraction. Histogram of oriented gradients which is described in section 4.3.1 and Haralick extracted from an GLCM matrix which is described in section 4.3.2. We refer the reader to review Figure 3.2 for an overview of where we are in the process.

4.3.1. Histogram of oriented gradients.

Histogram of oriented gradients (HOG) is a method that can be applied to an image to extract features related to texture and shape. It was developed by *Dalal and Triggs*[4] as a means of object detection. They used it on the MIT pedestrian set with which achieved near perfect results.

The steps.

There are some steps that needs to be taken when extracting HOG features. A detection window needs to be specified in which the features are going to be extracted from. This detection window is going to be divided into cells. Here, a number of pixels per cell needs to be decided. In each cell the gradient in the x- and y-direction is calculated. Thereafter the magnitude and angle of the gradients are calculated. These will be put into a histogram sorted by angle. The next step is to divide the detection window and cells into blocks. Here, a number of cells per block needs to be decided. These blocks are going to be placed with a 50% overlap. For each of these blocks we collect the histogram of each cell into a vector The final step is then to perform contrast normalization on each vector. Each of these steps are described in more details in the following section.

The detection windows.

Dalal and Triggs suggest that one need to create a detection window in which the object one wants to detect is present. In our case our detection window would be the plot, since we want to detect

[4] (Dalal, 2005)

lodging on the whole plot. In section 4.2.2 we describe how the pictures are cropped into three different seizes (Min, Mean, Max). These are displayed in Table 4.2. This size will influence the number of features we extract. Therefore, we will try these three different image sizes.

Cells

The image that we want to find the HOG features of is divided into spatial region called cells. The choice of pixels per cell has a big impact in terms of the number of computations and memory needed to extract the features. This is caused by a smaller number of pixels per cells entails a larger number of calculations since it can fit more cells within the image. We are going to test four different combinations of cells and blocks and find the combination which gives us to most optimal model. These are displayed in Table 4.3.

4x4, 2x2	16x16, 2x2
8x8, 2x2	8x8, 4x4

Table 4.3: Cell and block sizes

An important factor to consider when choosing a cell size is the computational cost. The number of computations happens cell and block-wise which means that the lower a cell size that we chose the more cells and block we can fit into an image. Likewise, this also means that the larger an image size that we chose the more cells and blocks we can fit. In both cases increasing the computational cost of both the feature extraction and the model training. Another thing to keep in mind is that a lower cell and block size also means that we can extract more descriptors. More descriptors might help our model capture the patterns of our data better. It is therefore important to find the right balance.

Calculating gradients.

The gradient of an image is defined as a two-dimensional vector which can be seen in formular (4.2)[5].

$$\nabla(x,y) = \text{grad}(x,y) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{df}{dx} \\ \frac{df}{dy} \end{bmatrix} \quad (4.2)$$

The gradient vector points in the direction in which the rate of change is the highest. We calculate magnitude and angle of each gradient in the cell with formular (4.3)[6]:

$$Magnitude(u) = \sqrt{G_x^2 + G_y^2}, Angle(\theta) = |\tan^{-1}\left(\frac{G_y}{G_x}\right)| \quad (4.3)$$

Notice that arctan return values between -90 and 90 so we add 180 to all negative values in order to place them in our histogram.

[5] (Woods, 2018), p 184
[6] (Woods, 2018), p 185

Gradient orientation histogram.

These magnitudes and angles/orientation are then put into a 9-bin histogram. The histogram is sorted by orientations. This reduces our gradient vectors to just 9 values. A 9-bin histogram means that each bin covers 20 °. It is important to notice that in each of the bins there is the summed magnitude of the gradients with a certain angle. In the original paper they also experiment with the number of orientations in the histogram. Though they found that 9-bins typically achieves the highest performance. We therefore chose to use 9-bins as well.

Blocks.

The next step is now placing our blocks in our detection image. Each block is going to consist of a number of cells. The combination of cells and blocks that we are going to investigate can be seen in Table 4.3. We place each of these blocks with a 50% overlap of each other.

Contrast normalization.

We then perform contrast normalization on each block. Recall that each cell now consists of a histogram vector. We then collect all the histogram vectors for each block. This is done by concatenating each histogram vector in X_{con}.

$$X_{con} = X_0 <> X_1 <> X_2 <> X_{n-1}$$

Where $X_0, X_1, \dots X_n$ is histogram vectors. n is number of cells per blocks. Each of these consist of

$$X_n = x_0, x_1, x_2, \dots x_{i-1}$$

Where x_i is a bin in the histogram and i is the number of bins in our histogram. Note that the length of X_{con} is given by.

$$len(X_{con}) = m = i \cdot n$$

We now have one X_{con} per block which we perform L2 normalization on. This normalization introduces better invariance to illumination, shadowing, and edge contrast. *Dalal and Triggs* experiment with different types of normalization, however they find that L2 achieves the highest performance. Therefor we chose this as our normalization.

$$X_{norm} = \sqrt{x_0^2 + x_1^2 + x_2^2 \dots x_m^2}$$

At last, we concatenate each of these normalized vectors into one final feature vector. Since the normalization happens block wise it makes each cell appear multiple times in the final output vector. *Dalal and Triggs* found that this significantly improved performance.

In Figure 4.3(a) an example of an image plot and its belonging HOG feature visualization Figure 4.3(b) can be seen. Each of the white dots is in fact a histogram. We rescaled the intensity to better visualize it. This feature vector can then be used for classification in object detection. In our thesis we have used the Scikit-image library to implement this.

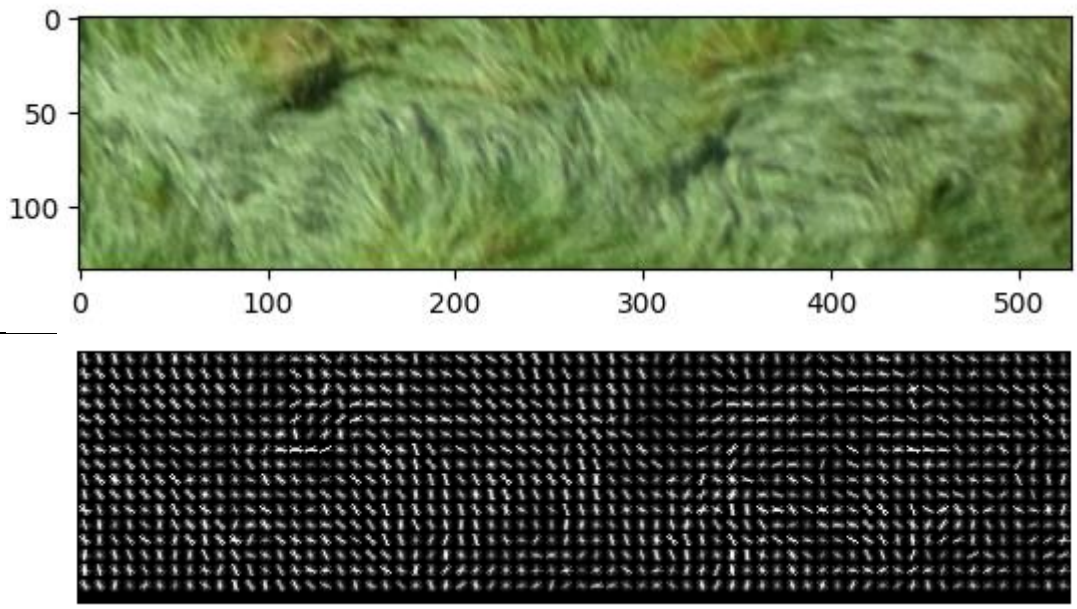

Figure 4.3: (a) Example of plot image (b) the belonging representation of HOG features. 20160607_mm_ebee_rgb_transparent_mosaic_group1_ROI_61.tif

4.3.2. Haralick textual features.

Haralick texture features is a way that you can describe the texture of an image. It can be used to distinguish between contrast in pictures. Haralick himself describes the Feature descriptor in his paper *"Textural features for image Classification"*[7]. As the title suggest, the features are well suited for image classification. Haralick textual features are calculated from the Grey Level Co-Occurrence matrix (GLCM). The GLCM describes how many times two pixels with specific values appear adjacent to each other in an image. This is visualized on Figure 4.4(a).

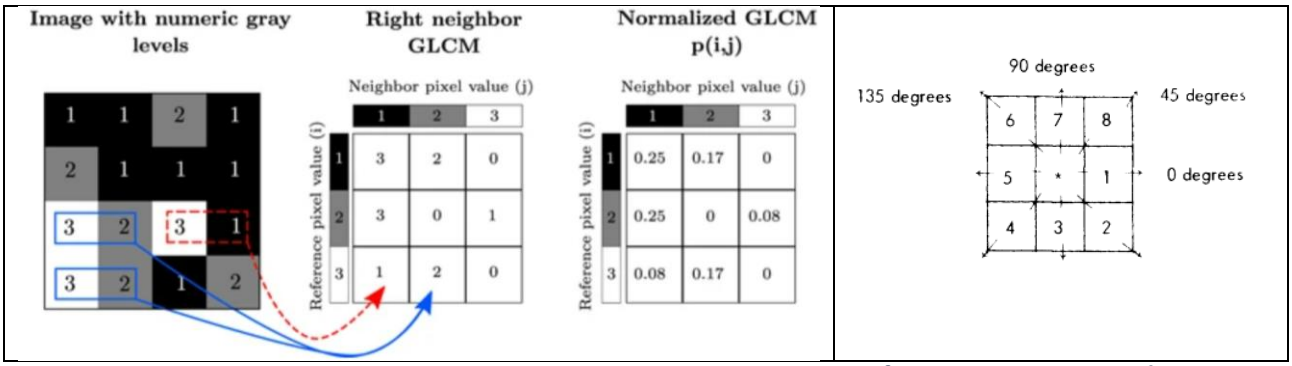

Figure 4.4: (a) GLCM shown. Left: Numeric, Middle: Right Neighbor, Right: Normalized.[8] (b) GLCM matrix degrees[9]

In Figure 4.4(a) we see that the gray-level pixel values of 3 and 2 are present adjacent to each other two times on the image on the left. This is then represented in the GLCM in the middle as 2. This means that each element (i, j) in our resultant GLCM is simply the number of times each (i,j) is present next to each other in each direction. There can be found a GLCM in each direction (0, 45, 90

[7] (ROBERT M. HARALICK, 1973)
[8] (Löfstedt, 2019)
[9] (Löfstedt, 2019)

and 135 degrees). The GLCM is then calculated in every direction, giving us 4 distinct GLCM. The degrees are visualized in Figure 4.4(b). Haralick suggests that when calculating the GLCM is to take the average of the 4 distinct normalized matrixes and calculate the textual features from this matrix. Which can be seen in formular (4.4).

$$GLCM_{mean} = \frac{GLCM_0 + GLCM_{45} + GLCM_{90} + GLCM_{135}}{4} \quad (4.4)$$

This leads to several properties which the GLCM possess, and which are used to calculate the Haralick textual features. In the following formulars N_g is the number of gray-levels and R is the sum of all values in $GLCM_{mean}$. (In Figure 4.4(a) we see a GLCM with $N_g = 3$.)

$$R = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} GLCM_{mean}(i,j)$$

$$p(i,j) = \frac{GLCM_{mean}(i,j)}{R}$$

This makes $p(i,j)$ a normalized GLCM.

