HAN_UNIVERSITY
OF APPLIED SCIENCES

Final Machine Learning report

JARRIK WEG 655958
RENE POSTEMA 645083
KASPER SMEEHUIJZEN 649126
BOAZ BREIMER 2116459

MINOR: EMBEDDED VISION AND MACHINE LEARNING
GROUP: 3
DATE: 10-01-2023

By submitting this portfolio the authors certify that this is their original work, and they have cited all the referenced materials properly.

Contents

1 LIST OF FIGURES

2 INTRODUCTION

This portfolio describes the preparations, approach and outcomes of a Machine Learning (ML) project within the Computer Vision and Machine Learning for Embedded systems minor. It is divided into two parts. The first part deals with the problem statement and data acquisition. The second part is about training, testing and deploying a chosen ML model.

In Computer Vision systems for embedded systems, classification is an important task. A self-driving car, will have to correctly classify many objects in its environment and then act on them differently. Think of traffic signs or pedestrians. Because real world conditions are often very different, there are many variables. It would be impossible to feed all the variations in situations as examples to a system. This is where ML offers a solution. Using data to train a model that uses features to classify objects in images, it is possible to provide the car with the correct interpretation of the situation, without the need for endless examples and large amount of computing times.

In this minor we focus on a main project in which we create a system that scans truck number plates and assigns the loading/unloading location based on the license plate number.

In the minor project we focus on ML classification of numbers. This is directly related to the main project. Number plates contain characters including digits, and we will eventually have to apply a similar model for classifying those characters.

With this project, we want to learn how to efficiently create, label and define features for a dataset itself. In addition, we will learn about model selection, training and deploying a model. This contributes directly to the main project, but also to the knowledge base for other ML and vision projects.

3 PROBLEM STATEMENT

3.1 Objective

The project objective is to classify hand written numbers ranging from 0-9, on a A4 sized paper, as their correct numeric representation while using a simple webcam. A minimum of 80% of the samples should be classified correctly (precision). The recall should also be 80%. The following figure contains the steps that need to be taken. Every step is then briefly discussed.

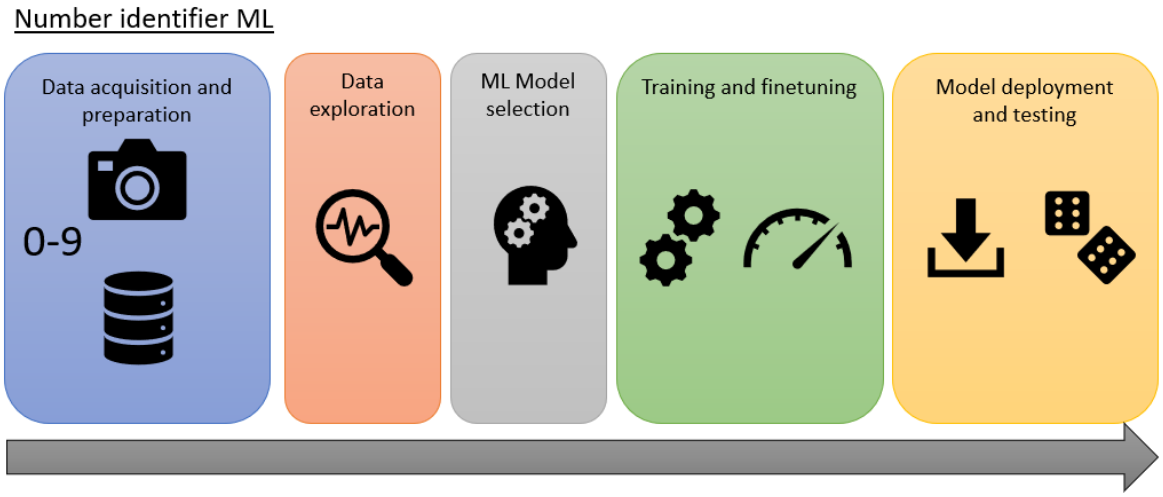


Figure 1 number identifier for ML

Data acquisition and preparation
In this step the data is generated, labeled and stored for further use. Image preprocessing steps are also part of this step and consist of segmentation and feature extraction steps.

Data exploration
The generated dataset of labels and features is explored to get an understanding of the data and assess the usefulness and selection of features.

ML model selection
The best classification model is selected in this step.

Training and finetuning
The training and finetuning of hyperparameters is done in this step.

Model testing and deployment
The trained model is tested with the test set. The model is then deployed in order to classify numbers.

3.2 Requirements

1.1. Diverse handwriting styles should be used for writing the numbers on a single A4. Sampling size should be > 80 For every label (0-9).

1.2. The camera should make > 80 coloured 640 x 480 image samples per label, each sample clearly showing the number on the A4 against a neutral (white) background in identical light

conditions under an angle of 80 – 100 degrees. The sample should be stored in a folder that has the correct label (number) as its name.

1.3. Captured images must be cleaned from unnecessary information in order to optimize feature recognition and filtering. These steps should be implemented in a segmentation pipeline.

1.4. A Pipeline for the segmentation, feature extraction and creation of an labelled dataset should be made in order to repeat the steps without data leakage and processing errors.

1.5. The data should be cleaned from outliers and NULL values before model training.

1.6. The exploration should give insights into the usefulness of the features. This will result in the selection of the best features, the creation of a test and train set and if deemed necessary the creation or scaling of features should be done.

1.7. The selected model should have a performance F score of > 0.8 since recall and precision is equally important in this use case. Over- and underfitting should be eliminated through correct feature selection.

4 DATA ACQUISITION AND EXPLORATION

In order to describe the data acquisition and exploration steps Jupyter Notebooks where used. In the notebooks code and explanatory text is mixed. The results from the Notebooks are copied to this report.

4.1 Collect and label set

For the collection and labeling of the dataset we used a laptop webcam and a white backdrop. The numbers were written on a piece of paper with a permanent marker.

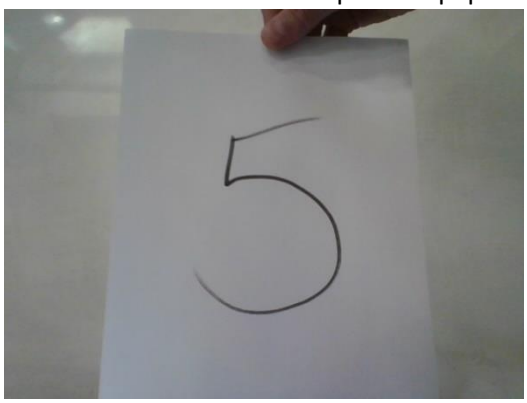


Figure 2: Example of image

To collect the samples, the aquire.py script was used with a modification to the label settings. The custom script we used is called AcquireNumbers.py (annex 1).

4.2 Segmentation and feature selection

4.2.1 Segmentation

As part of the pre-processing the images had to be segmented. The first step is to segment the image. The goal is to use a smaller area of the image where the figure is located and then clarify the figure.

The following segmentation actions were chosen:

1. Cropping the image to remove the hand that is often visible at the top. At the bottom also remove the empty space, this will reduce the number of edges.
2. For better contrast between the number and the background the image should be converted to gray scale.
3. Blur the image so that the lines have a less pronounced transition. This will make the edges of the paper less noticeable as edges and disappear with further manipulation. After some testing a 9x9 Gaussian filter was found adequate for this.
4. Turn the photo into a binary threshold so that the number has a white color. Because of the light condition and the slight bending of the paper an adaptive threshold was used. A local average is used to balance out the effects of oversaturation.
5. Thicken the lines of the figure with a dilate function. This clarifies the contours for contour recognition. For this an morphology dilate function was used with a 8x8 Ellipse kernel. The Ellipse shape gives a smooth finish around the edges, and the size thickens the number line to an easy to recognizable size.

To form a feature extraction pipeline, all these actions have been put into a function together with the feature extraction steps (annex 2, 3 and 4).

4.2.2 Feature selection

The important part of this step is the selection of usable Features. The classification end goal and the shape that will be used as input were taken into consideration for the features. The contour of the number would be the starting point of most features. So contour related features where selected for the first iteration[1]. The following contours and the rationale behind them:

Contour Area: The area of a contour is the pixel area of the closed contour. Because of different sizes between for example a 1 or an 8, the area can be quite discriminative and informative.

Contour perimeter (perimeter): The perimeter of a contour is the length of its closed border. It is related to the area but suffers less from closed circles inside numbers. It can be distinctive between a `0` or e.g. `5`, a `5` has (possibly) a longer perimeter.

Area bounding box to contour: The area of a drawn bounding box around a contour. This feature value is used to derive other features and to center the contour inside a fixed frame for a normalized centroid. This is done because not all numbers are placed centered in front of the camera.

[1] If these features are not useable more iterations of feature selection should be applied.

Extend: the contour Area divided by the bounding box area. This value is a form of normalization of the extremes of a contour. A `7` is more likely to give a higher value than a `1`. While the bounding box might still be similar.

Aspect ratio bounding box: The extent to which length and width of a figure relate can be determined with the aspect ratio. Wide and high numbers might have a constant aspect ratio.

Hu moments of digit contour (7 values): Hu moments are independent of rotation and scaling of the number contour. This is important because the digits are not all displayed equally `straight' and scaled in front of the camera. This feature set might be the most informative and discriminating feature.

Normalized centroid of the digit contour: The centroid is the calculated center of the contourarea. Similair contours should have a cntroid position that is in close proximity to similair contours. Take two 5 and the centroid should be near the same relative position. Because the position of each number can be different on the images, the images are centred in a new frame based on their contour bounding box. then the centroid is calculated. This produces the normalised centroid.

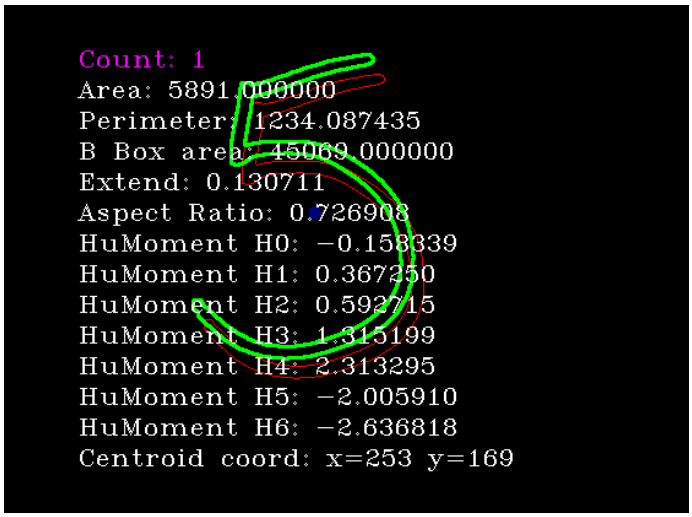


Figure 3 Visual example of the feature set. Count is giving an indication of how many contours where found and is used as a control measure. The green five is the centered contour (to a fixed frame). The blue dot is the centroid of the green five. The red five is the old position.