The following two formulars are the sum of all pixel values respectively in the x- and y-direction.

$$p_x(i) = \sum_{j=1}^{N_g} p(i,j)$$

$$p_y(j) = \sum_{i=1}^{N_g} p(i,j)$$

The following four formular are statistical features which are used to calculate the Haralick textual features, which can be seen listed in Table 4.4.

$$p_{x+y}(k) = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p(i,j), \qquad \text{for } i+j = k \text{ and } k = 2,3,\dots 2N_g$$

$$p_{x-y}(k) = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p(i,j), \qquad \text{for } |i-j| = k \text{ and } k = 0,1,\dots N_g - 1$$

$$HXY1 = -\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p(i,j)\log\left(p_x(i)p_y(j)\right)$$

$$HXY2 = -\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p_x(i)p_y(j)\log\left(p_x(i)p_y(j)\right)$$

Angular Second Moment (energy) $$f_1 = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p(i,j)^2$$ Will return values between 0 and 1. Higher values indicates higher uniformity of the texture in the image	Contrast $$f_2 = \sum_{k=0}^{N_g-1} k^2 p_{x-y}(k)$$ Contrast in images refers to the difference in brightness between different pixels in the image. Higher values of contrast mean brighter areas of the images.
Correlation $$f_3 = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_g}(ij)p(i,j) - \mu_x\mu_y}{\sigma_x\sigma_y}$$ Where μ_x, μ_y, σ_x and σ_y are the means and standard deviation of p_x and p_y	Sum of square: variance $$f_4 = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g}(i-\mu)^2 p(i,j)$$
Inverse difference Moment: $$f_5 = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g}\frac{1}{1+(i-j)^2}p(i,j)$$	Sum average $$f_6 = \sum_{i=2}^{2N_g} i p_{x+y}(i)$$
Sum variance: $$f_7 = \sum_{i=2}^{2N_g}(i-f_8)^2 p_{x+y}(i)$$	Sum entropy: $$f_8 = -\sum_{i=2}^{2N_g} p_{x+y}(i)\log\left(p_{x+y}(i)\right)$$
Entropy $$f_9 = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p(i,j)\log(p(i,j))$$	Difference Variance: $$f_{10} = var\ of\ p_{x-y}$$
Difference Entropy: $$f_{11} = -\sum_{i=0}^{N_g-1} p_{x-y}(i)\log\left(p_{x-y}(i)\right)$$	Information measure of correlation: $$f_{12} = \frac{f_9 - HXY1}{\max(HX, HY)}$$ Where HX and HY are entropies of p_x and p_y
Information measure of correlation: $$f_{13} = \sqrt{1 - e^{-2(HXY2 - f_9)}}$$	

Table 4.4: Haralick textual features

The features f_8, f_9 and f_{11} are features of entropy. Entropy is a measure of the amount of randomness in the distribution of pixel values. f_9 takes the sum of the elements in GLCM. Since GLCM is a measurement of how many of the same pixel values are present next to each other, it makes sense that higher values of entropy mean a more uniform distribution of pixels. The logarithmic part of f_9 the

calculation converts the probability of occurrence of a pixel value into bits. It tells us how many bits that are required to represent each pixel value in binary form. We are using a python package called mahotas to calculate the GLCM and extract the Haralick textual features.

4.4. Machine learning

We refer the reader to review where we are in the process by recalling Figure 3.2. We have made it to model training and optimization of hyper parameters. As mentioned in section 3.1 we have split the data up in classes going from 0-9. This entails that our problem becomes a multiclass problem with 10 classes, meaning we need a method which can be used for a multiclass problem. Therefore, for our machine learning model we have chosen to use a Support Vector Machine Classifier (SVMC), which has performed well for problems concerning object detection. We also wanted to look at the problem in another way. We could see the meaning behind looking at the problem as regression problem since it would allow us to create a model which would give us a lodging percentage, which we then could compare to our classification model. This is why we also have chosen to design a Support Vector Machine Regression (SVMR) model.

4.4.1. SVM classifier

Before we describe the multiclass SVM it would be simpler to start describing the binary classification the occurs when applying SVM. The training data will include N inputs, $x_1, \dots, X_N$ and N corresponding target values $t_1, \dots, t_N$ where $t_N \in \{1, -1\}$. The predictions will then be made on the linear model shown in formular (4.5).

$$y(x) = w^T \phi(x) + b \tag{4.5}$$

Where y denotes the prediction, w denotes the weights, $\phi(x_n)$ denotes the transformed input space vector and b denotes a bias parameter.

The purpose of SVM is to find a hyperplane in a (N-1)-dimensional space which separates our classes. This hyperplane should then have the maximum distance between the hyperplane and the data points, where the closest data points to the hyperplane are called support vectors. In Figure 4.5 we see the support vectors as the blue and the green dots shown on the dashed lines called the margin. The margin is defined by the smallest distance between the hyperplane and any of the data points. The support vectors will, because of their close position influence the exact position of the hyper plane more than other more distant points. In SVM the hyperplane is chosen such that the margin is maximized, thus ensuring the maximum distance between our hyperplane and the nearest data point and therefore the maximum distance to all points. SVM's are also known as kernel SVMs or kernelized SVMs.

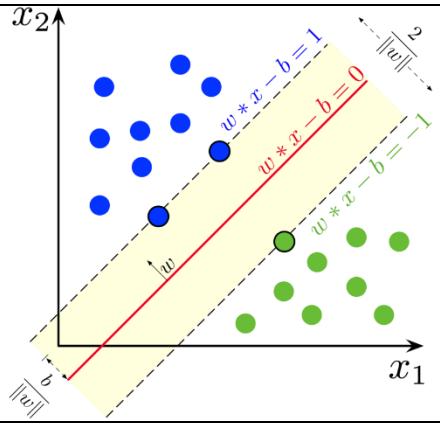

Figure 4.5: Representation of SVM principles.
(https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:SVM_margin.png)

Kernel

The reason it is also known as kernelized SVM is the use of kernels in the training of the model and later in the predictions using the model. The function of a kernel is to take the input data and represent it in a higher dimension without transforming the data to that dimension, which saves on processing power and time. This is known as the kernel trick.

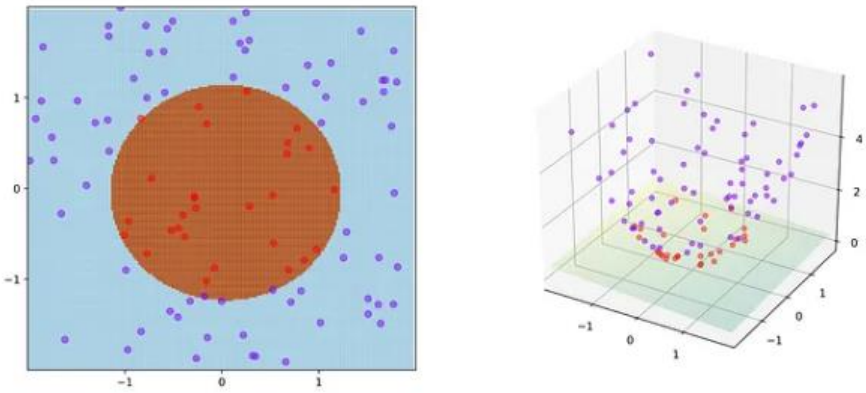

Figure 4.6: Representation of the Kernel Trick.
(https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:Kernel_trick_idea.svg)

By looking at Figure 4.6 we see data being transformed from 2d to 3d. From this we can see that classes which were separable using a circle now can be separated using a hyperplane, since the red class is located lower than the purple class when transformed to 3d. Therefore, if we were using a 3d kernel for this we would not have to do the transformation of the data to find this hyperplane. When talking specific kernels there are infinite possibilities, but some of the most popular when talking machine learning are the following:

1. Linear Function	$k(x_n, x_m) = x_n \cdot x_m$
2. Polynomial Function	$k(x_n, x_m) = (\gamma \cdot (x_n \cdot x_m) + r)^d$
3. Radial Basis Function (RBF)	$k(x_n, x_m) = \exp(\gamma \cdot \|x_n - x_m\|^2)$
4. Sigmoid Function	$k(x_n, x_m) = \tanh(\gamma \cdot (x_n \cdot x_m) + r)$

Table 4.5: Kernels in machine learning

The implementations of these kernels are part of the scikit-learn package, and the optimal kernel will be chosen later. This optimization process is described in section 4.5.

SVM mathematical concept

The mathematical explanation behind SVM to be presented are based on the derivations found in *Pattern Recognition and Machine Learning section 7.1*[10].

As stated earlier the SVM method is to maximize the margin, between the decision boundary and the closest data point. We start by listing the problem with the assumption that all points are linearly separable. This can be achieved by solving the maximum margin problem shown in formular (4.6).

$$\arg\min_{w,b}\left\{\frac{1}{2}||w||^2\right\} \tag{4.6}$$

$$y_i(w^T\phi(x_i)+b)\geq 1, \qquad \mathrm{n}=1,\dots,\mathrm{N}$$

Where w denotes the weights, $\phi(x_n)$ denotes the transformed input space vector, b denotes a bias parameter and t_n denotes the target value for the n^{th} training data point. The exact justification for deriving this as the maximum margin problem can be seen in *Pattern Recognition and Machine Learning.* In the same section a solution to this maximum margin problem is also listed but skipped in this thesis since we are more interested in the case where it is not necessarily linearly separable. This is because it is often the case that some of the classes overlap. To accommodate this, we add the possibility that some points might be misclassified. This is done by introducing slack variables $\xi_n \geq 0$. With one variable for each of the n points x_n.

From here we still want to maximize the margin, but we also want to penalize points which are misclassified and thereby put on the wrong side of our decision boundary. We therefore want to solve formular (4.7) instead of formular (4.6).

$$\min_{\omega,b,\zeta}\left\{\frac{1}{2}w^Tw+C\sum_{n=1}^{N}\xi_n\right\} \tag{4.7}$$

$$y_i(w^T\phi(x_i)+b)\geq 1-\xi_n, \qquad \mathrm{n}=1,\dots,\mathrm{N}$$

Where $C>0$ is controlling the degree of which we want to penalize these misclassifications. The optimization of this parameter will be elaborated in section 4.5. The exact justification for deriving this as the maximum margin problem with the introduced slack variables can be seen in *Pattern Recognition and Machine Learning.*

[10] (Bishop, 2006) p. 326-336

From here we want to solve this minimization problem while upholding our constraint. This leads to a dual problem which can be solved using Lagrange multipliers. The following Lagrangian function can be constructed:

$$L(w,b,a) = \frac{1}{2}||w||^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N}\mathrm{a_n}\{t_n\mathrm{y(x_n)} - 1 + \xi_n\} - \sum_{n=1}^{N}\mu_n\xi_n$$

(4.8)

With the constraints

$$a_n \geq 0 \quad and \quad \mu_n \geq 0 \quad and \quad \xi_n \geq 0 \quad and \quad \mu_n\xi_n \geq 0$$

$$t_n y(x_n) - 1 + \xi_n \geq 0$$

$$\mathrm{a_n}(t_n\mathrm{y(x_n)} - 1 + \xi_n) = 0$$

Where $a = (a_1, \dots, a_N)^T$ and $\mu = (\mu_1, \dots, \mu_N)^T$, which are the Lagrangian multipliers. From here we want to obtain the dual Lagrangian. Which can be found by eliminating w, b and ξ_n by finding the derivatives with respect to these and set them equal to 0. We will list these since the derivative with respect to w is important since it is used to make predictions later.

$$\frac{\partial L}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{N}\mathrm{a_n}t_n, \quad \frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{N}\mathrm{a_n}t_n = 0, \quad \frac{\partial L}{\partial \xi_n} = 0 \Rightarrow a_n = \mathrm{C} - \mu_\mathrm{n}$$

(4.9)

After eliminating w, b and ξ_n we arrive at the dual Lagrangian:

$$\tilde{L}(a) = \sum_{n=1}^{N}\mathrm{a_n} - \frac{1}{2}\sum_{n=1}^{N}\mathrm{a_n a_m}t_n t_m k(x_n, x_m)$$

(4.10)

This is maximized with respect to the constraints:

$$0 \geq a_n \geq C, \qquad n = 1, \dots, N$$

$$\sum_{n=1}^{N}\mathrm{a_n}t_n = 0$$

It is this optimization problem which is solved to train the SVM model whereafter we can use the resulting values for a to make a prediction of a new given point x. The predictions can now be made by substituting w found from the derivative of the Lagrangian, shown in formular (4.9), into formular (4.5), whereafter we arrive at formular (4.11).

$$y(x) = \sum_{i=1}^{N}\mathrm{a_n}t_n k(x, x_n) + b$$

(4.11)

Multi class SVM

At this point we have introduced the SVM as a binary classification between two classes. To achieve multiclass SVM, we continue with this view in the training, but create multiple models to achieve our predictions. The number of models vary depending on the method used. We still train it using binary classification between one class, and either all other classes together or each individual class. These two different methods are respectively called *one vs the rest* and *one vs one*.

For the *one vs one* method we construct $\frac{K(K-1)}{2}$ SVMs , where K is the number of classes. What this entails is that we construct an SVM for each combination of classes. Meaning that if we have five classes class 1 is trained against class 2-5 and class 2 is trained against class 1 and 3-5 and so on. New data is then predicted by finding the class which receives most votes. Meant as the class with the largest number of models predicting that class.

For the *one vs the rest* method we construct K SVMs, where K is the number of classes. From here the k^{th} model $y_k(x)$ is trained using the training data classified as class C_k as the positive value and the data for all other classes as the negative value. We then make predictions for new data x by finding the k, which results in the maximum value of $y_k(x)$. This can be seen mathematically in formular (4.12). It is *one vs the rest* which is the standard used in Sickit-learn and is the method which will be used in this thesis.

$$y(x) = \max_k y_k(x) \tag{4.12}$$

4.4.2. SVM regression

SVMR is a variation of the standard SVM, which is used for classification. The reason these two methods are bundled together is because of the mathematical method used to derive their models. The two methods use two different concepts in machine learning, where the SVMC is a classification model and SVMR is a regression model. Common factor for regression models is the minimization of its error function. Furthermore, the two models also have two different goals. For SVMC the goal is to find a hyperplane which separates the classes by maximizing the margins between the hyper plane and the support vectors. For SVMR the goal is to find a decision boundary which best fits the data, which is done by minimizing the margin between the decision boundary and the support vectors, while also minimizing the error function for the regression model. Note that the points can lie within an ϵ-tube, which determines an area which the points can lie within while still accepting the decision boundary. This tube is described as the area between $y + \epsilon$ and $y - \epsilon$, where y is the decision boundary and ϵ is parameter determining the size of this area. The area can be seen on Figure 4.7.

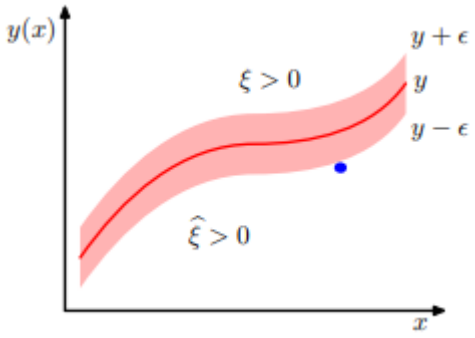

Figure 4.7: SVMR slack variables and decision boundary represented. (Bishop, 2006, s. 340) Figure 7.6

SVMR mathematical concept

The mathematical explanation behind SVM regression to be presented are based on the derivations found in *Pattern Recognition and Machine Learning section 7.1.4*[11].

For the mathematical background of the SVMR we can draw a parallel to SVMC. Where we wanted to maximize the margin by minimizing formular (4.7). For SVMR we want to minimize the margin which can be done similar to that of formular (4.7). The difference is with the slack variable. For the SVMC we wanted the slack variable to account for the class being misclassified. For SVMR we want to account for the data to be either above or below the ϵ-tube. Thus, creating the need for two slack variables ξ and $\hat{\xi}$, where they respectively account for the data being either above or below the ϵ-tube shown on Figure 4.7. Then to solve this problem we want to minimize (4.13), with constraints listed below it.

$$C\sum_{n=1}^{N}(\xi_n + \hat{\xi}_n) + \frac{1}{2}||w||^2 \qquad (4.13)$$

$$\xi \geq 0 \quad and \quad \hat{\xi} \geq 0$$

$$t_n \leq y(x_n) + \epsilon + \xi_n$$

$$t_n \geq y(x_n) - \epsilon - \hat{\xi}_n$$

Where C denotes the regularization term and ϵ determines the size of the tube. Both parameters will be optimized following the process described in section 4.5.

The process to arrive at this as the minimum margin problem which must be solved to solve the SVMR, can be seen in *Pattern Recognition and Machine Learning*. This can then be solved by introducing Lagrange multipliers $a_n \geq 0$, $\hat{a}_n \geq 0$, $\mu_n \geq 0$ and $\hat{\mu}_n \geq 0$. The Lagrangian and the process of arriving at the dual problem shown in formular (4.15) with its appropriate constraints, can be seen in *Pattern Recognition and Machine Learning*. During this process we eliminate the

[11] (Bishop, 2006)p. 339-344

variables w, b, ξ_n and $\hat{\xi}_n$ from the Lagrangian in the same way as we did for SVMC, by finding the derivative with respect to listed variables. We will not list the Lagrangian nor the derivative since this process is similarly to that of SVMC. We will though give the derivative with respect to w since this is later used to make the predictions for the model.

$$\frac{\partial L}{\partial w} = 0 \Rightarrow w = \sum_{n=1}^{N} (a_n + \hat{a}_n)\, k(x, x_n)$$

(4.14)

$$\tilde{L}(a, \hat{a}) = -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M}(a_n - \hat{a}_n)(a_m - \hat{a}_m)k(x_n, x_m) - \epsilon\sum_{n=1}^{N}(a_n + \hat{a}_n) + \sum_{n=1}^{N}(a_n + \hat{a}_n)t\_n$$

(4.15)

$$0 \leq a_n \leq C$$

$$0 \leq \hat{a}_n \leq C$$

It is the maximization of this dual problem which is the training of the SVMR model. The predictions can now be made by substituting w found from the derivative of the Lagrangian, shown in formular (4.14), into formular (4.5), whereafter we arrive at formular (4.16).

$$y(x) = \sum_{n=1}^{N}(a_n + \hat{a}_n)k(x, x_n) + b$$

(4.16)

The exact justification for deriving this as the prediction of new points can be seen in *Pattern Recognition and Machine Learning*.

4.5. Hyperparameters in SVM

We are now at stage in the process shown on Figure 3.2 which is shown in the rectangle labeled 'model training and hyperparameter optimization'. At this stage we will be training optimizing each of the models listed in Table 4.6. For our project we have chosen the scikit-learn implementation of SVMC and SVMR. For both methods it implements the formulars listed in sections 4.4.1 and 4.4.2, and allows us to adjust several different hyperparameters. The exact parameters which we have chosen to adjust will be explained in this section. The way we will perform this optimization is by using the 'GridSearchCV' package from scikit-learn which will train each combination of the hyper parameters mentioned later in this section. This will for SVMC find the model with the highest accuracy to be optimal and for SVMR will use the R^2-metric to decide the optimal model.

4.5.1. Regularization term C for SVMC and SVMR

The first hyperparameter we have chosen to optimize is the regularization parameter C, which is present in formular (4.7) and formular (4.13). For respectively the training of the SVMC and SVMR. As said in the section, C allows us to choose which degree we want to penalize misclassifications. Hence it can possibly allow us to achieve a higher accuracy of our model. For scikit-learn the standard is set to $C = 1.0$, which is the value used while choosing the optimal HOG

features. We will finetune this value for both SVMR and SVMC, firstly by checking which C value results in the best accuracy of the model. Later will we examine the results using a custom scoring function, which will account for the case that misclassifying class 0 as 1 is better than classifying it as class 9. We will be testing two scoring functions. To start with we have chosen to try a cost matrix with a cost of 0 if the class is correctly classified and add one for each step away from the class we go. Another Matrix could be that after a distance of 4 classes the cost will stagnate such that it is the same cost to misclassify 0 as 4 as it is to misclassify 0 as 10. Both matrixes can be seen in Figure 4.8.

```
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
 [1, 0, 1, 2, 3, 4, 5, 6, 7, 8],
 [2, 1, 0, 1, 2, 3, 4, 5, 6, 7],
 [3, 2, 1, 0, 1, 2, 3, 4, 5, 6],
 [4, 3, 2, 1, 0, 1, 2, 3, 4, 5],
 [5, 4, 3, 2, 1, 0, 1, 2, 3, 4],
 [6, 5, 4, 3, 2, 1, 0, 1, 2, 3],
 [7, 6, 5, 4, 3, 2, 1, 0, 1, 2],
 [8, 7, 6, 5, 4, 3, 2, 1, 0, 1],
 [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]]
```

```
[[0, 1, 2, 3, 4, 4, 4, 4, 4, 4],
 [1, 0, 1, 2, 3, 4, 4, 4, 4, 4],
 [2, 1, 0, 1, 2, 3, 4, 4, 4, 4],
 [3, 2, 1, 0, 1, 2, 3, 4, 4, 4],
 [4, 3, 2, 1, 0, 1, 2, 3, 4, 4],
 [4, 4, 3, 2, 1, 0, 1, 2, 3, 4],
 [4, 4, 4, 3, 2, 1, 0, 1, 2, 3],
 [4, 4, 4, 4, 3, 2, 1, 0, 1, 2],
 [4, 4, 4, 4, 4, 3, 2, 1, 0, 1],
 [4, 4, 4, 4, 4, 4, 3, 2, 1, 0]]
```

Figure 4.8: Cost matrixes for optimization of C value. (1) Step away from correct class. (2) Stagnate at 4

4.5.2. Kernel for SVMC and SVMR

Another hyperparameter we have chosen to optimize is the kernel used for the training and prediction of the model. Here we will try to find the optimal kernel of the four listed in Table 4.5, 'linear', 'poly', 'RBF', 'sigmoid', which are all implemented in the scikit-learn library. While choosing the optimal values for the HOG features will we be using the 'RBF' kernel since it is the kernel generally used for SVMC. This is also the reason that scikit-learn has it as its standard kernel if nothing else is stated. Later when we have found optimal HOG parameters from the combination shown in Table 4.3, we will try all four kernels while training the model and, in that way, find the optimal kernel for our model.

4.5.3. Class weights for SVMC and SVMR

The class weights are a parameter which puts a weight on the regularization parameter C for each class i. This can be written as, class weight[i] $\cdot$ C where i is the i^{th} class. As standard this is set to 'None' where each class is given a weight of 1. Another available value is 'balanced' where the weights are automatically adjusted to be inversely proportional to class frequencies. This can be written as, $\text{class weight[i]} = \frac{n_{samples}}{\text{n}_{\text{classes}} \cdot \text{np.bincount(y)}}$, where $n\_samples$ is number of data in class i, $n\_clasess$ is the number of classes and $bincount$ is the total number of data. This will consider if the data is not evenly distributed between classes which it is not in our case, recall Figure 3.4. We will test both a class weights of 'None' and 'balanced', and see which gives a better accuracy, but also which gives a better distribution of classifications between classes. This is the hyperparameter which we will not be using grid search for since the grid search only take the accuracy into consideration and not the distribution of the classes.

4.5.4. γ value for SVMC and SVMR

The next hyperparameter we will be optimizing is the γ value. This is the kernel coefficient for the 'RBF', 'sigmoid' and 'poly' kernel and is used to adjust the distance to which a single training data can affect the decision boundary. A high value will result in a more tightly fitted decision boundary

around the training data which can cause overfitting. A low γ value will result in a loosely fitted decision boundary and can therefore result in underfitting. In scikit-learn it has a standard value of 'scale' which is $\gamma = \frac{1}{n_{features} \cdot X.var()}$. Another option is 'auto' which is $\gamma = \frac{1}{n_{features}}$. It can also be given as a concrete float value. This will be set to default until we have determined the optimal HOG parameters but later on will be optimized for our specific models.

4.5.5. Epsilon value for SVMR

The last hyperparameter we will be optimizing ϵ which is described in section 4.4.2. This value determines the width of the tube around the decision boundary where points can lie within without the model considering it an error. This value is for scikit-learn as standard set as 1. To optimize this, we will be using the grid search at first together with the different kernels, C values and γ values, but afterwards we will finetune it to the best accuracy.

4.6. Combination of models

In his book Bishop describes the advantages of using a combination of models where he states *"It is often found that improved performance can be obtained by combining multiple models together in some way, instead of just using a single model in isolation"*[12]. Which is why we have chosen to use a committee of models to improve our final model. Note that this step is performed after the optimization of hyperparameters.

We have chosen to use committee to make a combination of the models described in section 4.4.1 and section 4.4.2 for both HOG and Haralick textual features. This process is simply finding the average prediction of the models and using that as the prediction for our committee. This average is found as the result of formular (4.17) for M different models.

$$y_{com}(x) = \frac{1}{M} \sum_{m=1}^{M} y_m(x)$$

(4.17)

Note as in earlier sections y is the prediction of the model where in this case y_m is the prediction of the m^{th} model and y_{com} is the prediction of the committee.

The main advantage of using a committee can be seen by looking at formular 14.14 in "*Pattern Recognition and Machine Learning*"[13]. Which can be seen in formular (4.18).

$$E_{COM} = \frac{1}{M} E_{AV}$$

(4.18)

Where E_{COM} is the error of the committee model and E_{AV} is the average error of the M models. This indicate that the error of the committee can be reduced by a factor of M, by just averaging the predictions between the M models.