4.2.3 Data exploration

The generated data set has been saved for further use. For the exploration the data was loaded into a Pandas dataframe (annex 5). An information and statistical exploration has been done. With the information exploration the total amount of entries were inspected. A good outcome was that no Null values where found. The statistics exploration shows statistical information on all numeric data columns.

	area	perimeter	aspect_ratio	extent	HuM1	HuM2	HuM3	HuM4	HuM5	HuM6	HuM7	centroidX	centroidY
count	1486.000000	1486.000000	1486.000000	1486.000000	1486.000000	1486.000000	1486.000000	1486.000000	1486.000000	1486.000000	1486.000000	1486.000000	1486.000000
mean	9719.770188	1036.126611	0.570366	0.261217	2.264845	5.052031	8.179772	8.779424	6.580872	4.960587	-7.139777	248.357335	159.652086
std	9731.617864	334.311760	0.184976	0.204766	0.109370	0.481112	0.996356	1.049445	16.299550	10.500635	16.333913	10.776119	17.396101
min	95.500000	37.899495	0.048485	0.062186	1.987535	4.022682	6.497614	7.035828	-27.379021	-15.838197	-24.046052	207.000000	116.000000
25%	4764.000000	845.769114	0.511135	0.123997	2.203765	4.762632	7.371849	7.934874	-16.112106	-10.345165	-17.633533	242.000000	149.000000
50%	6050.500000	1035.537622	0.576000	0.157277	2.268617	5.021885	8.001955	8.575241	15.714053	10.624276	-16.072614	245.000000	164.000000
75%	12561.375000	1251.971909	0.695951	0.358405	2.317576	5.324166	8.641550	9.328074	17.411938	11.683900	16.346929	256.000000	169.000000
max	56966.500000	2035.405320	1.466667	0.925737	3.198296	8.279983	11.696299	14.317746	25.124239	18.530492	27.652495	277.000000	211.000000

Figure 4 statistical information of dataset

Column by column the following conclusions have been made:

- *area* has a large max value, this might be an outlier.
- *perimeter* at a glance the values look ok.
- *aspect ratio* with a low std like this, the values might lay close to each other resulting in great overlap between labels. it might not be discriminating enough.
- *extend* at a glance the values look ok.
- *Hu moments* at a glance this looks ok. These should be plotted to see a better relationship.
- *Centroids* the values look like they are close toghether, but they might be disciminating enough.

Up to this point the conclusions were paused, and a test/train set was created. The target labels have been encoded using the LabelEncoder. After this a train and test split has been made with a 75% train 25% test split.

Now it was possible to check the *area* feature. By taking a sub set with a range of the (max value / 2) to the max value a good outlier area could be investigated. Plotting this in a stacked histogram results in the following chart:

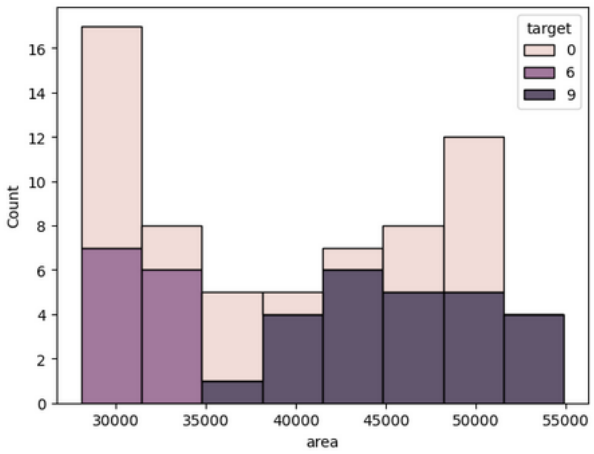


Figure 5 area outliers distribution over target labels

The conclusions from the chart:

6, 0 and 9 are among the 94 outliers. By their nature these will have large areas when filling up their contours. The absence of 8's in this data sample is curious. It should be expected that an 8 has the same contour effect on it's filled contour area as a 6, 0 or 9. This might imply the data is not uniform

in its input. The different numbers have different sizes, because no fixed distance from the camera was used. The effective boundary of the numbers was the paper size (A4) where they were drawn on.

After consulting the dataset to test this hypotisis, the findings where that there are a substantial number of `0` and `9` that are using the larger part of the A4 and are close to the camera. Whereas the `8` are more uniform on the images.

This means the input images are causing the large *area* distribution.

Possible solutions:

- Not use the *area* feature in the model.
- scaling the data using standard scaling
- Delete the outliers from the data set.
- Recapture the images with a fixed distance to the camera.

Deleting the rows on this small dataset is not productive, the data is needed for training. Recapturing the images is possible but time consuming.

The non-fixed distance is part of our use case challenge. Before excluding the *area* from the features. standard scaling should be tried.

A standard scalar function has been applied to the data, with the exclusion of the target labels. We don't want to scale these. The labels were added to the set again after the scalar pipeline.

	area	perimeter	aspect_ratio	extent	HuM1	HuM2	HuM3	HuM4	HuM5	HuM6	HuM7	centroidX	centroidY	target
0	3.754558	-0.282242	-0.308558	3.012062	-0.657483	-0.214445	1.883829	1.157703	0.825527	0.722017	1.659855	-0.503351	0.206884	9
1	2.160343	-0.485321	0.167667	1.520791	0.431228	0.918482	-0.145603	2.656518	-1.744412	0.893749	-0.901291	-1.160277	1.554309	6
2	-0.156820	1.909959	0.625398	-0.691658	-1.238141	-0.399684	0.174131	-0.273307	0.622369	0.617467	1.496824	-0.878737	-0.086035	1
3	-0.284302	1.226617	0.826890	-0.697871	-1.134817	1.864266	-0.147999	-0.475043	-1.445397	0.606769	1.474456	-0.409505	-0.261786	9
4	-0.228535	1.443495	1.310921	-0.781762	-1.782962	3.737957	-0.989861	-1.087189	-1.360722	0.842440	-0.537892	-0.784891	-0.671872	9

Figure 6 The sample of the first 5 rows in the training set

4.2.4 Feature exploration

Graphical representations were made to try and get an understanding of the features graphical representations (annex 7). The first one is a correlation matrix of the features (Figure 7 correlation matrix of all features). Features that have a correlation above 0.8 or below -0.8 should not be combined (one would be dropped). The '*area'* and '*extend'* are 0.72 and close to 0.8, in this case the '*extend'* will not be used. '*HuM3'* and '*HuM4'* have a correlation of 0.86 and should not be used. The variance of all features was also checked. They are all above 1. They should not be close to 0 to be considered a useful feature, because a low variance means the feature is not discriminating enough over the different targets. No features should be dropped because of the variance.

Histograms where plotted to get a better understanding of the features relation to the targets (Figure 8 Histogram of all Hu moments and the targets, Figure 9 Histogram of all, but the Hu moments.). '*HuM2'* seems the best candidate from the Hu moments. The '*area'*, '*perimeter'* , '*aspect_ratio'* and *centroids* seem to have a meaningful distribution on the targets. But it is hard to say.
In order to get a better selection the scikit *SelectKbest* was used to get the top 6 (rounded half of 13

features) features. These 6 features will be used on the first iteration of training. Later more can be added if the results are not sufficient. The method used for *SelectKbest* is *'f_classify'*. It calculates the F score of the features and can be used on negative values as well as positive. It is meant to be used in classification use cases.

The result are the 6 features *'area'*, *'perimeter'*, *'aspect_ratio'*, *'HuM2'*, *'centroidX'* and *'centroidY'*. This is in accordance with the speculations made on the histogram outcome. Figure 10 Histogram of the top 6 features showz the relation to the targets. These 6 features will be used in the training of the model. The Kbest feature selection step can be skipped in the pipeline. The 6 features will be selected instead.