[12] (Bishop, 2006, s. 653)
[13] (Bishop, 2006)

For the committee we will try a combination of the models shown in Table 4.6, where we will try all combinations:

Feature/SVM	SVMC	SVMR
HOG	HOG SVMC	HOG SVMR
Haralick	Haralick SVMC	Haralick SVMR

Table 4.6: Different models in this thesis.

4.7. Training and validation data.

The final step of developing a machine learning algorithm is to validate it. This is done by separating the data into a training set and validation set. We did this in two different ways. The first way was collecting all our data in a data frame. We randomly shuffle the data frame and take 80% of the data to use for training of our models and 20% for validation. This random shuffle ensured that our validation data and training data would be evenly distributed and avoid bias which certain distributions of data might contain. When we are validating our model, we are going to train each model 10 times and take the average accuracy of that model. By doing this we make the model easier to compare to each other.

The second way was to use four of the years as training data and then validate on a single year. This is the final step seen in Figure 3.2. The reason for this is that it would reflect how it would be used in a real-life setting. E.g., if one where to apply the model to a brand-new year. We chose 2019 as our validation year since the data from 2019 accounts for approximately 20% of the data. For this model we are going to use the optimal hyperparameters that we have found. How these are found are explained in section 4.5. We would expect the one-year validation to perform worse than the other since the model would not have training data from that specific year that it validated on.

Considering the range of plus/minus 10% to be correct

For the committee results and, we have chosen to list the results considering the class before and after the target value as being correct predictions. One reason for this is the manual score given by the seed producers, which were subjective and could vary from person to person and from month to month. Therefore, the manual scoring used as targets by the model could easily be off by 10%. Another reason for this is our bin number containing 10% lodging each meaning that it does not differentiate between 19.9% and 20% where these two would be considered respectively class 1 and 2. This change should only have a visible effect if the model prediction already lies within one class of the target class and all predictions further away are not impacted.

5. Results/analysis

The following sections will describe the results that we have achieved. First, we are going to explain the way we chose to structure our results. We are then going to explain the results that we achieved by training models on HOG features. Next, we are going to continue with the models trained with Haralick textual features. Afterwards we will show the results for our optimization of hyper parameters. Finally, we are going to show the result of using a committee and validating the optimized model on the data from 2019.

5.1. Result structure.

The following section will describe the format which we saved our results. In Figure 5.1(a) we see a classification report. The report provides metrics such as precision, recall and F1 score. Here we will

mainly be using the weighted average for the F1-score and the accuracies for the models. The report also provides an overview of the validation data used. In Figure 5.1(a) we are validating our model on 983 pictures. We see that we achieve an accuracy of 47%. We see that our model has a high precession (75%) while detecting lodging between 0-10% (bin 0). For the SVMR we took the predictions from the regression model and translated them into which of the 10 classes they would correspond to. This would mean if it predicted 16.66 it would be equal to the classification being class 1. From this we constructed the same classification report and confusion matrix made for the SVMC models. The formulars, as described in scikit-learn, and a small description of the metrics can be seen below.

$$Precision = \frac{TP}{TP + FP} = \frac{\#(plots\ with\ LS\ classified\ as\ LS)}{\#(plots\ classified\ as\ LS)}$$

Where TP is the number of true positive, FP is the number of false positive, TN is the number of true negatives and FN is the number of false negative. Precision describes the percentage of times the class is predicted where the prediction was correct.

$$Recall = \frac{TP}{TP + FN} = \frac{\#(plots\ with\ LS\ classified\ as\ LS)}{\#(plots\ with\ LS)}$$

Recall describes the percentage of all plots belonging to a class which has been predicted correctly.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{\#(correctly\ classified\ plots)}{\#(plots)}$$

Accuracy is the percentage of all plots which where correctly classified.

$$F1_{score} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

F1-score is a combination of both precision and recall. This is found as the harmonic mean of the two, which is why we will be referring to this metric mostly in addition to the overall accuracy of each model.

With each of the classification reports we also produced a confusion matrix that more precisely describes what predictions the model makes. An example of this can be seen on Figure 5.1(b). If we look at the LS between 80-90% (bin 8) we see that our model predicts class 0 once and class 3 thrice and so forth.

Accuracy: 0.47100712105798576

	precision	recall	f1-score	support
0	0.75	0.70	0.73	446
1	0.21	0.38	0.27	117
2	0.09	0.07	0.08	75
3	0.17	0.18	0.17	73
4	0.13	0.10	0.11	60
5	0.18	0.06	0.09	34
6	0.37	0.34	0.35	65
7	0.49	0.57	0.53	83
8	0.83	0.33	0.48	30
accuracy			0.47	983
macro avg	0.36	0.30	0.31	983
weighted avg	0.49	0.47	0.47	983

314	92	12	8	10	2	6	2	0	0
41	44	12	12	4	0	3	1	0	0
22	27	5	6	4	2	4	5	0	0
21	19	9	13	6	2	1	2	0	0
12	14	5	9	6	1	5	8	0	0
3	8	5	9	4	2	3	0	0	0
3	3	3	7	9	1	22	17	0	0
2	3	4	9	1	1	14	47	2	0
1	0	0	3	1	0	1	14	10	0
0	0	0	0	0	0	0	0	0	0

Figure 5.1: (a) classification report for block size (8x8) cells per block (2x2), picture size min. (b) Confusion matrix for block size (8x8) cells per block (2x2), picture size min.

5.2. Histogram of oriented gradient (HOG) parameters.

The following section will show the results of different picture, block, and cell sizes for training an SVMC model on HOG features. We have collected all the classification reports and created plots which shows the different average accuracies for each of the block, cell, and image size. We are testing three different image sizes, Min, mean and max, which can be seen in Table 4.2. We have selected four different combinations of Block and cell sizes. These can be seen in the Figure 5.2 or by recalling Table 4.3.

5.2.1. Cell, Block and Image sizes

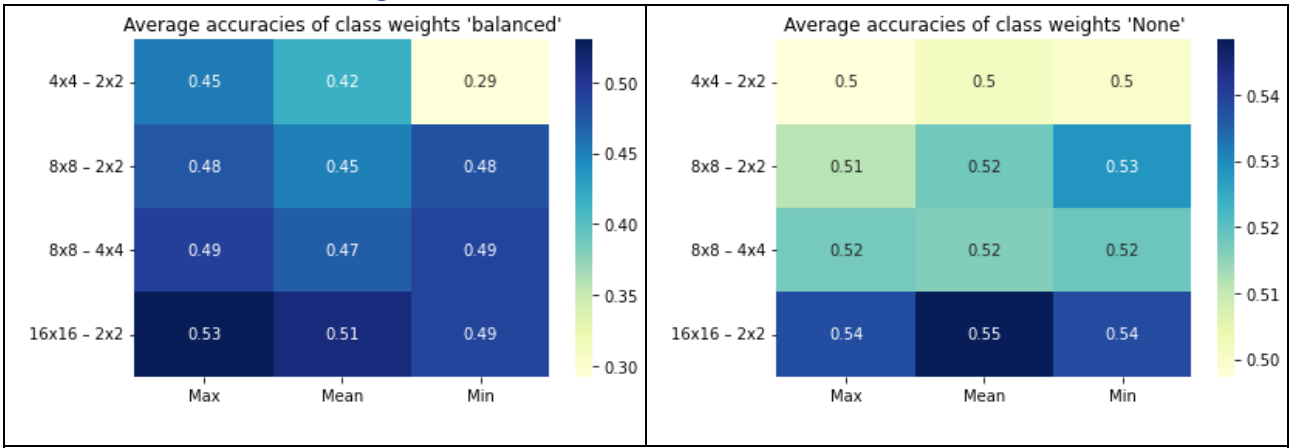

Figure 5.2: average Accuracies of cell, block and image sizes. (a)"balanced" class weights. (b) "None" class weights

On the top of Figure 5.2(a) we see the result of 4x4 pixels per cell and 2x2 cells per block. We see that the picture of the maximum size achieves an average accuracy of 45%. We see that the best accuracy for (8x8-2x2) achieves an average accuracy of 48% which is for both the max and min sized pictures. We see that the best accuracy for (8x8-4x4) is the max and min sized pictures with an average accuracy of 49%. We test a size of (16x16-2x2). Here we see that the max sized pictures achieve 53%, while the mean achieve 51% and the min 49%. This means that the best performing cell and block size is (16x16-2x2), the best performing image size is max.

On the top of Figure 5.2(b) we see the result for (4x4-2x2). All the image sizes achieve an average accuracy of 50%. Next, we see the accuracies for (8x8-2x2) achieves 53% which is for the min sized pictures. We see that all the average accuracies for (8x8-4x4) is 52%. We test a size of (16x16-2x2). Here we see that the mean sized pictures achieve 55%, while the max and min achieve 54%. Once again, we see that the cell and block size that performs the best is (16x16 – 2x2). For that block and cell size the mean image size performs the best.

In appendix 9.1.1 we can see the standard deviation for Figure 5.2. However, they were so low that we did not include them.

5.2.2. Class weights of 'balanced' vs 'None'.

In section 4.5.3 we explained the difference between setting class weight equal to "balanced" and 'None'. The following section are going to show the results both for the 'balanced' and 'None' SVMC models.

On Figure 5.3(b) we see the results of a classification report of the 'balanced' SVMC using (16x16 - 2x2) HOG features. On Figure 5.3(a) we see the results of a classification report of the 'None' SVMC using (16x16-2x2) HOG features. We see that the accuracy goes down from 57% to 48%, and the weighted average for F1-score increases from 44% to 48%. If we look at Figure 5.2 we do though see that the 'None' SVMC perform slightly better for all block, cell, and image sizes. The results are going to be discussed in section 6.1.1.

Accuracy: 0.5656154628687691

	precision	recall	f1-score	support
0	0.58	1.00	0.74	473
1	0.60	0.03	0.05	119
2	0.33	0.02	0.03	66
3	0.00	0.00	0.00	66
4	0.33	0.02	0.03	59
5	0.00	0.00	0.00	46
6	0.48	0.43	0.45	51
7	0.49	0.67	0.56	73
8	0.82	0.30	0.44	30
accuracy			0.57	983
macro avg	0.40	0.27	0.26	983
weighted avg	0.48	0.57	0.44	983

Accuracy: 0.4832146490335707

	precision	recall	f1-score	support
0	0.76	0.72	0.74	462
1	0.21	0.32	0.25	105
2	0.06	0.04	0.05	75
3	0.19	0.20	0.19	65
4	0.16	0.17	0.17	53
5	0.36	0.10	0.16	50
6	0.36	0.42	0.39	62
7	0.40	0.58	0.47	74
8	0.69	0.31	0.42	36
9	0.00	0.00	0.00	1
accuracy			0.48	983
macro avg	0.32	0.29	0.28	983
weighted avg	0.50	0.48	0.48	983

Figure 5.3: classification report of the svm classifier using (16x16-2x2) HOG features. (a) "None" (b)"balanced".

5.2.3. Remarks on HOG metrics

The figures and tables of section 5.2 can help us decide some parameters to move forward with to improve our model in other ways. We have chosen to extract HOG features using 16x16 pixels per cell and 2x2 cells per block. We have chosen the image size mean (144x496 pixels). This decision will be further discussed in section 6.1.1.

We have also shown the differences between setting class weight to' balanced' or 'None'. What this means is described in section 4.5.3. We see that we achieve a higher accuracy by setting class weights to 'None' compared to 'balanced'. However, the 'balanced' model almost only guesses on 0 lodging. This will be further discussed in section 6.1.1. We are going to continue with class weight 'None' in our further investigations.

5.2.4. SVM Regression

In section 4.4.2 we described how we can train an SVMR model. As mentioned in section 5.2.3, from now on will we only run on the (16x16-2x2) HOG size and mean image size. This were also used for the SVM regression model which were trained 10 times where the average accuracy can be seen in Table 5.1. We see that we achieve an average accuracy 34% and the standard deviation is low enough to be disregarded. The result will be discussed in section 6.1.1.

Average	Standard deviation
0.3493	0.0109

Table 5.1: Average and standard deviation for HOG SVMR trained 10 times

5.3. Data Augmentation

In section 3.2 we discussed how we could use data augmentation to increase the model's robustness. We will be looking at two cases for the HOG features (16x16-2x2). These are the class weight 'None', where the results can be seen in Figure 5.4, and the class weight 'balanced', where the results can be seen in Figure 5.5.

On Figure 5.4 we see that the standard version has an accuracy of 56% where the mirrored version has 54%. We also see that the weighted averages for F1-score are 44% and 43%. Meaning a slightly worsening of the model. On Figure 5.5 we see the data trained with the class weight 'balanced'. This again shows similarly to the class weights 'None' version that the mirroring results in a fall in accuracy from 50% to 49%. The weighted average went from 48% to 50%. This will be discussed in section 6.1.1

Accuracy: 0.5635808748728383

	precision	recall	f1-score	support
0	0.59	1.00	0.74	476
1	0.21	0.03	0.05	117
2	0.00	0.00	0.00	78
3	0.17	0.02	0.03	61
4	1.00	0.02	0.04	55
5	0.50	0.03	0.05	38
6	0.42	0.27	0.33	63
7	0.45	0.70	0.55	71
8	0.88	0.29	0.44	24
accuracy			0.56	983
macro avg	0.47	0.26	0.25	983
weighted avg	0.48	0.56	0.44	983

Accuracy: 0.5439979654120041

	precision	recall	f1-score	support
0	0.59	0.98	0.74	1868
1	0.00	0.00	0.00	440
2	0.00	0.00	0.00	269
3	0.30	0.03	0.06	301
4	0.15	0.03	0.05	210
5	0.24	0.02	0.04	204
6	0.38	0.33	0.35	266
7	0.38	0.67	0.49	266
8	0.57	0.27	0.37	106
9	0.00	0.00	0.00	2
accuracy			0.54	3932
macro avg	0.26	0.23	0.21	3932
weighted avg	0.39	0.54	0.43	3932

Figure 5.4:classification report for 16x16-2x2 'None', img size mean, right has data augmentation.

Accuracy: 0.5025432349949135

	precision	recall	f1-score	support
0	0.74	0.81	0.77	422
1	0.28	0.33	0.31	117
2	0.16	0.09	0.12	87
3	0.15	0.16	0.15	70
4	0.12	0.09	0.10	56
5	0.08	0.02	0.03	48
6	0.35	0.36	0.36	64
7	0.43	0.62	0.51	86
8	0.73	0.33	0.46	33
accuracy			0.50	983
macro avg	0.34	0.31	0.31	983
weighted avg	0.47	0.50	0.48	983

Accuracy: 0.4910986775178026

	precision	recall	f1-score	support
0	0.82	0.69	0.75	1893
1	0.22	0.35	0.27	450
2	0.18	0.15	0.17	265
3	0.18	0.18	0.18	280
4	0.16	0.15	0.15	221
5	0.16	0.10	0.12	180
6	0.36	0.40	0.38	260
7	0.44	0.61	0.51	284
8	0.49	0.43	0.46	99
accuracy			0.49	3932
macro avg	0.33	0.34	0.33	3932
weighted avg	0.53	0.49	0.50	3932

Figure 5.5:classification report for 16x16-2x2 'balanced', img size mean, right has data augmentation.

5.4. Data balancing.

In section 3.3 we described four different approaches to try to create a more balanced dataset.

16x16, 2x2, 10 classes	Normal	Mirror	Sorted zero Normal	Sorted zero Mirror	Sorted zero Mirror 2
Accuracy	0.56	0.54	0.38	0.36	0.34
Macro avg F1	0.25	0.21	0.31	0.27	0.26
Weighted avg F1	0.44	0.43	0.32	0.31	0.28

Figure 5.6: Accuracy comparison for Normal, Mirror and sorted zeroes.

In Figure 5.6 we see the results of this overall balancing. We see that no version of balancing improves the model. We see the optimal model is the one using none of the balancing methods described, where the second best is the mirrored version seen in Figure 5.4 right. We see that the sorting of zeroes generally worsened the model where the best accuracy where 38%. This will be further discussed in section 6.1.1.

5.5. Haralick metrics.

In the following section we will evaluate the result of training an SVMC from Haralick textual features. We are going to discuss setting class weights 'balanced' vs 'None'. We have extracted features from the same three different sizes, as we did with the HOG features. Here we also tried to use an additional feature which is the date of the manual classification. The process of extracting Haralick textual features is described in section 4.3.2.

5.5.1. 'balanced' vs 'None'.

```
Accuracy: 0.3041709053916582
              precision    recall  f1-score   support

           0       0.71      0.48      0.57       463
           1       0.13      0.20      0.15       106
           2       0.14      0.05      0.08        75
           3       0.08      0.13      0.10        77
           4       0.14      0.31      0.20        64
           5       0.07      0.07      0.07        44
           6       0.38      0.10      0.16        61
           7       0.31      0.06      0.10        68
           8       0.08      0.46      0.13        24
           9       0.00      0.00      0.00         1

    accuracy                           0.30       983
   macro avg       0.20      0.19      0.16       983
weighted avg       0.42      0.30      0.33       983

220 66 3 65 61 19 1 0 28 0
38 21 3 21 10 2 1 1 9 0
18 14 4 12 6 6 3 2 10 0
19 15 4 10 7 6 2 4 10 0
4 11 4 7 20 7 1 2 8 0
4 6 3 4 12 3 1 0 11 0
5 13 2 4 7 2 6 0 22 0
2 11 5 4 11 1 0 4 30 0
0 7 1 0 4 0 1 0 11 0
0 1 0 0 0 0 0 0 0 0
```

```
Accuracy: 0.43540183112919634
              precision    recall  f1-score   support

           0       0.44      1.00      0.61       428
           1       0.00      0.00      0.00       107
           2       0.00      0.00      0.00        86
           3       0.00      0.00      0.00        91
           4       0.00      0.00      0.00        61
           5       0.00      0.00      0.00        49
           6       0.00      0.00      0.00        63
           7       0.00      0.00      0.00        68
           8       0.00      0.00      0.00        30

    accuracy                           0.44       983
   macro avg       0.05      0.11      0.07       983
weighted avg       0.19      0.44      0.26       983

428 0 0 0 0 0 0 0 0 0
107 0 0 0 0 0 0 0 0 0
86 0 0 0 0 0 0 0 0 0
91 0 0 0 0 0 0 0 0 0
61 0 0 0 0 0 0 0 0 0
49 0 0 0 0 0 0 0 0 0
63 0 0 0 0 0 0 0 0 0
68 0 0 0 0 0 0 0 0 0
30 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

Figure 5.7: Haralick textual features - Classification reports and confusion matricies (a)"balanced" (b) "None"

To the right we see the classification report and confusion matrix of extracting Haralick textual features on Max sized images and training an SVM classifier which has class weights 'balanced'. To the left we see the 'None' version. We see that the 'None' only predicts class 0. This will be further discussed in section 6.1.3, but going forward we will be using class weight 'balanced' to train Haralick feature models.

5.5.2. Image Size

Img size	Min	Mean	Max
Mean accuracy	0.2902987145103117	0.2970498474059003	0.20020345879959306
Standard deviation	0.022242467042581624	0.0319318964994635	0.03342797737915617

Table 5.2: average accuracies of training an SVM classifier on Haralick textual features. Bin size is 10.

In Table 5.2 we see the average accuracies from our classification report which we generated using Haralick textual features. We notice that we obtain a slightly more accurate model using the mean image size achieving 30% and that the biggest image size seems to perform poorly with 20%. All the standard deviations of Table 5.2 are low enough to be disregarded. This will be further discussed in section 6.1.3.

5.5.3. Date

On Table 5.3 we see the result of taking the date into account. It is worth noticing that we achieve a higher accuracy in the mean and min sized compared to Table 5.2. The standard deviation of Table 5.3 are all low enough to be disregarded. Taking the data into account is discussed in section 6.1.2

Img size	Min	Mean	Max
Mean accuracy	0.3043404543913191	0.3035492257262349	0.2303605742059455
Standard deviation	0.022881578202303847	0.030631904742197354	0.03555447868174919

Table 5.3: plot of accuracies of training an SVM classifier on Haralick textual features. Bin size is 10. Class weights is 'balanced'. The date which the image is taken, and LS is assessed is considered.

5.5.4. Haralick SVMR

In section 4.4.2 we described how we can train an SVMR model. In this section the results from training the SVM regression model on Haralick textual features are listed. This model was trained 10 times where the average accuracy can be seen in Table 5.4. The standard deviation is low enough to be disregarded. The result will be discussed in section 6.1.1.

Average	Standard deviation
0.3391	0.0080674

Table 5.4: Average and standard deviation for Haralick SVMR trained 10 times

5.6. Optimization of hyper parameters

In this section we will list the results of the hyperparameter optimization described in section 4.5. As mentioned for SVMC we will optimize the hyperparameters C, γ and kernel function. These parameters will also be optimized for SVMR in addition to the hyperparameter ϵ. The optimal hyperparameters found in this section can be seen in Table 5.5 and the results will be discussed in section 6.1.4.

Model	Kernel	γ	C	ϵ	Avg-Acc
HOG Features SVMC	RBF	Scale	2.9764	None	0.5656
Haralick Features SVMC	Poly	Scale	3.0888	None	0.4089
HOG Features SVMR	RBF	Scale	48.3293	0.0001	0.4025
Haralick Features SVMR	RBF	Scale	284.8036	10.0	0.3204

Table 5.5: Optimal Hyperparameters for the 4 models. Note that Avg-Acc is the average accuracy when running the model with parameter 10 times.

5.6.1. HOG Features SVMC

We started by performing a grid search over the different kernels and γ values with three different C values, which resulted in the default values for kernel function ('RBF') and γ value ('scale') to be the optimal. An increase in the accuracy was still found, which indicates it is the C value which creates a difference. We therefore ran the optimization again focusing only on the C value to finetune this even further. This gave an optimal C value of 2.98, which gave a rise in accuracy of 2%. The classification report for this can be seen in Figure 5.8. This shows a change in the accuracy from 53% to 55%. Furthermore, there is a change in the weighted average of the F1-score from 40% to 46% which indicates an increase in robustness for the model.

```
Accuracy: 0.5300101729399797
              precision    recall  f1-score   support

           0       0.55      1.00      0.71       431
           1       0.36      0.04      0.07       104
           2       0.00      0.00      0.00        83
           3       0.00      0.00      0.00        75
           4       0.00      0.00      0.00        58
           5       0.00      0.00      0.00        55
           6       0.44      0.42      0.43        66
           7       0.42      0.67      0.52        76
           8       1.00      0.23      0.37        35

    accuracy                           0.53       983
   macro avg       0.31      0.26      0.23       983
weighted avg       0.38      0.53      0.40       983

430 0 0 0 0 0 0 1 0 0
97 4 0 0 0 0 1 2 0 0
71 5 0 0 0 0 1 6 0 0
66 2 1 0 0 0 3 3 0 0
43 0 0 0 0 0 5 10 0 0
43 0 0 0 0 0 7 5 0 0
18 0 0 1 0 0 28 19 0 0
9 0 0 0 0 0 16 51 0 0
0 0 0 0 0 0 3 24 8 0
0 0 0 0 0 0 0 0 0 0
```

```
Accuracy: 0.5462868769074263
              precision    recall  f1-score   support

           0       0.61      0.97      0.75       431
           1       0.24      0.09      0.13       104
           2       0.29      0.06      0.10        83
           3       0.22      0.08      0.12        75
           4       0.25      0.07      0.11        58
           5       0.50      0.04      0.07        55
           6       0.42      0.41      0.41        66
           7       0.45      0.70      0.54        76
           8       1.00      0.34      0.51        35

    accuracy                           0.55       983
   macro avg       0.44      0.31      0.30       983
weighted avg       0.48      0.55      0.46       983