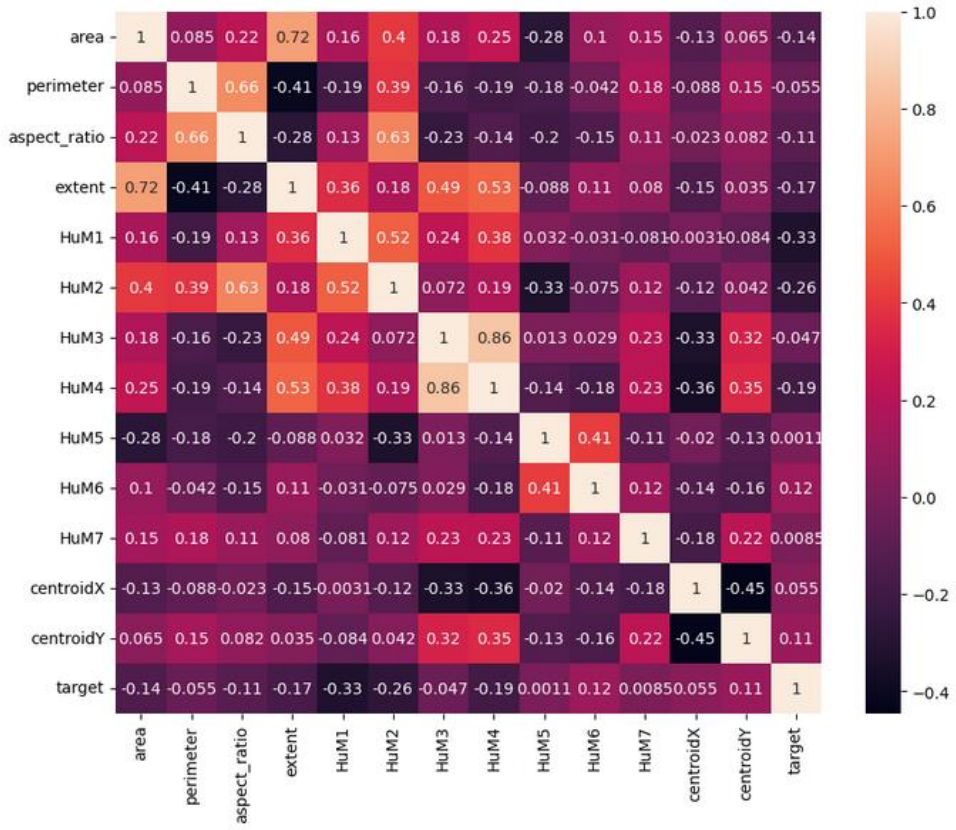


Figure 7 correlation matrix of all features

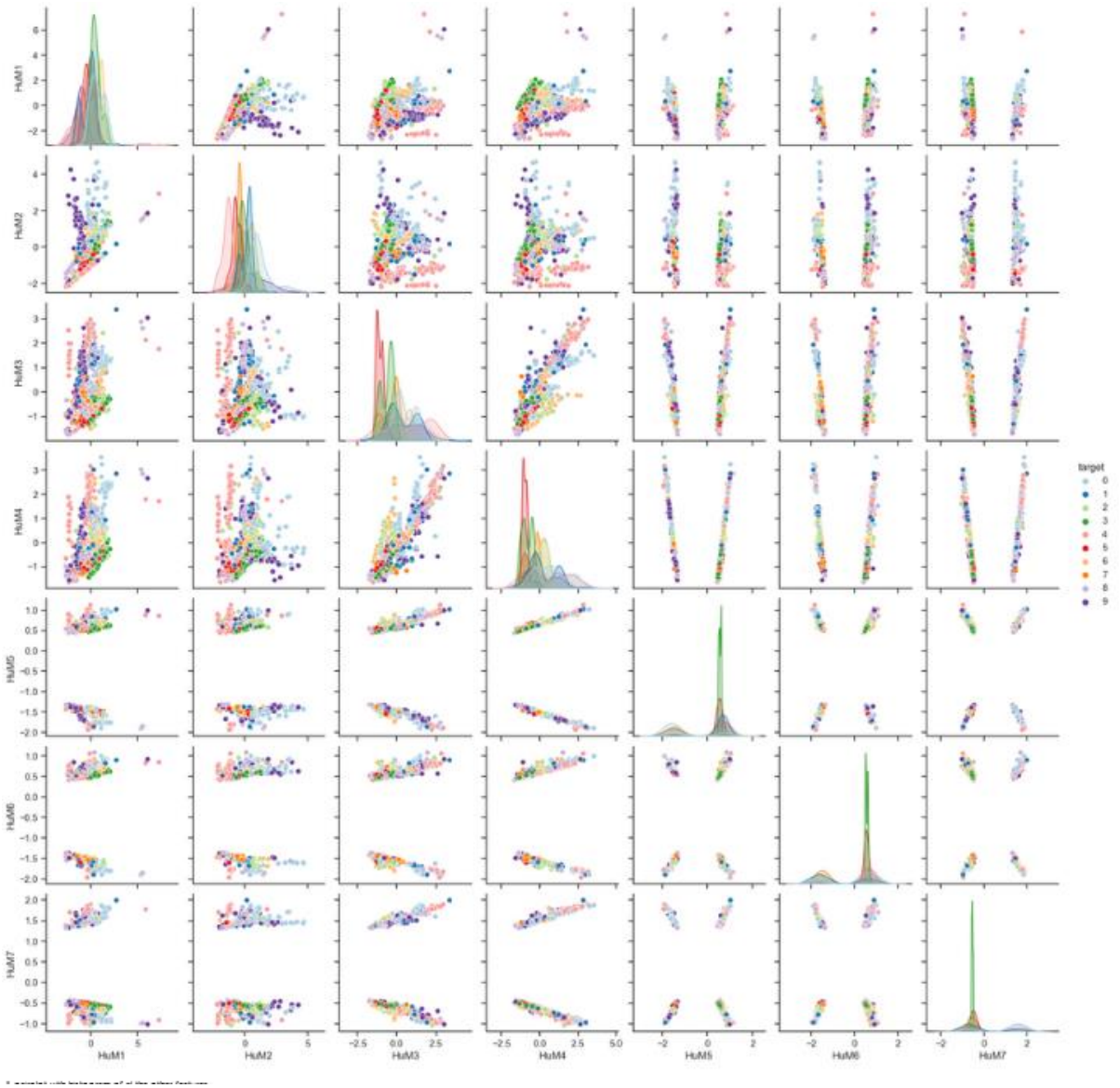


Figure 8 Histogram of all Hu moments and the targets

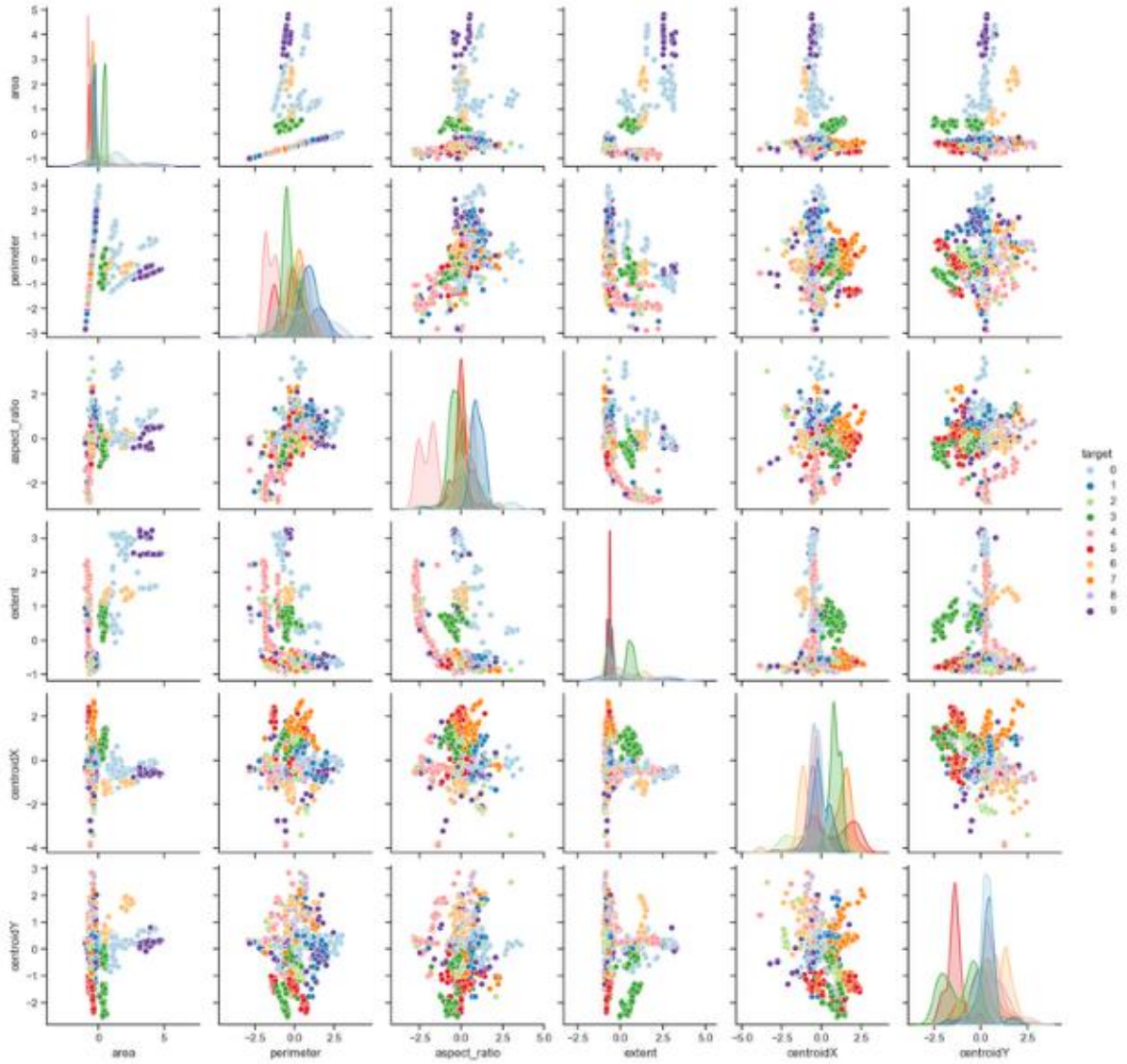


Figure 9 Histogram of all, but the Hu moments.

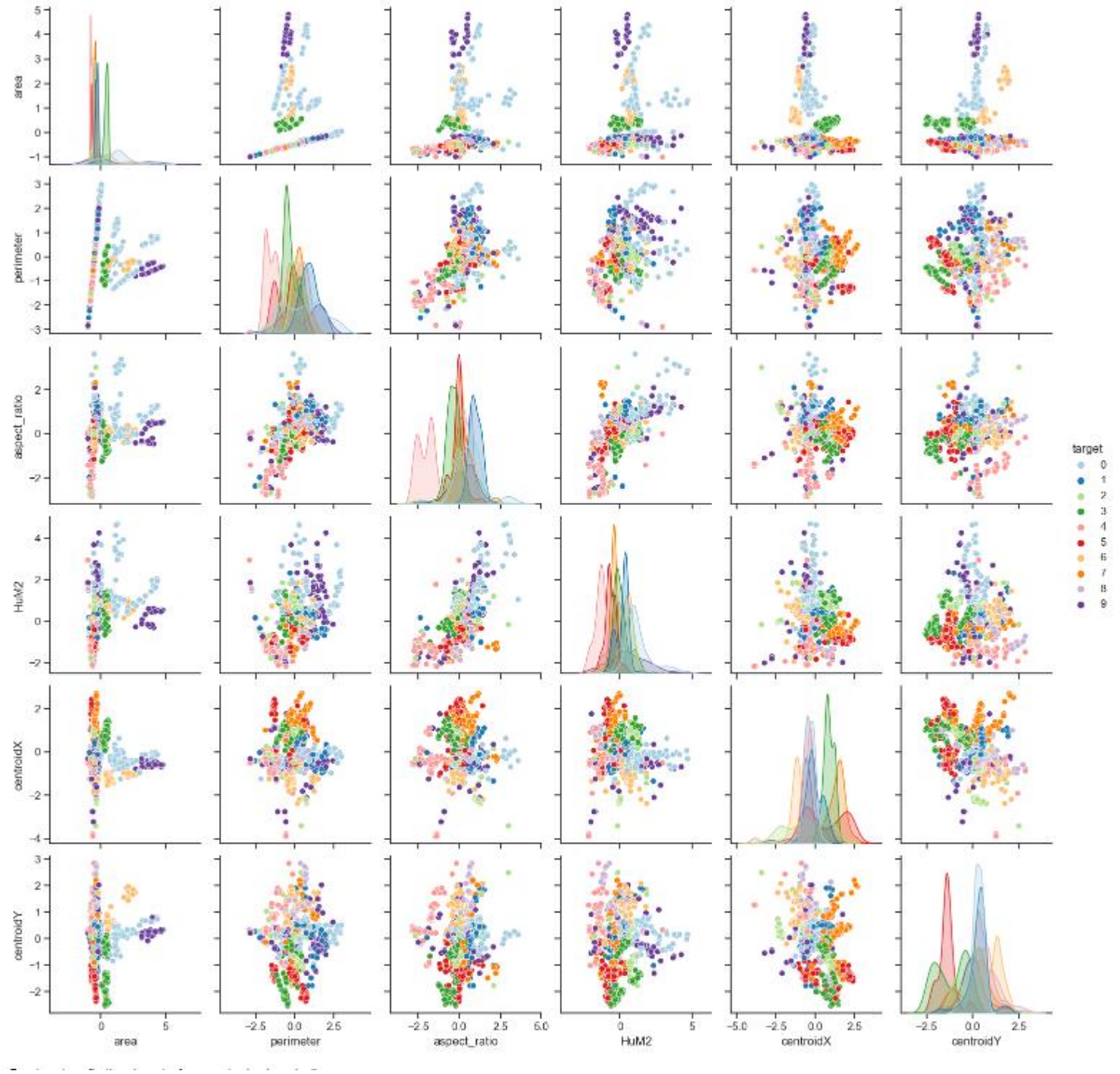


Figure 10 Histogram of the top 6 features

4.2.5 Outlier detection

Following the first observations on outliers we decided to apply outlier removal to enhance the model training. A solution to the outlier detection was chosen. The Local Outlier Factor (LOF) was used (annex 6) after also considering Standard Deviation and Isolation Forest. It is important that the classes are observed as separate clusters in the bigger data set. Using a standard deviation will result in unintentional deletion of labels, that might be a local cluster. Isolation Forest resulted in a to high number of outliers.