419 6 3 0 2 0 0 1 0 0
87 9 2 3 0 0 1 2 0 0
60 7 5 6 0 0 1 4 0 0
48 6 5 6 3 0 4 3 0 0
30 3 1 3 4 0 7 10 0 0
24 3 0 7 5 2 11 3 0 0
9 3 1 2 2 1 27 21 0 0
8 1 0 0 0 1 13 53 0 0
0 0 0 0 0 0 1 22 12 0
0 0 0 0 0 0 0 0 0 0
```

Figure 5.8: Classification reports and confusion matrices for: (a) Default values (b) Optimized values

At last, we optimized the C value using the two cost matrixes described in section 4.5.1, which gave the results seen in Table 5.6. Note that the 'Score' is score given by our cost functions, the 'Accuracy Default' is the accuracy obtained by the model with default hyper parameters for scikit-learn and 'Accuracy optimized' is the accuracy with the found optimal C-value.

Custom cost matrix	Step at 4	Distance
Optimal C value	2.98	2.98
Score	-777.4	-853.2
Accuracy Default	0.5595116988809766	0.5544252288911495
Accuracy Optimized	0.5595116988809766	0.5503560528992879

Table 5.6: Costum scorer results

For both cost functions the custom scorer did not have a big effect on the precision but had similar effect on the robustness of the model as with the optimization without the custom scorer. It is worth noting that it is the same C-value found in all optimizations. The overall optimized values have been found to be the one listed in Table 5.5, which will be used when running the committee.

5.6.2. Haralick textual features SVMC

We repeated the same procedure where we started by running a grid search with the kernel and γ-values, this time only processed with the default C-value "1.0". We found that the 'linear' kernel and the γ-value auto started an endless loop in the training of the model. Hence these values were skipped, which will be discussed in section 6.1.4. The grid search found that the optimal kernel was the 'polynomial' kernel.

We then finetuned the C value by running a grid search on the C-value. Here we found that the optimal C-value of those used were 3.09. This resulted in the classification reports and confusion matrix shown in Figure 5.9. Note that (a) is the default values for the hyperparameters and (b) is the optimal C-value. We see that the weighted average F1-score increased from 27% to 37% and the accuracy increased from 21% to 42%. Indicating an increase in robustness with the optimal values. The optimal

hyperparameters for Haralick SVMC can be seen in Table 5.5, which will be used for the committee later.

Accuracy: 0.2126144455747711

	precision	recall	f1-score	support
0	0.78	0.30	0.44	466
1	0.15	0.08	0.11	106
2	0.09	0.01	0.02	76
3	0.07	0.15	0.10	73
4	0.11	0.23	0.15	56
5	0.06	0.19	0.10	43
6	0.08	0.07	0.07	61
7	0.30	0.29	0.29	77
8	0.00	0.00	0.00	25
9	0.00	0.00	0.00	0
accuracy			0.21	983
macro avg	0.16	0.13	0.13	983
weighted avg	0.44	0.21	0.27	983

```
141 41 0 86 53 33 8 4 0 100
12 9 0 23 12 13 4 3 0 30
12 0 1 19 3 12 10 5 0 14
11 4 2 11 3 18 5 3 0 16
2 2 2 5 13 11 6 3 0 12
1 3 0 5 8 8 4 8 0 6
0 0 0 4 10 16 4 14 0 13
1 1 6 4 11 12 7 22 0 13
0 0 0 1 6 2 1 12 0 3
0 0 0 0 0 0 0 0 0 0
```

Accuracy: 0.4150559511698881

	precision	recall	f1-score	support
0	0.57	0.74	0.64	451
1	0.29	0.11	0.16	105
2	0.13	0.31	0.18	75
3	0.00	0.00	0.00	69
4	0.31	0.18	0.23	60
5	0.25	0.07	0.11	45
6	0.18	0.32	0.23	57
7	0.20	0.01	0.02	86
8	0.33	0.21	0.25	34
9	0.00	0.00	0.00	1
accuracy			0.42	983
macro avg	0.23	0.19	0.18	983
weighted avg	0.37	0.42	0.37	983

```
333 17 76 4 9 1 8 1 2 0
58 12 21 0 4 1 8 0 1 0
32 3 23 0 3 1 12 1 0 0
41 1 15 0 2 0 9 0 1 0
24 3 6 1 11 5 8 2 0 0
25 1 5 0 4 3 7 0 0 0
15 2 15 0 1 0 18 0 6 0
40 2 16 0 2 1 20 1 4 0
17 0 2 0 0 0 8 0 7 0
0 0 0 0 0 0 1 0 0 0
```

Figure 5.9: Claissifcation reports for the finetuning of the C-value. (a) Default values (b) Optimal C-value

5.6.3. HOG Features SVMR

Similar, to the process for SVMC we started by finding optimal kernels, γ, and three C-values. It found the default values for kernel function ('RBF') and γ-value ('scale') to be optimal and found that a higher C-value, than the standard value for C ('1.0'), increases the accuracy. We then finetuned the ϵ-value which was found to be optimal at 0.001. This was though the lowest value for the grid search, which could indicate a lower value could be better.

We further fine-tuned the C-value, which resulted in a C-value of 48.32. The classification reports and confusion matrixes can be seen on Figure 5.10. The big difference here is by finetuning the C-value we went from an accuracy of 34% with default value to 40% with the optimized C-value. We also see the weighted average for the F1-score went from 36% to 44%. It is worth noting that on the confusion matrix it went from never predicting classes 6, 7, 8 to making correct predictions on those classes. The optimized SVMR model has the values shown in Table 5.5, which will be used later for the committee.

```
Accuracy: 0.34282807731434384
              precision    recall  f1-score   support

           0       0.77      0.55      0.64       473
           1       0.15      0.42      0.22       117
           2       0.11      0.28      0.16        74
           3       0.08      0.09      0.09        64
           4       0.04      0.04      0.04        56
           5       0.10      0.02      0.03        52
           6       0.00      0.00      0.00        57
           7       0.00      0.00      0.00        68
           8       0.00      0.00      0.00        21
           9       0.00      0.00      0.00         1

    accuracy                           0.34       983
   macro avg       0.13      0.14      0.12       983
weighted avg       0.41      0.34      0.36       983

MSE 4.089521871820956
RMSE 2.0222566285763426
258 174 37 4 0 0 0 0 0 0
42 49 18 8 0 0 0 0 0 0
14 33 21 6 0 0 0 0 0 0
10 25 19 6 4 0 0 0 0 0
6 22 20 6 2 0 0 0 0 0
3 13 29 5 1 1 0 0 0 0
1 4 25 13 9 5 0 0 0 0
0 3 20 19 23 3 0 0 0 0
0 1 2 6 11 1 0 0 0 0
0 0 0 1 0 0 0 0 0 0
```

```
Accuracy: 0.40386571719226855
              precision    recall  f1-score   support

           0       0.85      0.59      0.69       473
           1       0.20      0.37      0.26       117
           2       0.14      0.26      0.18        74
           3       0.14      0.23      0.18        64
           4       0.11      0.18      0.14        56
           5       0.15      0.13      0.14        52
           6       0.39      0.23      0.29        57
           7       0.46      0.16      0.24        68
           8       0.50      0.05      0.09        21
           9       0.00      0.00      0.00         1

    accuracy                           0.40       983
   macro avg       0.29      0.22      0.22       983
weighted avg       0.53      0.40      0.44       983

MSE 2.2380467955239065
RMSE 1.4960102925862195
278 134 43 12 5 1 0 0 0 0
30 43 26 14 4 0 0 0 0 0
10 18 19 19 6 2 0 0 0 0
5 11 17 15 9 7 0 0 0 0
3 5 14 19 10 3 2 0 0 0
2 3 11 14 12 7 2 1 0 0
0 2 4 5 23 6 13 3 1 0
0 0 2 7 15 21 12 11 0 0
0 0 0 1 5 1 4 9 1 0
0 0 0 0 1 0 0 0 0 0
```

Figure 5.10: Finetuning of the C-value. (a) Default (b) optimized C-value and ϵ -values

5.6.4. Haralick textual features SVMR

Similar, to the process for SVMC we started by finding optimal kernels, γ, and three C values. For Haralick textual features the linear kernel and γ value 'auto' both resulted in an endless loop and is therefore omitted in this grid search. It found the default values for kernel function ('RBF') and γ-value ('scale') to be optimal and found that a higher C-value than the standard value for C ('1.0') to increase the accuracy. We then finetuned the ϵ-value which was found to be optimal at 10. This was though the highest value for the grid search, which could indicate a higher value could be better.

A finetuning of the C-value found that the optimal value for this were 284.80. The classification report and confusion matrix can be seen in Figure 5.11. We see here that the accuracy increased from 36% to 38%. We also see that the weighted average for the F1-score increased from 33% to 41%. In the confusion matrix we see a shift in the predictions where it now predicts higher classes more frequent. Though it still does not predict the higher classes correctly. The overall optimal hyperparameters can be seen in Table 5.5, which will be used for the committee.

```
Accuracy: 0.3611393692777213
              precision    recall  f1-score   support

           0       0.62      0.63      0.62       469
           1       0.14      0.42      0.21       110
           2       0.07      0.17      0.10        69
           3       0.00      0.00      0.00        64
           4       0.00      0.00      0.00        61
           5       0.00      0.00      0.00        39
           6       0.00      0.00      0.00        62
           7       0.00      0.00      0.00        83
           8       0.00      0.00      0.00        26

    accuracy                           0.36       983
   macro avg       0.09      0.14      0.10       983
weighted avg       0.31      0.36      0.33       983

MSE 7.8585961342828075
RMSE 2.8033187714355297
297 138 34 0 0 0 0 0 0 0
52 46 12 0 0 0 0 0 0 0
30 27 12 0 0 0 0 0 0 0
29 23 12 0 0 0 0 0 0 0
13 27 21 0 0 0 0 0 0 0
7 17 15 0 0 0 0 0 0 0
20 18 24 0 0 0 0 0 0 0
29 23 31 0 0 0 0 0 0 0
5 8 13 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

```
Accuracy: 0.38453713123092575
              precision    recall  f1-score   support

           0       0.83      0.62      0.71       469
           1       0.20      0.42      0.27       110
           2       0.10      0.25      0.14        69
           3       0.09      0.12      0.11        64
           4       0.11      0.13      0.12        61
           5       0.04      0.05      0.05        39
           6       0.19      0.05      0.08        62
           7       0.50      0.04      0.07        83
           8       0.00      0.00      0.00        26

    accuracy                           0.38       983
   macro avg       0.23      0.19      0.17       983
weighted avg       0.49      0.38      0.41       983

MSE 4.068158697863683
RMSE 2.016967698765571
291 104 42 24 4 4 0 0 0 0
28 46 18 12 5 1 0 0 0 0
11 26 17 7 7 1 0 0 0 0
15 22 13 8 4 1 1 0 0 0
5 16 18 9 8 5 0 0 0 0
0 6 8 11 12 2 0 0 0 0
0 8 23 7 11 9 3 1 0 0
0 6 25 9 12 20 8 3 0 0
0 0 6 1 7 6 4 2 0 0
0 0 0 0 0 0 0 0 0 0
```

Figure 5.11: Finetuning of the C-value. (a) Default (b) optimized C-value and ϵ -values.

5.7. Committee

For the committee we trained each model listed in section 5.7 with the optimal hyperparameters found in section 5.5.4. We then made a committee for each combination of the models which yielded the average accuracies listed in Table 5.7.

	HOG SVMC (a)	Haralick SVMC (b)	HOG SVMR (c)	Haralick SVMR (d)
	0.5656	0.4089	0.4025	0.3204
All	0,4226			
a	x	0.4431	0.5120	0.4536
b	0.4431	x	0,4293	0.3981
c	0.5120	0.4293	x	0.3933
d	0.4536	0.3981	0.3933	x
a + b	x	x	0.4363	0.4063
c + d	0.4167	0.3721	x	x

Table 5.7: Average accuracies of the different committee combinations. a, b, c, d on the left column indicate the corresponding model shown the first row. The 'x' indicate that a model was not trained on this combination.

We see the committees with the best average accuracies are the ones containing the HOG SVMC model and the best models contains only the HOG Features. We do see that the committees do not improve the accuracy from any standalone model and the committee with the highest accuracy is 4 percent point lower than that of HOG SVMC alone.

We can compare the classification report and confusion matrix of the best committee and the best standalone model which can be seen in Figure 5.12. On this we see a small difference in the weighted average of the F1-score, with a change from 52% to 51%. On the confusion matrix we see an improvement in the way that the predictions have been centered around the target class. We also see that the committee miss-predict other classes as class 0 less but has also miss-predicted more of class 0. At class 7 we see that for HOG SVM alone it predicted 59 class 7's correct but for the committee this changed to predicting 51 as class 6.

```
Accuracy: 0.5940996948118006
              precision    recall  f1-score   support

           0       0.68      0.97      0.80       490
           1       0.24      0.09      0.14       107
           2       0.33      0.10      0.15        71
           3       0.23      0.13      0.17        60
           4       0.12      0.04      0.06        52
           5       0.25      0.05      0.08        40
           6       0.25      0.23      0.24        56
           7       0.58      0.67      0.62        88
           8       0.64      0.47      0.55        19

    accuracy                           0.59       983
   macro avg       0.37      0.31      0.31       983
weighted avg       0.50      0.59      0.52       983
474 8 3 1 0 2 0 2 0 0
87 10 2 4 1 1 1 1 0 0
38 12 7 4 2 1 2 5 0 0
36 4 3 8 1 0 3 3 2 0
26 4 3 7 2 0 7 3 0 0
10 3 2 4 5 2 9 4 1 0
14 0 0 5 3 1 13 20 0 0
8 0 1 1 2 0 15 59 2 0
2 0 0 1 0 1 2 4 9 0
0 0 0 0 0 0 0 0 0 0
```

```
0.5269582909460834
              precision    recall  f1-score   support

           0       0.77      0.84      0.81       490
           1       0.20      0.21      0.21       107
           2       0.21      0.32      0.26        71
           3       0.27      0.07      0.11        60
           4       0.11      0.10      0.10        52
           5       0.13      0.07      0.10        40
           6       0.21      0.41      0.28        56
           7       0.83      0.22      0.34        88
           8       0.60      0.32      0.41        19

    accuracy                           0.53       983
   macro avg       0.37      0.28      0.29       983
weighted avg       0.55      0.53      0.51       983
412 51 21 2 2 0 2 0 0 0
65 23 10 3 4 1 1 0 0 0
20 14 23 2 6 2 4 0 0 0
18 10 16 4 5 1 6 0 0 0
11 8 17 2 5 3 6 0 0 0
3 6 6 1 10 3 10 0 1 0
4 5 5 1 7 9 23 2 0 0
0 0 9 0 3 3 51 19 3 0
0 0 2 0 2 1 6 2 6 0
0 0 0 0 0 0 0 0 0 0
```

Figure 5.12: Classification reports and confusion matrices for the best standalone model and the best committee. (a) Standalone HOG SVM (b) Best Committee.

Considering +/- 10% to be correct

As mentioned in section 4.7 we also wanted to consider a range of 1 class, to the target class, to be correct and see if this made a significant difference. The result of this can be seen in Table 5.8 where we see an increase in the HOG SVMC, which were the best standalone model, from 56% to 75% indicating that some of the miss-predictions were within one class of the target class. For the previous best committee which were the combination of the two HOG models is now increased from 51% to 76% making its accuracy better than that of HOG SVMC alone. We see in the confusion matrix for HOG SVM shown in Figure 5.12(a) that most of this increase is from class 1 being miss-classified as class 0 which consist of 87 classifications there are added to be a correct prediction.

	HOG SVMC (a)	Haralick SVMC (b)	HOG SVMR (c)	Haralick SVMR (d)
	0.7503	0.6172	0.7230	0.6210
All	0.6956			
a	x	0.6401	0.7633	0.7018
b	0.6401	x	0.6544	0.5987
c	0.7633	0.6544	x	0.6679
d	0.7018	0.5987	0.6679	x
a + b	x	x	0.6992	0.6809
c + d	0.7396	0.6727	x	x

Table 5.8: Average accuracies (+/- 10%) of the different committee combinations. a, b, c, d on the left column indicate the corresponding model shown the first row. The 'x' indicate that a model was not trained on this combination.

5.8. Practical validation for the year 2019

In section 4.7 we discussed validating our model using a single separate year as validation data. In Table 5.9 we see the result of using the committee and validating it on 2019 data. The training data is all other years than 2019. We see that we achieved the highest average accuracy by using SVMC trained by using HOG features which achieves 43%. We see that all combinations which includes

SVMC on HOG performs higher than the ones without. It is also worth noting that the combination of SVMR models seems to perform well with an average accuracy of 38%. The classification report of the best performing model can be seen in Figure 5.13. It is worth noting that it is unsuccessful in predicting class 1. These results are going to be discussed further in section 6.1.6.

	HOG SVMC (a)	Haralick SVMC (b)	HOG SVMR (c)	Haralick SVMR (d)
	0.4271	0.1065	0.3755	0.2948
All	0.2029			
a	x	0.1637	0.3610	0.3408
b	0.1637	x	0.1783	0.1648
c	0.3610	0.1783	x	0.3800
d	0.3408	0.1648	0.3800	x
a + b	x	x	0.1917	0.1692
c + d	0.3094	0.1793	x	x

Table 5.9: Average accuracies of the different committee combinations for the Practical validation for the year 2019. a, b, c, d on the left column indicate the corresponding model shown the first row. The 'x' indicate that a model was not trained on this combination.

Accuracy: 0.42713004484304934

	precision	recall	f1-score	support
0	0.47	0.98	0.63	363
1	0.00	0.00	0.00	121
2	0.00	0.00	0.00	97
3	0.07	0.04	0.05	49
4	0.04	0.02	0.02	65
5	0.00	0.00	0.00	45
6	0.58	0.11	0.18	100
7	0.32	0.23	0.27	52
accuracy			0.43	892
macro avg	0.19	0.17	0.15	892
weighted avg	0.28	0.43	0.30	892

```
355  5  0  2  0  0  0  1  0  0
121  0  0  0  0  0  0  0  0  0
 93  4  0  0  0  0  0  0  0  0
 45  1  0  2  0  0  0  1  0  0
 58  2  0  1  1  0  0  3  0  0
 28  2  1  4  3  0  2  5  0  0
 39  6  2 10 16  1 11 15  0  0
 17  0  1  8  8  0  6 12  0  0
  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0
```

Figure 5.13: Optimal HOG values on 2019 as validation year

6. Discussion

In this section we will discuss the results listed in section 4.7 and elaborate on some of the decisions made throughout the process of obtaining these results. We will also discuss parts of the process we wish to have done differently and what effect it would have had if done so. At last, we will discuss what could be done in the future to further enhance our model.

6.1. Discussion of results

6.1.1. Histogram of oriented gradient

In section 4.3.1 we described what cell and block size were in terms of HOG features. In section 5.2 we saw the different result of varying block and cell size. We saw very similar performance for all of the block sizes. However, (16x16-2x2) performed slightly better which is a reason we chose to use (16x16-2x2) for our model. Another reason is the computational cost of extracting HOG features and training the model on them (recall section 4.3.1 – cells). The computational cost increases as the cell and block size decreases. Therefore, it makes sense to continue with (16x16-2x2). In section 5.2 we also tested different image sizes. We decided that the mean size would be the best suited. This is because it achieves the highest average accuracy in all cases but one. The Max image size performs

slightly better when class weights are set to 'balanced' and with a cell and block size of (16x16-2x2). However, we moved away from using class weights as 'balanced', which is further discussed later in this section. Another reason why we chose the mean sized images is that the max sized comes with an extra cost. Since the images is bigger it takes more computational power to extract HOG features from them. The number of HOG descriptors increases which increases the training time of the SVM.

As mentioned in section 5.2.2 we saw the results of setting class weights to 'balanced' and 'None'. This showed that for models with class weights set to 'None' the average accuracies were higher for all cell and block sizes. However, with class weights set to 'balanced' the robustness of the models were slightly better. This robustness comes in form of a slightly better macro average. This make sense since class weights 'balanced' adjusts the model to be inversely proportional to class frequencies. Thus, taking the distribution of the classes into account when training the model and making each class contribute equally to the training of the model. Since the robustness were only found to be slightly better, we chose to prioritize the overall increase in accuracy which we achieved from the class weight being set to 'None'.

Continuing the topic of HOG feature sizes. If we recall Figure 5.2 we notice a pattern. As the cell size increases so does the average accuracy of our model. If we were to continue our work, it would be interesting to see if the pattern continues as we increased the cell size even more. What makes it even more interesting is that as the cell size increases the computational power decreases. Meaning we could possibly achieve a higher accuracy for a lower computational cost.

A reason for why the accuracy seems to increase as we increase the cell size could be the size of lodging area. In their paper *"Histograms of Oriented Gradients for Human Detection"*[14] Dalal and Triggs discussed which cell size and block size that performs the best. They discovered that 6–8-pixel wide cells perform the best when not considering block size. They suggest that there might be a connection to human limbs being six to eight pixels wide in their images. Based on this observation it could be interesting to investigate how many pixels the average lodging area is in our image. This would also allow us to take a more qualified guess on which cell size which would be ideal. This is something further work could investigate.

Another aspect of HOG features we could have explored were, as described in section 4.3.1, the block normalization which is performed when extracting HOG features. During our project we never tested how different normalizations would perform. If we were to try to improve our performance even more, this would be an area in which we could experiment. However, we would not expect a huge difference since *Dalal and Triggs* results showed that the L2 norm performed the best. An additional aspect that we did not explore were different histogram sizes. In 4.3.1 we showed how the histogram is used. This could be a way to improve the performance of our model even more. However, we chose 9 bins since it was what *Bill and Triggs* found to be the best in their original paper, but it would be better suited for our model. However, choosing a different histogram size or normalization also changes the number of computations needed. This is something to keep in mind.

In section 5.2.4 we showed the result of training an SVMR model on HOG features. We trained it for the optimal values (16x16-2x2) and image size mean which we found for SVMC. We achieved an average accuracy of 34%. This is significantly worse than the SVMC model which achieved

[14] (Dalal, 2005)

accuracies of around 50% or higher. It is worth noticing that SVMR perform regression which is a very different task than classification. We therefore had to come up with a way to compare the two. This is described in section 5.2. This way of comparison means that we lose some information about the prediction of SVMR. If we were to continue our work on SVMR we would like to explore the results that we got from SVMR as a lodging percentage instead of the comparison that we just mentioned. We did not explore it further in this thesis due to a lack of time.

Another aspect we could have investigated further relative to the SVMR for HOG were the investigation of the HOG parameters which we performed for SVMC, where the results to this can be seen in Figure 5.2. This was not performed for SVMR where there could have been a more optimal combination of cell and block sizes specific for SVMR. This was a process omitted to save time and the optimal parameters for SVMC were used instead.

In section 5.3 and 5.4 we explain how we would use data augmentation and data balancing to improve the robustness of our dataset. In section 4.7 we presented the results of the data augmentation. The four methods used seemed to slightly worsen the model's accuracy, and no significant improvement in the prediction distribution were seen. One reason this did not achieve a higher accuracy were that there were still an abundant number of class 0 present in the training data and therefore a bias for class 0. This bias would then only increase when we increase the data. This were accounted for in section 5.4 where the data were balanced in different ways. This did though not lead to an improvement and only worsened the accuracy even further. Another reason overall could be subjective scoring performed by the seed producers, which will be further discussed in section 6.2. This led us to conclude that the data augmentation was unsuccessful, and we chose to not explore the topic further.

6.1.2. Haralick textual features

In section 4.3.2 we explained how we could extract Haralick textual features for image classification. We used these features in section 5.5 where we see the results of training an SVMC and SVMR using them. First, we are going to discuss the SVMC model. We chose to set it to 'balanced' since it would guess class 0 every time if we did not. We would prefer it to guess other things than 0 since it would mean that our model would be more robust. Furthermore, in a real-life scenario, it would not be very useful to have a model which guesses 0 every time. The next step was to settle on the most optimal image size. Here we see that min and mean perform around 10% better than the max size. One reason for this could be that when the images are smaller the texture patterns that we try to capture becomes more concentrated hereby makes it easier for the features to capture them. We ended up choosing the mean image size to continue with, since it performs slightly better. However, an argument could be made that the min image size would be better since it requires less computational power. Since Haralick textual features are efficient in themselves, we chose to prioritize the slight advantage in average accuracy. For the SVMR model we used the mean sized images that we found to be the most accurate for SVMC. SVMR achieved an average accuracy of 34% which is slightly better than the 30% that we achieved with SVMC. As mentioned in the previous section we had to use a method of comparison which can cause a loss of information. However, here we achieve a better result than when using SVMC. We did not investigate this further since we achieved more promising results using SVMC with HOG features and chose to focus on that.

Next, we displayed the results of using the date as a feature. In section 1.1 we described how lodging usually happens after heavy rain and winds. This made us consider using weather data as an additional

feature to train our model. The thought was that the model would be more likely to classify lodging if rain or wind had recently happened. This would require extract the weather data from a database, then correctly use them along with the pictures. Since this would be a significant workload, we tried a different approach, which using the date as a feature. The idea behind using the date as a feature was the longer the grass had grown the more likely lodging would be. In section 7, we comment on the result of this. We tried combining Haralick textual features and the dates. As the date is simply one feature compared to thousands in terms of HOG features it would be insignificant to use here. In figure 5.5.3 we see an improvement in accuracy when using the date as a feature. This suggests that an even bigger improvement could be achieved using weather data, however more research would need to be conducted. Previous studies have also used the date since the crop was planted, which would be a better than our idea. We simply used a naïve approach which was a counter since the beginning of the year. By using the date since it was planted, we would have a more exact measurement. Thus, concluding that using the date seems to improve performance when using together with Haralick textual features.