LOF will identify an outlier on the basis of its nearest neighbors, density and the distance parameter (Jayaswal, 2020).

The LOF results are a total of 115 outliers spread out as follows:

- target: 9 number of outliers: 9
- target: 6 number of outliers: 11
- target: 8 number of outliers: 62
- target: 0 number of outliers: 22
- target: 5 number of outliers: 5
- target: 1 number of outliers: 2
- target: 4 number of outliers: 1
- target: 2 number of outliers: 3

The figure below shows the PCA dimension reduced plot of the outliers.

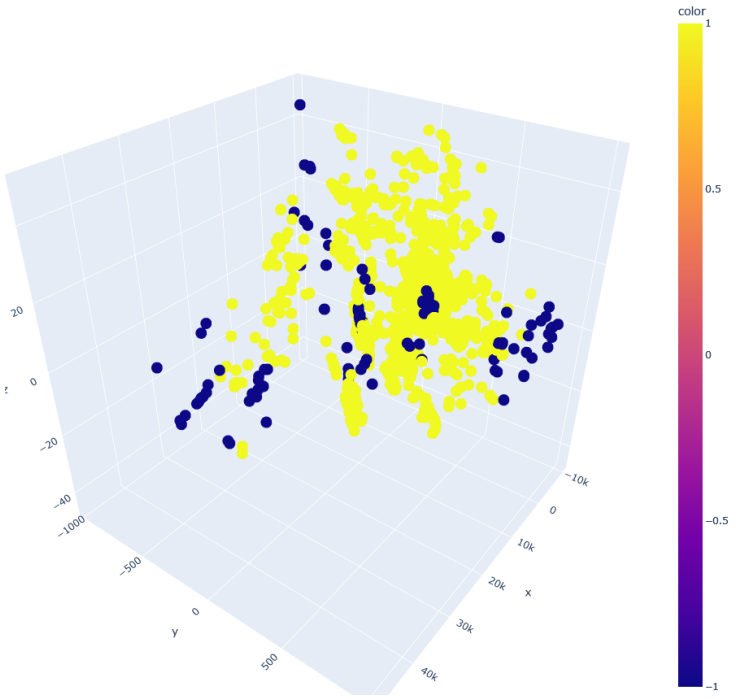


Figure 11 LOF outliers in a dimension reduction PCA graph. Blue are the outliers.

5 FEEDBACK FROM ANOTHER TEAM ON YOUR PRELIMINARY REPORT

This chapter contains the feedback given by the other team and the resulting actions taken to improve the report.

Proposed improvement of the problem definition
Feedback from other team: *Problem statement is defined and specific, the performance is not measured and made practical.* The requirements contain measurable performance metrics that can be made practical during testing or model selection. it is unknown to us what the feedback is based on.

Proposed improvement of the list of requirements
Feedback from other team: *Functional requirements are given, no technical requirements (camera specifications etc).* The list of requirement contain technical specifications as discussed in the report feedback form. More specific camera type information is not an requirement that gives any added value to the system.

Proposed improvement of data collection
Feedback from other team: *Features are engineered, only not bespoken or visible.* The features are visible in figure 3. Most features are numerical features that are a result of calculations made on the contour or area. Therefore, these features can only be represented in a numerical fashion. To illustrate this, figure 3 was made. Features are also discussed in chapter 3.2.2.

Proposed improvement of feature engineering and visualization
Feedback from other team: *Features are visialized and scikit learn is used to get the most relevant features. Note please make the picture readable within reports. The preprocessing pipeline is discussed, only no code sample is given and therefore no review can be performed.* The figures of the histograms are hard to read. This is true. The notebooks will be added as annexes to make it possible to view the figures. Also the code will be present in the notebooks. A pipeline was designed but not included in the preliminary report. The pipeline will be added.

6 MODEL SELECTION, TRAINING AND VALIDATION

We have chosen to try three different classification models and select the best performing model for deployment. Hyperparameter optimalisation, validation and final validation on the test set have been applied to all three models.

The three models and what they do:

- **Support Vector Machine (SVM)**

  SVMs are a type of supervised learning algorithm that can be used for classification. They work by finding the hyperplane in a high-dimensional space that maximally separates the classes. Given an input data set, the algorithm finds the hyperplane that has the largest distance to the nearest training data points of any class (support vectors). New examples are then classified based on which side of the hyperplane they fall.

- **K nearest neighbors (KNN)**

  KNN is a type of supervised learning algorithm that can be used for classification. Given an input data set and a value for K, the algorithm finds the K data points in the training set that are closest (in terms of distance) to the input. The classification of the input is then based on the majority class among these K nearest neighbors.

- **Random Forest (RF)**

  Random forests are an ensemble learning method that can be used for classification. An ensemble is a group of models that work together to make predictions. A random forest consists of a number of decision trees, where each tree is trained on a random subset of the data. During the training process, the algorithm selects a random subset of features for each tree, and the tree is trained to make predictions based on these features. When the random forest is used to make a prediction for a new example, the predictions of all the individual decision trees are combined to give the final prediction.

These three models where chosen because we had to use a supervised learning algorithm that could perform a classification task. These models have a good track record in classification performance. Decision Tree or simpler models would likely underperform anyway, so these where skipped.

6.1 Function to create a common approach

To ensure that we use the same data and approach for all three models we created a custom function called *TrainTestSetFunction.py* that will work as a pipeline (annex 9). The goal is to transform and train the data in the same way by having the data set as input and a train, validation and test sets as an output. The function also performs outlier detection, scaling and label encoding.

We tried to implement all the steps in a Sklearn pipeline, but this was not possible. Implementing an functional outlier removal into a pipeline is not yet feasible (*Feature Request*, z.d.). The pipeline is

only used for scaling, and also added as a component to the function.
At first we used the label encoder from Sklearn, but we found out that it encoded the labels in a confusing way. The encoding results in a number assigned to the `eight` or `one` string labels. Because the samples where ordered in alphabetical order, the eight got the label 1, etc. We mistook them for their real label, i.e. we thought the prediction of 1 meant it really was a `one` but in reality it was the encoded label for `eight`. When checking the model with a single sample from the trainset we found out that this was the problem. To solve this problem we changed the label encoder to a custom function that would take the string and assign the appropriate integer number to it.

6.2 Model training setup

For every model we used a separate Jupyter Notebook with an identical layout following the same steps.

1. Load the dataset from file;
2. Transform the data set using *TrainTestSetFunction.py*;
3. Check the basic (default) model performance;
4. Use Grid search to optimize and finetune hyperparameters;
5. Perform cross validation;
6. Analyze performance metrics: Confusion matrix, F-score and learning curve;
7. Check test set performance with Confusion matrix and F-score.

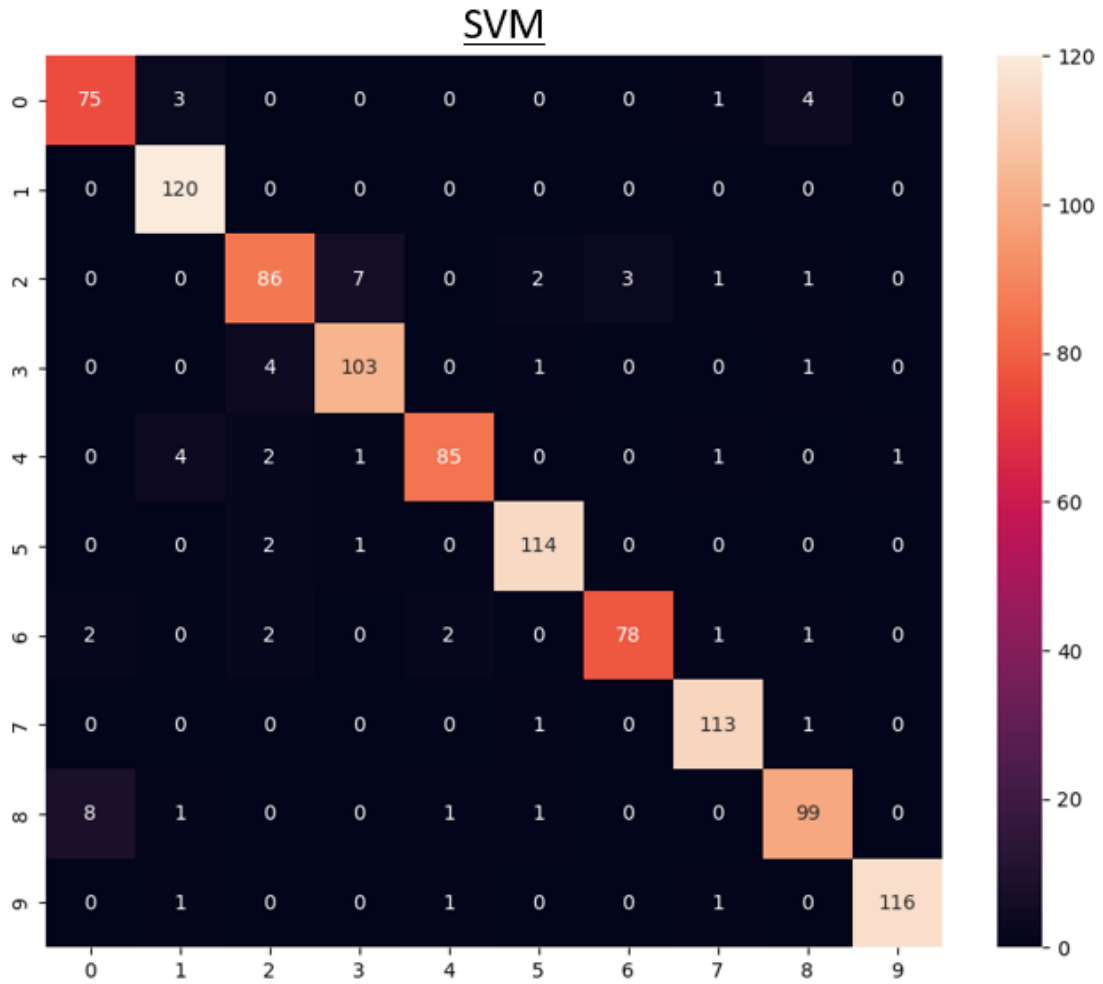
The three Jupyter notebooks contain all the steps, comments and analyses.
SVM: *Training and evaluation SVM (annex 10)*
KNN: *Training and evaluation KNN (annex 11)*
RF: *Training and evaluation RF (annex 12)*

6.3 Model performance and selection

To compare the different models the Confusion matrix, F-scores and Learning curves will be used.