Another aspect which could improve the performance of the model could be feature selection. This is used in other studies, such as Li et al[15] which uses Haralick textual features applies feature selection. This is something that could help our model in terms of enhancing the performance, improve interpretability and gain insight into which features are most important in terms of our classification model. The performance is enhanced in terms of selection the features which best represent the underlying pattern in the model. Typical feature selections schemes that we could consider is statistical indicators, recursive feature elimination RFE, and the Boruta algorithm. These are all tested by Li et al[16]. They end up using the Boruta algorithm to some success. It is worth noting that this tailored random forest models and might not work very well with our model. It could also be worth considering applying a feature selection scheme to our HOG features since we have a lot of features here. However, we did not find any research where the previously have been done. For further research this might be an area worth looking into.

6.1.3. HOG vs Haralick

In section 5.2 and 5.5 we described the different results of using Haralick and HOG feature to train SVM models. In all the cases the classifier trained on HOG features outperforms the classifier trained on Haralick textual features. In Figure 5.3 we see that the accuracy for HOG-based models are 30%-55% while Table 5.3 shows the Haralick based models range from 20% to 34%. One of the reasons for this difference could be the number of features which is extracted from each method. While Haralick method only extract 13 features HOG features range in the thousands depending on the block and cell size. This would mean that the HOG-based model would have a much more descriptive representation of the data. However, this also comes with cons. A lot more features also mean a much bigger need for computational power and memory when extracting these. More features also means that the model takes longer to train. These are all important factors to consider when choosing a model.

Recall section 1, here we describe why it is relevant to develop a model which can classify lodging and how it would be used. Since the model would ideally be used to classify new lodging, we would

[15] (Li, 2020)
[16] (Li, 2020)

need to extract new features from each of these images. Here It would be computational expensive to extract HOG features compared to Haralick. However, the analysis would most likely be performed by a computer which has the necessary computational power to perform these computations. Therefore, HOG would likely be the superior choice.

6.1.4. Optimization of hyperparameters

In section 5.5.4 we listed and analysed the results for the optimization of the hyperparameters for the four models listed in Table 4.6. By running a grid search we found the optimal values collected in Table 5.5.

We found that the kernel and gamma values mostly were optimal with values 'RBF' and 'scale'. For both Haralick textual features models the kernel function 'linear' and the γ-value 'auto' resulted in an infinite loop while training the model. This indicates that the Features were not linearly separable and were therefore omitted from the grid search. The C- and ϵ-values were optimized on a finite number of possibilities, meaning that these could possibly be improved further by running the grid search multiple times adjusting the possible values each time. The ϵ-value would possibly benefit the most from this since it for HOG Features SVMR repeatedly found the lowest value given in the grid search to be optimal and for Haralick textual features SVMR found the highest value to be optimal. Thus, a further finetuning of these could be beneficial.

We also found that the cost matrixes described in section 4.5.1 had a small effect on the accuracy of the model. The 'step to 4' were identical to that without a cost matrix and the 'distance' worsened the model. We also found that there was no major change in the distribution of the predictions where the confusion matrixes were almost identical but, in some cases, they worsened the results. Thus, we found that both cost matrixes had no positive effect on our model if any effect at all. Therefore, since the result were almost identical and, in some cases, worse, we chose to perform no further investigations for concerning the cost matrixes.

6.1.5. Committee

In section 5.7 we listed the results from training the committees described in section 4.6. We found that the best committee where a combination of HOG SVMC and HOG SVMR which also were the two best models. This committee has some minor advantages with the predictions being more concentrated around the target values. It also stopped predicting some of the higher classes as class 0 but at the same time miss predicted class 0 as higher classes instead. Therefore, have we concluded that the committee did not perform in the way we had hoped and the HOG SVM alone is still our best model.

The reason for the committees not giving a better result could be that all the models used were biased towards class 0 with some correct predictions at higher classes. Furthermore, are the predictions just an average of the predictions of all models which seems to have skewed some of the predictions to surrounding classes. Therefore, it could be advantageous if we had a model which performed well when predicting high and medium LSs, which could skew the committee to predict medium lodging more precisely.

Furthermore, in section 5.7 we presented the possibility to consider predictions laying within a range of 1 class of the target value to be considered correct. This resulted in a substantial increase in the accuracy of the models. We found that even though the accuracy increased this were probably caused

by the models existing bias of miss-classifying other classes as class 0. We saw that class 1 acquired 87 additional correct predictions from this alone, which explains most of the increase in accuracy. By this standard if we had considered the miss-classifications for class 2 as class 0 we would have increased the correct predictions by 38. And a similar pattern exists for a few more classes. A success for this test would be that the increase in accuracy were caused by miss-predictions laying around some of the higher classes, but this was not the case.

6.1.6. Practical validation on the year 2019

In section 5.8 we display the result of validating the model on data from 2019 and training the model on the remaining years. This gives us a practical estimate of how our model would perform if we where to put it to use in real life. Here we achieved an accuracy of 43% when using the optimized hyperparameters of our HOG based model. This would be insufficient in the real-world since the model would simply be too inaccurate to use. It is noteworthy that our model seems to fail to correct guess class 1. Instead, it guesses 121 times on class 0. This could indicate that our model is not able to separate class 0 from class 1 lodging scores. This can be seen in Figure 5.13. We also tested the committee described in section 4.6 on this type of validation. From here it was clear that the most efficient model is still the HOG-based model. However, we also see promising results from the two SVMR based model with a combined accuracy of 38% Unfortunately this is still not good enough to be useful in a real-life setting. There are paths that we could explore to make the model more useful is a real-life setting. Some of these have already been discussed. A list of future work can be found in section 6.3 which gives the reader a nice overview of the discussed possible improvements.

6.2. Discussion of data

When training any type of supervised learning algorithm, the key to an optimal result is the data. The following section is going to discuss our data and what it means for our models. We are also going to discuss ways that we could improve our data and thereby improve our model.

Recall section 3.2. Here we saw a plot of the distribution of the data. We saw an increase in lodging from may till June and a decrease from June till July. As we mentioned this decrease was unexpected. Instead, we would have liked to see an increase as lodging is more likely as the grass grows higher. There are several reasons why this might happen. One could be that due to different weather conditions one year the grass height could peak earlier than other years. Another reason could be the subjectiveness of the data. One person might have judged the lodging to be 50% in June and then another person judged it to be 40% in July. This makes it a lot harder for our model to learn the patterns of our dataset, and it could be one of the reasons that we do not achieve a higher accuracy.

To improve the data set it might be an advantage to annotate the pictures. This annotation would only be done by one person, which would make the dataset more objective. By annotating the pictures, marking where the lodging is present in each crop image, the model would know the exact pixels which is considered lodged. This would have improved the model since it could link each HOG feature to whether it showed lodging or not instead of the current model which link all HOG features from a plot image to a single percentage. This could then allow us to perform image tiling on the plot images, where each plot image is sliced into tiles, which would have their own LS derived from the annotation. By doing this it could possibly improve the model since it would have n (number of tiles) more images to perform the training on.

Using these tiles with the augmentation explained in section 3.2, where instead of just the image is mirrored and flipped, each tile would be mirrored and flipped. This would then, create four times the number of tiles pieces of data the model could use in its training. This would not cost extra processing power when extracting the HOG features since the number of pixels does not change. The training of the model would neither change, since each tile would have the number of HOG features of a whole plot image divided by n, which would then be done for the number of tiles n. It could be slightly more expensive since each tile would have its own LS. This could possibly lead to better results for the data augmentation than the results we arrived at in our thesis.

In section 3.1 we mentioned that there were different flight heights when the drones took pictures. This led to different pixel densities which is a problem when we are extracting features from the pictures. We solved this problem be resizing the pictures to the same size. In section 4 this process is described. As mentioned in section 4 this process leads to a loss of information. This loss of information could also be a reason that our model does not perform better. However, the loss of information is limited and is therefore not a critical factor.

In section 4 it is described how we resize to the mean size of all the pictures. Instead of resizing to the mean of all the pictures. A different approach could be to find the size of the median image and resize all images to that size. This would mean that we would have to resize a minimum of pictures and thereby have a minimum loss of information. Again, it is worth noting that this loss of information is limited as its effect on our model is limited.

6.3. Future work

As mentioned earlier in this section there are several ways, we suggest we could improve our model. Some of these suggestions are listed below, as well as some new ideas which has not been discussed so far.

- Bigger Cell size for HOG features.
- Investigate the average number of pixels in lodging area to take a more coalified guess on optimal Cell and Block size for HOG features.
- Investigation in the effect on using different histogram sizes and different normalizations for HOG features.
- Investigation for optimal Cell and Block sizes for SVMR.
- A further utilization of the found percentages from SVMR models.
- A possible utilization of weather data in the training of the models.
- Feature selection for both HOG and Haralick textual features.
- Further finetuning of variable hyperparameters. (C and ϵ)
- Annotation of lodging of plot images to improve the model's knowledge of the exact location of lodging, and generally decrease the bias in the data.
- Using the median of image sizes instead of mean image size to further reduce the loss information due to the resize of the image plots.
- In recent years Neural networks have evolved and been used extensively for image recognition problems. Here, convolutional networks have been used with great success. A Neural network model could be an alternative solution for our problem. However, it is worth noting that it might suffer from some of the same bias and limitations as our current model does.

- In section 1 we stated that we wanted to use a classification range of ten. However, as we discussed in section 6.1.6 we have not achieved a result that can be used in a practical real-life setting. A simple way to improve the performance of our model could be to simple decrease the number of classes to five. This would naturally give our model a wider range of error while still be more fine scale than a range of three. However, we would still expect to have the same bias in our dataset. This bias being that our dataset is a lot better at predicting class 0 than the rest.

7. Conclusion

In conclusion, this bachelor thesis developed an objective image-based model for assessing fine-scale lodging severity of seed grass in fields with a range of 10 classes. Each class being 10%. We tested different combinations of image features and machine learning algorithms. The models were trained and validated on data supplied by the department of agriculture of Aarhus university, which were extracted from images taken by an Unmanned Arial Vehicle. Afterwards we processed the images by rotating and cropping them such that they could be used by the image descriptors used to extract the features used by the machine learning algorithms in use. Then through optimizing the hyperparameters of both the image descriptors and machine learning algorithms we found that the optimal model developed made use of Histogram of Oriented Gradients to extract image features used with a Support Vector Machine Classifier machine learning algorithm. However, despite our efforts, the accuracy achieved by the models was found to be subpar, reaching only 58%. This outcome indicates that there is further research needed to improve the model's ability to accurately identify and classify lodging severity in seed grass fields for the fine-scaled lodging severity presented in this thesis. We finally tested our models practical use by training and validating on data from separate years mimicking the practical use of our model. This resulted in an accuracy of 43%, indicating further room for improvement.

8. Referencer

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning.* Springer.

Dalal, T. &. (25. July 2005). Histograms of Oriented Gradients for Human Detection. *IEEE*.

Li, L. L. (24. November 2020). A UAV-based framework for crop lodging assessment. *European Journal of Agronomy*, s. 10.

Löfstedt, T. A. (Febuary 2019). Gray-level invariant Haralick texture features. *PLOS ONE*, s. 18.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . P. (2011). Scikit-learn: Machine Learning in Python. 2825−2830.

ROBERT M. HARALICK, K. S. (6. NOVEMBER 1973). Textural Features for Image Classification. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS,*.

Tan, S. (5. July 2021). Assessment of grass lodging using texture and canopy height distribution . *Agricultural and Forest Meteorology*, s. 13.

Tan, S., Mortensen, A. K., Ma, X., Boelt, B., & Gislum, R. (15. October 2021). Assessment of grass lodging using texture and canopy height distribution. *Agricultural and Forest Meteorology*.

Wilke, S. K. (3. March 2019). Quantifying Lodging Percentage and Lodging Severity Using a UAV-Based Canopy Height Model Combined with an Objective Threshold Approach. *remote sensing*, s. 18.

Woods, R. C. (2018). *Digital Image Processing.* Pearson.