6.3.1 SVM performance

SVM performance is based on the best hyperparameter set that was extracted through Grid search and used in the cross validation.




```
Test set F1 scores
Micro: 0.9392212725546059
Macro: 0.9361035787176768
Weighted: 0.9389507277233778
Per label f1 array: [0.89285714 0.96385542 0.87755102 0.9321267  0.92896175 0.96610169
 0.93413174 0.96995708 0.9124424  0.98305085]
Lowest f1 scoring label: 0.8775510204081632
```

Figure 12 SVM performance. Confusion Matrix and F-scores. Note that rows are the actual classes, columns are the predictions.

Because not all samples are evenly distributed we have to look at the Weighted F-score for overall model performance. The model has a 0.938 performance score. This is more than the 0.8 score set in the requirements. Even the lowest class scores above 0.8.

Some observations and conclusions taken from the Confusion Matrix:

- The `0` has the lowest score on correct classifications. It is often confused for an `8`. This is not strange because the `8` contains two circles that might trigger this classification as features.
- The `1` are always correctly predicted.
- `2` and `3` are sometimes mixed with a higher change of a `2` being classified as a `3` than the other way around. A `3` shares the shape of some `2` in It's shape. This might be the reason of the wrong classification.
- `4`s have a tendency to be mistaken for `1`s. The vertical line in a `4` might be the reason for this.
- `6`s are mistaken for other numbers containing circles or `/` shapes (`2`,`4`, `7`). The `6` is never mistaken for a `9`. This is something I would have expected, because it is the inverted shape.
- Just like the `0` is often mistaken for an `8` the opposite is also true.

SVM learning curves

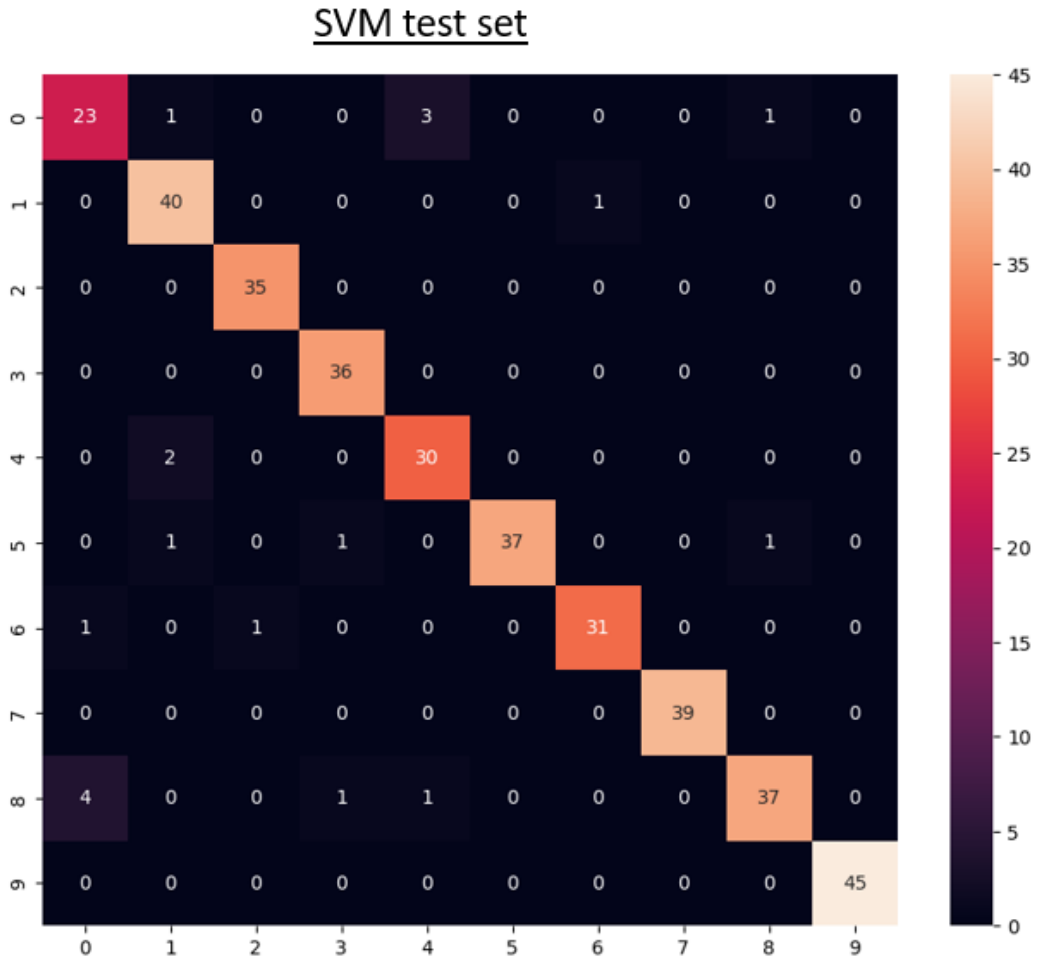
Figure 13 Learning curves for SVM

The learning curve shows how at training size `480` the score starts to rise in smaller increments. At `700` the score is around `0.94`. This is a good accuracy score for the amount of data used in the training. The model might be overfitting because of the gap between the two lines.

The variance of the model is also higher than its bias. This is because the trained model is more sensitive to the variations in the data set size.

Adding more data will not improve the model because it has reached an almost plateau like level.

SVM test set results




```
Test set F1 scores
Micro: 0.9489247311827957
Macro: 0.9447908556313793
Weighted: 0.9487542291205293
Per label f1 array: [0.82142857 0.94117647 0.98591549 0.97297297 0.90909091 0.96103896
 0.95384615 1.         0.90243902 1.        ]
Lowest f1 scoring label: 0.8214285714285714
```

Figure 14 SVM test set performance. Confusion Matrix and F-scores. Note that rows are the actual classes, columns are the predictions.

Considering the Confusion Matrix, the results are not bad, and overlap the training errors.

Some numbers have errors not seen in the training.

- the `0` is mistaken for a `4`. In training this has not occurred.
- the `1` is mistaken once for a `6` once. In training this has not occurred.
- the `5` is mistaken for a `1` and `8`. In training this has not occurred.
- the `8` is mistaken for a `3` once. In training this has not occurred.
- No errors with the `2` and `3` this are a good result and against the expectations.

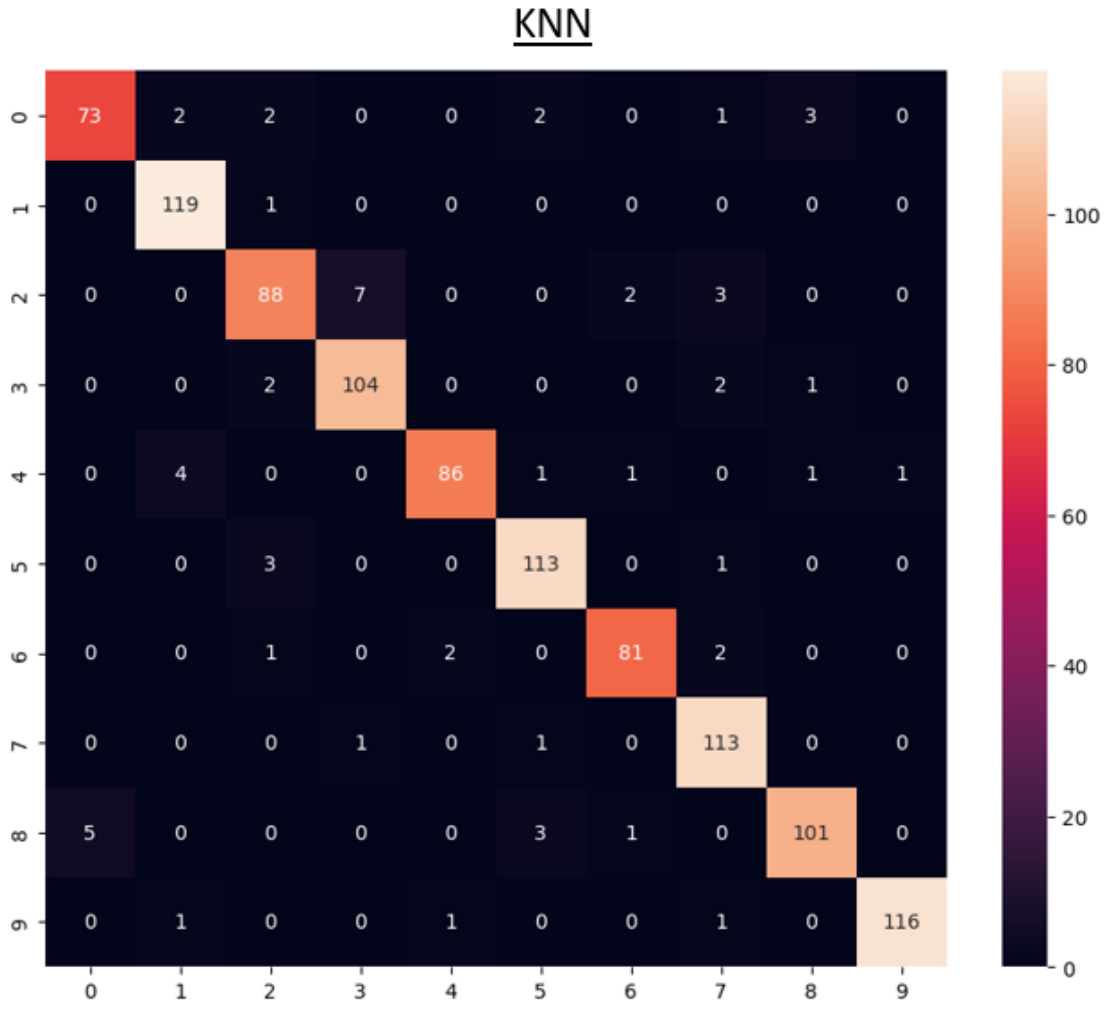
This is not a bad outcome. A reason for the difference in errors on the labels might come from the absence of outlier removal in the test set. The unseen error cases might be part of an outlier in the test set.

The F1 score is good with a weighted score of `0.948` vs. the `0.938` from training. Test performance is slightly better. The model does generalize well. The lowest F score is `0.82` vs. the `0.87` from training. this is still acceptable. The low score in testing belongs to the `0` label. In training the `0` is also error prone.

Overall, the SVM model is a strong performer. Both in training, validation and testing the f-score 0.8 requirement is achieved on all classes. The results from the test set prove that the model generalizes well.

6.3.2 KNN performance

Just as SVM, the KNN performance is based on the best hyperparameter set that was extracted through Grid search and used in the cross validation.

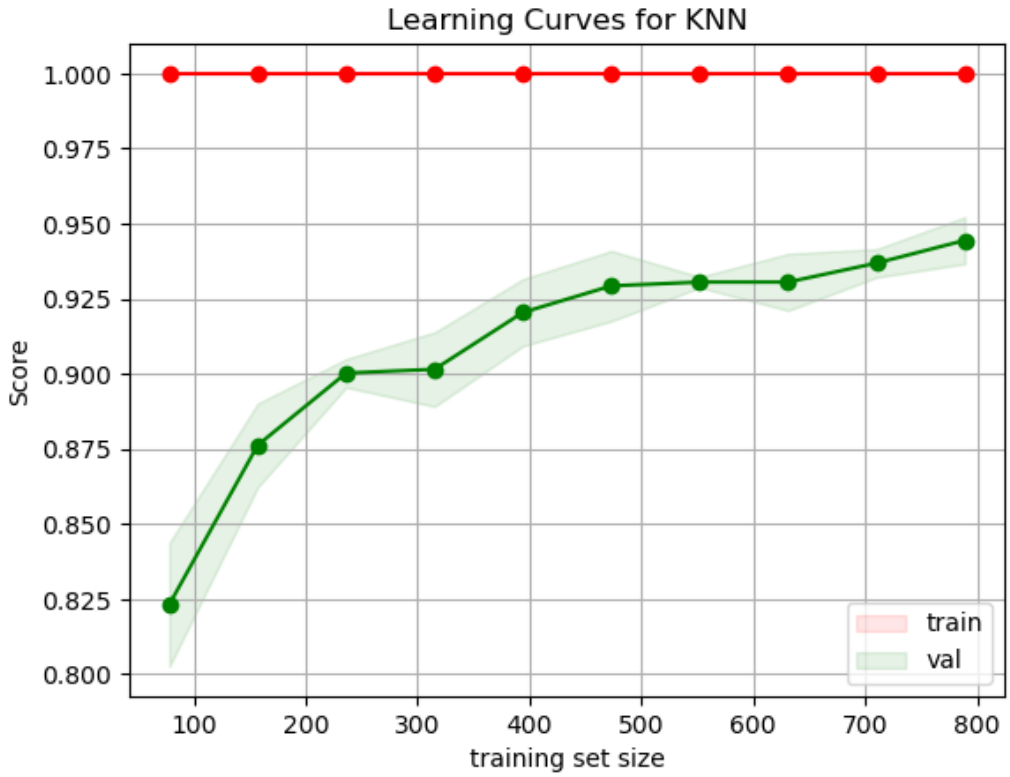



```
Test set F1 scores
Micro: 0.9439696106362773
Macro: 0.941755095265108
Weighted: 0.9437237527708032
Per label f1 array: [0.9068323  0.96747967 0.89340102 0.94117647 0.93989071 0.9535865
 0.94736842 0.94957983 0.93518519 0.98305085]
Lowest f1 scoring label: 0.8934010152284264
```

Figure 15 KNN performance. Confusion Matrix and F-scores. Note that rows are the actual classes, columns are the predictions.

The weighted score of 0.94 is high and slightly outperforms the SVM model. Just like the individual classes it performs above the 0.8 requirement. But the high scores might mean overfitting is happening. When looking at the learning curve the risk of overfitting becomes more evident.

KNN Learning Curves

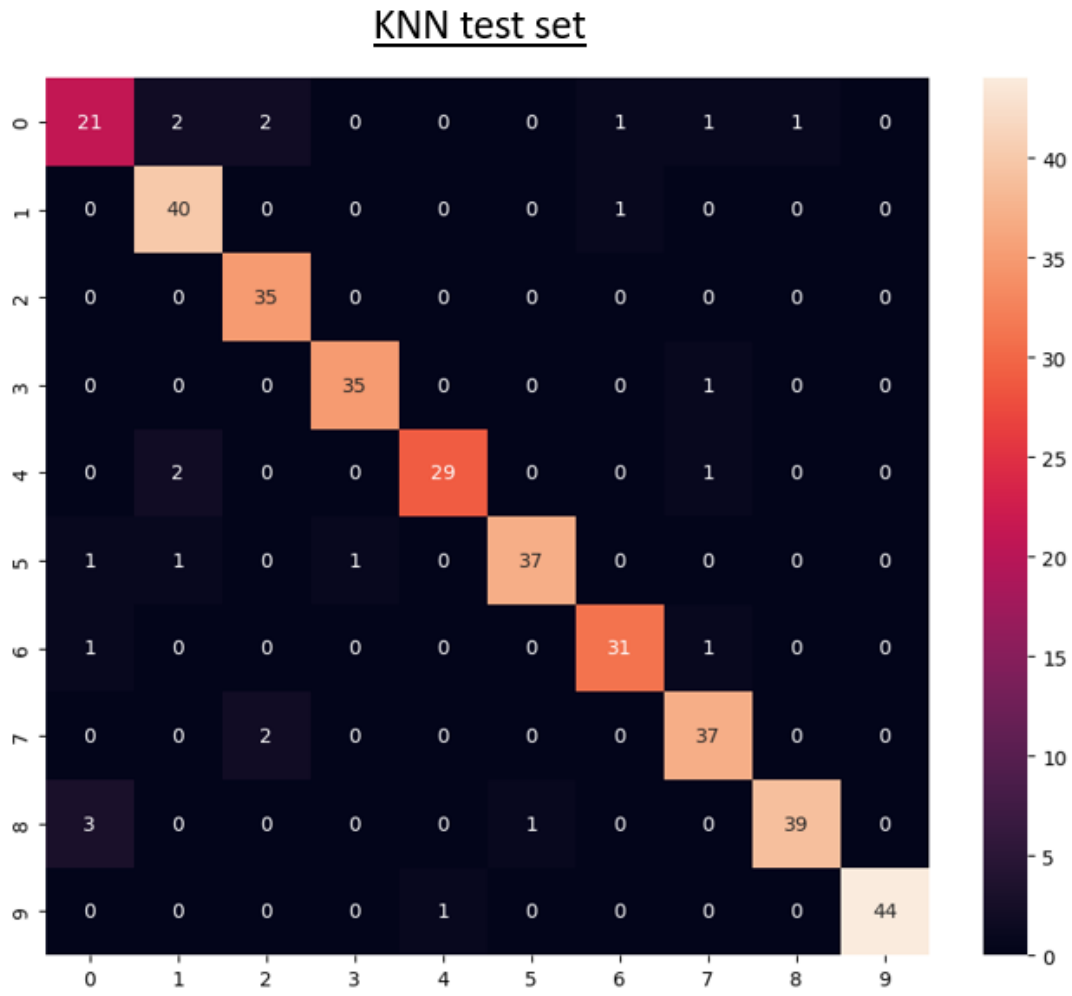


The learning curve shows how at training size `400` the score starts to rise slowly with every `100` set size. At `800` the score is around 0.945. This is a high accuracy score for the amount of data used in the training.

The model might be overfitting because of the large gap between the two lines. It is larger than on the SVM model.

The model has a high variance because it is sensitive to the training data. Adding more data will keep on improving the model slightly. the model is still sensitive to more variations in the training set and has not come to a plateau like state. Adding more data will keep on improving the model slightly, but risks overfitting the model even more. The model seems to continue learning with more samples.

KNN test set results



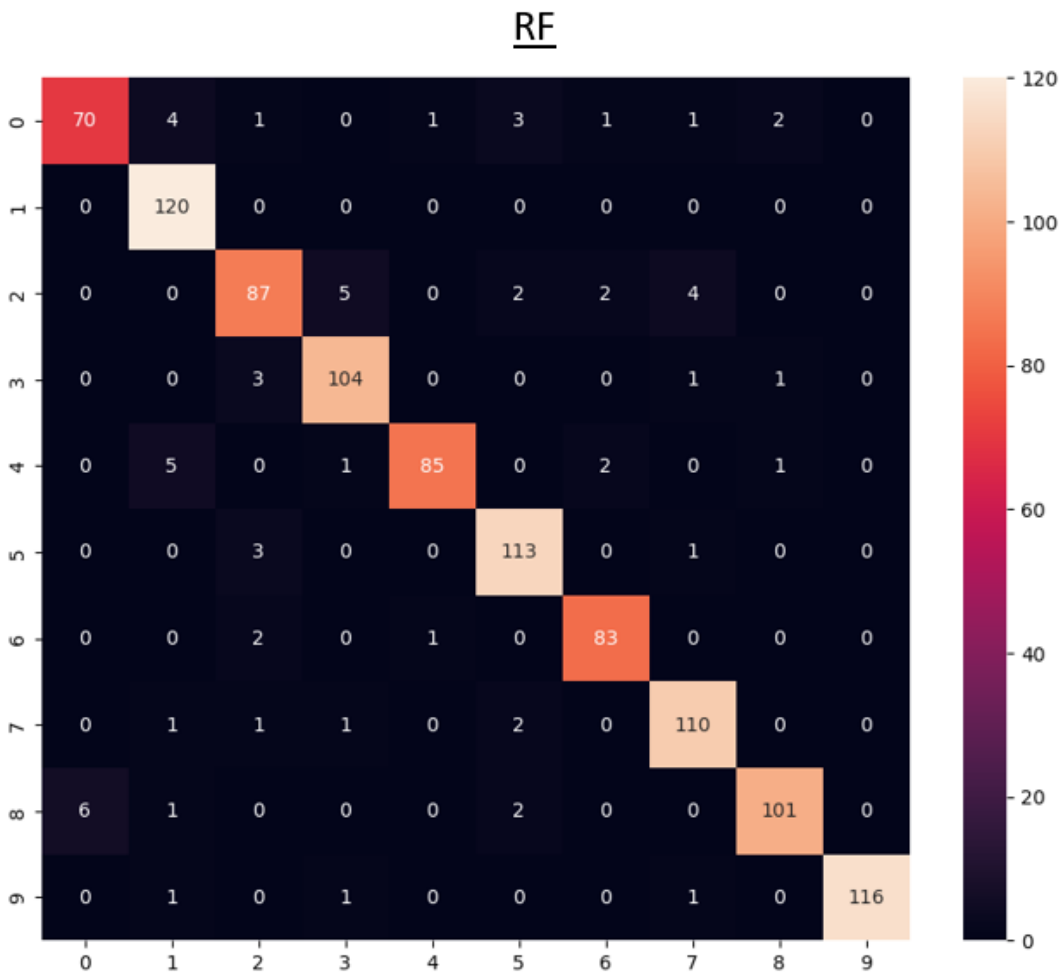

```
Test set F1 scores
Micro: 0.9354838709677419
Macro: 0.9303297344253509
Weighted: 0.9351845246397347
Per label f1 array: [0.77777778 0.93023256 0.94594595 0.97222222 0.93548387 0.94871795
 0.93939394 0.925      0.93975904 0.98876404]
Lowest f1 scoring label: 0.7777777777777779
```

Figure 16 test set KNN performance. Confusion Matrix and F-scores. Note that rows are the actual classes, columns are the predictions.

The KNN model scores worse on the test set than it did on the training set. The `0` class is even underperforming on the 0.8 requirement scoring a 0.777. These scores together with the Learning Curve show the model is overfitting on the training data and cannot generalize as good as the SVM.

6.3.3 RF performance

Just as SVM and KNN, RF performance is based on the best hyperparameter set that was extracted through Grid search and used in the cross validation. Applying these techniques requires a lot of computational power, so the best scores from Grid search where manually copied into the classifier. This means that running the Notebook again might result in different hyperparameters. So caution should be applied when doing that.




```
Test set F1 scores
Micro: 0.9392212725546059
Macro: 0.936714043584899
Weighted: 0.9389014397299114
Per label f1 array: [0.88050314 0.95238095 0.88324873 0.94117647 0.93922652 0.94560669
 0.95402299 0.94420601 0.93953488 0.98723404]
Lowest f1 scoring label: 0.8805031446540881
```

Figure 17 RF performance. Confusion Matrix and F-scores. Note that rows are the actual classes, columns are the predictions.

The `0` has the lowest score on correct classifications. The error rates are higher than on the SVM and the KNN. `0`, `2` , `4` and the `8` are the most error prone.

The impresion is that the `/` shape present in 2, 4, 7, some 8 and some 6es and some 9s is to blame for the errors. This feature might be biased by the model.

The F score for the RF is high with `0.938` , similair to SVM but lower than KNN. The weighted and individual class scores are above the 0.8 requirement.

RF Learning Curves

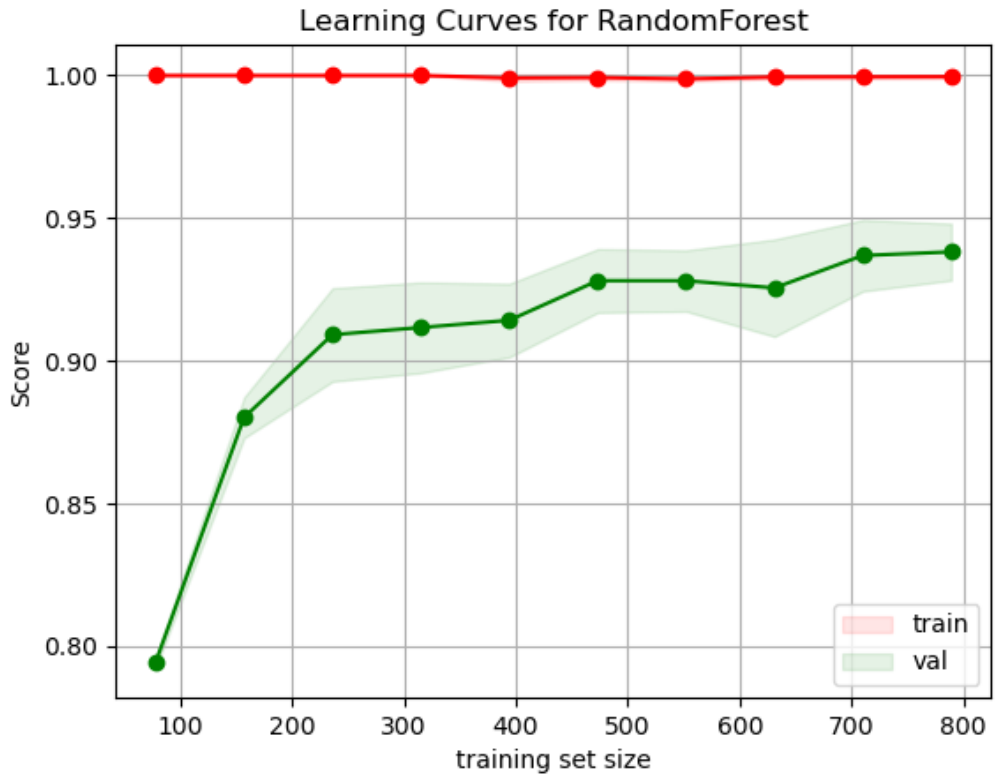
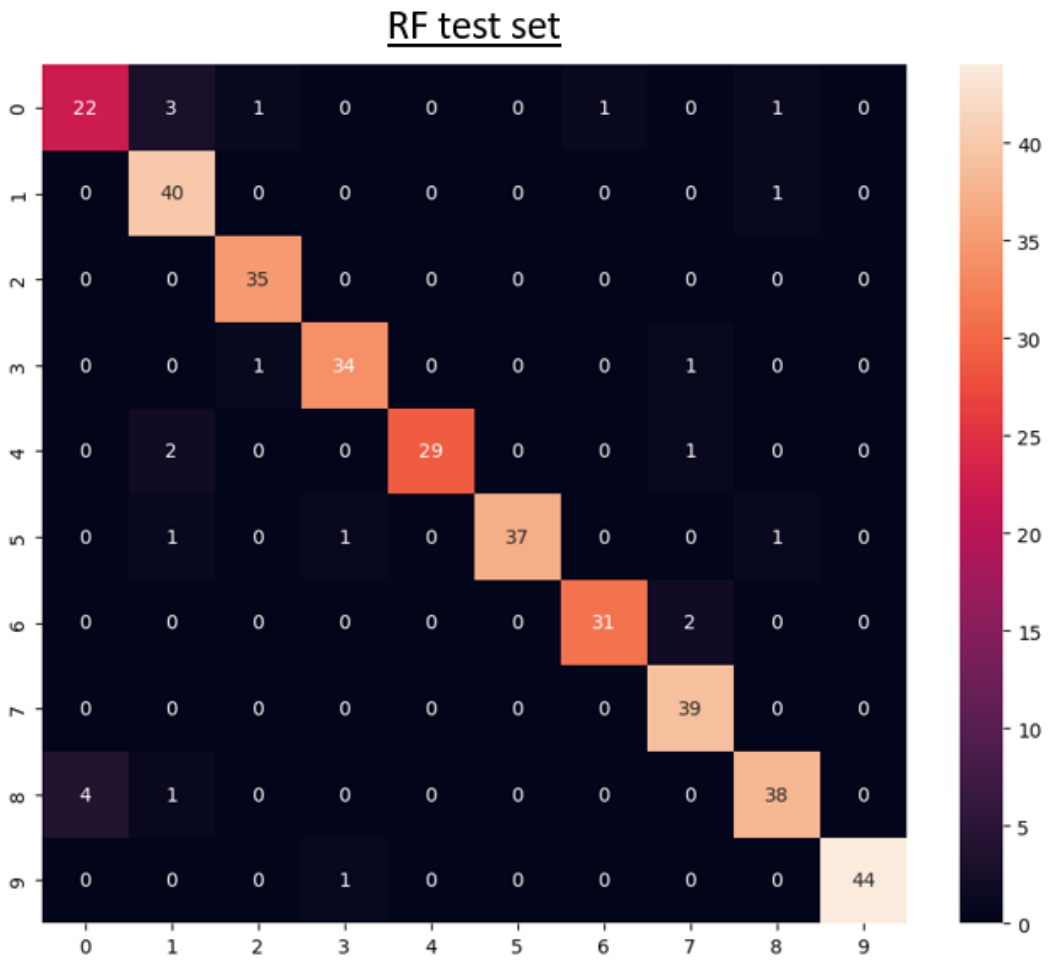


Figure 18 RF Learning Curves

The learning curve for validation shows a standard deviation that is not really shrinking in size when train samples are added. The line is almost plateau like with some valleys. The model might be biased and not fit the data quite well because of the standard deviation not shrinking.

RF Test set results




```
Test set F1 scores
Micro: 0.9381720430107527
Macro: 0.93510226394895
Weighted: 0.9380557413839828
Per label f1 array: [0.81481481 0.90909091 0.97222222 0.94444444 0.95081967 0.96103896
 0.95384615 0.95121951 0.9047619  0.98876404]
Lowest f1 scoring label: 0.8148148148148148
```

Figure 19 test set RF performance. Confusion Matrix and F-scores. Note that rows are the actual classes, columns are the predictions.

Generally speaking, the results are not bad, and overlap the training errors. Some numbers have errors not seen in the training. the most obvious being the `6` classified as a `7`

The number and spread of errors over the labels are bigger than with SVM. Many errors over the labels occur with label `0`. This is curious, as this is not the case in the training. After looking at the `0` samples. There are faulty input samples of `0`. Where only part of the `0` is visible. The trained model has features for `0` that are not unique enough for this class because of these mistakes. This in combination with outlier removal during training can be the cause for the substandard performance of the `0`.

The F1 score drops a bit. The lowest F score is `0.81` vs. the `0.88` from training. This is still acceptable, but a large deviation. Considering the bias and this result the model might be underfitting the data.

6.3.4 Final model selection

Considering the tradeoffs between bias and variance the SVM is outperforming the other models. It is neither overfitting nor underfitting the data. The F-score for all labels is above the required 0.8 and therefor it passes the selection criteria. The deployment will be done on the trained SVM model.

7 DEPLOY AND TEST

Deployment has been done on the training machine (annex 13 and 14). The trained SVM model was saved so it can be loaded into the deployment pipeline. A Python file was created that takes a webcam image as input and outputs a classification. All segmentation steps and feature extraction steps are performed before submitting the data to the classifier.

When you present the webcam with an image, you then press the corresponding number on the keyboard. The code then returns a classification and saves a copy of the image to a folder belonging to the real label. The first character in the filename is the classification made. Pressing the right key is done purely for testing purposes.

7.1 Test plan

To test the deployment a simple testplan has been set up to check the performance of the deployed model. The goal is to have a model that complies to the problem definition:

The project objective is to classify hand written numbers ranging from 0-9, on a A4 sized paper, as their correct numeric representation while using a simple webcam. A minimum of 80% of the samples should be classified correctly (precision). The recall should also be 80%.

To test this the following steps will be taken:

- Create a total of 30 handwritten samples, three of every number in a different style and color. One in black, red, green. This to ensure unique samples.
- Feed each sample 10 times to the model and let a prediction be made. The image must be saved in the folder of the real label. The file name should contain the classification.
- Calculate the recall and precision of the classifications and compare these with the performance criterium.
- Document the test results and visualize them.

Criterium for testing are:

- Diverse handwriting styles should be used for writing the numbers on a single A4. Sampling size should be > 80 For every label (0-9).
- Color should not impact the result.
- The camera should make > 80 coloured 640 x 480 image samples per label, each sample clearly showing the number on the A4 against a neutral (white) background in identical light conditions under an angle of 80 – 100 degrees.
- The sample should be held in one hand and not fixated.
- The sample should be stored in a folder that has the correct label (number) as its name.

- The precision of the test outcome should be 80% of the provided samples are classified correctly.
- It should always make a prediction when a key is pressed.

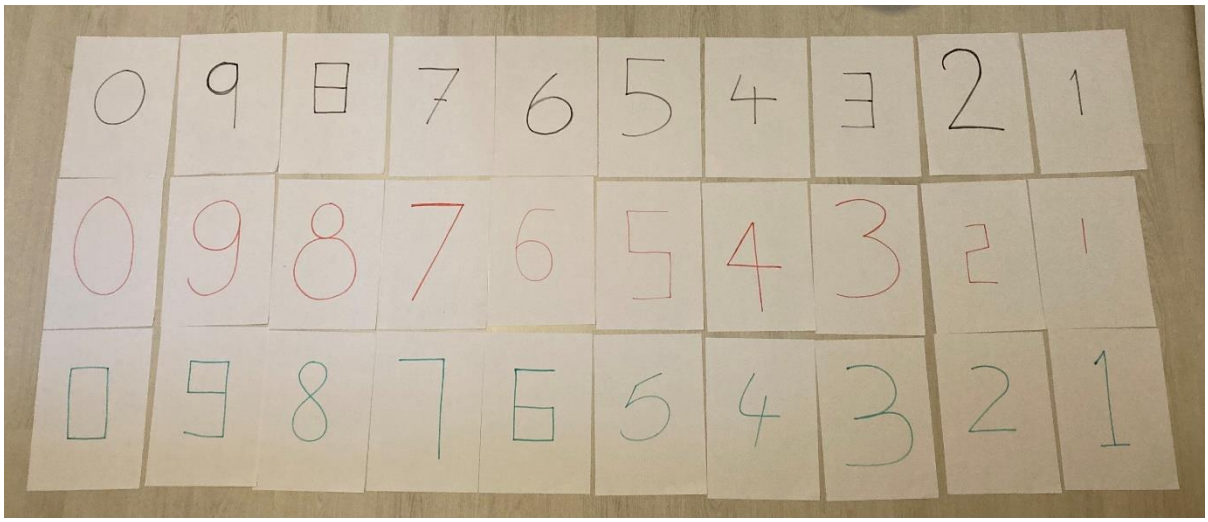


Figure 20 The 30 test samples in unique styles and colors

The styles are defined as digital, for the squared numbers; type, for the keyboard style numbers; rounded for the more rounded handwriting.

7.2 Test results

All but the classification criteria were met during testing. The results from testing are documented in the excel file matrix test outcomes (annex 15). The results have been put in a confusion matrix showing the total scores over the 30 samples. And a confusion matrix has been made that shows the spread over the character styles and colors. This is done to get a better understanding of the results.

Confusion matrix of the 30 samples

	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine	***0,45*** **precision**
Zero	9	4	0	0	1	0	0	0	16	0	0,30
One	0	30	0	0	0	0	0	0	0	0	1,00
Two	4	0	7	0	8	4	1	6	0	0	0,23
Three	2	0	2	21	5	0	0	0	0	0	0,70
Four	3	0	0	0	9	0	9	0	7	2	0,30
Five	0	0	0	0	0	17	0	0	13	0	0,57
Six	1	2	0	3	16	0	6	0	2	0	0,20
Seven	5	7	1	2	4	2	0	9	0	0	0,30
Eight	8	0	0	0	0	0	0	3	19	0	0,63
Nine	1	0	0	5	4	1	1	0	11	7	0,23
0,44 **recall**	0,27	0,70	0,70	0,68	0,19	0,71	0,35	0,50	0,28	0,78	
correct %	30	100	23,3	70	30	56,7	20	30	63,3	23,3	

Figure 21 confusion matrix of the 30 test samples including recall and precision

The matrix shows how the model is not performing according to the requirement of precision and recall >= 0.8.

The model is good at predicting ones and performs average on threes. The highest recall is on the nines just under the required performance. When looking at the color and style spread confusion matrix, red numbers underperform with 5 out of the ten labels having an accuracy of 0. Black numbers do perform the best.

Style and colorwise spread of the 30 samples

	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine	precision black	precision red	precision green
Zero	0	0	0	0	0	0	0	0	10	0	5	4	0	0	0	0	0	0	0	0	4	0	0	0	1	0	0	0	6	0	0,00	0,56	0,36
One	0	10	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	1,00	1,00	1,00
Two	0	0	6	0	0	4	0	0	0	0	4	0	0	0	5	0	1	0	0	0	0	0	1	0	3	0	0	6	0	0	0,60	0,00	0,10
Three	0	0	0	10	0	0	0	0	0	0	2	0	1	6	1	0	0	0	0	0	0	0	1	5	4	0	0	0	0	0	1,00	0,60	0,50
Four	0	0	0	0	8	0	0	0	2	0	3	0	0	0	0	0	0	0	5	2	0	0	0	0	1	0	9	0	0	0	0,80	0,00	0,10
Five	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	7	0	0	3	0	1,00	0,00	0,70
Six	1	0	0	0	8	0	1	0	0	0	0	2	0	3	3	0	0	0	2	0	0	0	0	0	5	0	5	0	0	0	0,10	0,00	0,50
Seven	5	0	0	1	4	0	0	0	0	0	0	0	1	0	0	2	0	7	0	0	0	7	0	1	0	0	0	2	0	0	0,00	0,70	0,20
Eight	0	0	0	0	0	0	0	0	10	0	8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	2	8	0	1,00	0,10	0,80
Nine	1	0	0	0	0	0	0	0	2	7	0	0	0	5	4	1	0	0	0	0	0	0	0	0	0	0	1	0	9	0	0,70	0,00	0,00
style	rounded	type	rounded	digital	digital	type	type	rounded	digital	rounded	type	digital	digital	rounded	type	digital	rounded	type	rounded	type	digital	rounded	type	type	rounded	rounded	digital	digital	type	digital			
recall	0,00	1,00	1,00	0,91	0,40	0,71	1,00	0,00	0,42	1,00	0,23	0,63	0,00	0,43	0,00	0,00	0,00	0,88	0,06	0,00	1,00	0,59	0,50	0,83	0,07	1,00	0,33	0,20	0,31	0,00			
correct %	0	100	60	100	80	100	10	0	100	70	50	100	0	60	0	0	0	70	10	0	40	100	10	50	10	70	50	20	80	0			

Figure 22 Style and color confusion matrix. See the excel for a clearer view.

One of the reasons for this effect is that the red lines are more narrow and lighter of color than the black and green ones. During segmentation this might result in mistakes making predictions harder. The black numbers perform quite good except for the zero, six and seven. Style of the number can also be a key factor. To check this a recall and precision table has been made.

Style based recall and precision

		Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine	total
rounded	*recall*	0,00		1,00					0,00		1,00				0,43			0,00		0,06			0,59			0,07	1,00					0,41
	precision	0,00		0,60					0,00		0,70				0,60			0,00		0,10			1,00			0,10	0,70					0,38
type	*recall*		1,00				0,71	1,00				0,23				0,00			0,88		0,00			0,50	0,83					0,31		0,55
	precision		1,00				1,00	0,10				0,56				0,00			0,70		0,00			0,10	0,50					0,80		0,48
digital	*recall*				0,91	0,40				0,42				0,63	0,00			0,00				1,00						0,33	0,20		0,00	0,39
	precision				1,00	0,80				0,70				1,00	0,00			0,00				0,36						0,50	0,20		0,00	0,46

Figure 23 Style recall and precision matrix. Numbers associated with a style and color are distinguished. See the excel for a clearer view.

Numbers in the type style group have a slightly better performance than the other styles. The lowest scores are also part of the red numbers, that we found out perform worse than the black and green ones.
The rounded zero and seven are the low performing numbers of the black color group. The type version of the six is quite round as can be seen in Figure 20 The 30 test samples in unique styles and colorsFigure 20, and similar in shape to the red six.

The model performance on the black numbers is close to the criterium. The other colors are not. There is room for improvement to be made on the model. In the next chapter the possible fixes are discussed based on the results from the training and deployment of the model.

The problems are most likely linked to the training set. In hindsight we identified the following issues:

- This set contains 2 black style samples for every number. The style and color variations are too low. Color might be less of an issue because of the image processing steps taken, but red samples with thin lines do not perform well as was shown in the test.
- The train set contains wrong samples of numbers that are not fully in the picture. It is unknown if outlier detection has deleted these samples from the data. If not, they pollute the data, making classification harder.
- Consistency of size and position is not maintained throughout the samples. This is part of the challenge of the problem definition, but some more constrains to movement and size during capture should improve the results.

Improving the train set would be a great way to improve the performance of the classifier model itself.

8 CONCLUSION

The deployed SVM model did not meet the performance criteria of precision and recall >= 0.8 when tested on 30 handwritten samples of numbers 0-9. The model performed well in predicting the number 1 and average on predicting the number 3.
The model's performance was impacted by the color and style of the numbers, with red numbers underperforming and black numbers performing the best. It does not generalize well on new images with different handwriting than the style and color of the train and test set.

The problems are for the most part related to the training set. The model is trained on a set containing wrong samples with too little variations of numbers. Garbage in is garbage out seems an appropriate statement for this model.

To improve the model's performance the biggest gain can be made by improving the train set. The set size should be enlarged and contain more variations of style and color. Data Augmentation could be used to enlarge the training set further. The wrong samples should be discarded and a more standardized method of presenting the samples in front of the camera should be used. A fixed frame where the A4 should be presented in is a solution to this issue. Size of numbers can be still influenced by writing and data augmentation.

There is also still improvement possible on the feature side. Only six features have been used, a couple more might help the classification. Round shapes and shapes containing a `/` are often mistaken for another number. More features from the Hu moments or a roundness feature could improve the model performance.

Reflecting on our work in relation to the ML project checklist we have implemented steps in an effective way, but there is also room for improvement. Data collection and exploration could be greatly improved. The wrong samples should have been extracted and we could have used more style examples and constrain the way we presented them to the camera more. The data set of features could have had an instance reference to the corresponding image. This would have made it easier to explore and analyze outputs during the other ML steps.

On the positive side we have explored different models and fine-tuned the hyperparameters for each. We have documented our steps in Jupyter notebooks and made use of custom functions to ensure data preparation was constant.

The project has learned us the whole cycle from data creation to classification model deployment. We also learned to be more critical of the data we feed the model in order to make a classifier that generalizes well. It is a slight disapointment that the deployed model did not perform as expected, but the proces of building and analysing it was of great interest.

9 REFERENCES

Feature Request: Pipelining Outlier Removal · Issue #9630 · scikit-learn/scikit-learn. (z.d.). GitHub. Geraadpleegd 3 januari 2023, van https://github.com/scikit-learn/scikit-learn/issues/9630

Jayaswal, V. (2020, November 9). *Local Outlier Factor (LOF)—Algorithm for outlier identification*. Medium. https://towardsdatascience.com/local-outlier-factor-lof-algorithm-for-outlier-identification-8efb887d9843

10 APPENDICES

This chapter contains the refferences to the Python code scripts and Jupyther notebooks.

1. AcquireNumbers.py – Custom script to aquire the images from the webcam.
2. FeatureExtract.py – Custom script that segments an image and returns an array of 13 features.
3. Data Preperation.ipynb – Notebook with the data preperation steps.
4. Data Segmentation and f-extraction.ipynb – Notebook with the segmentation and extraction steps.
5. Data Exploration A Train Test.ipynb – Notebook with the data exploration steps.
6. Data Exploration B Outlier detection.ipynb – Notebook with the outlier detection steps.
7. Data Exploration C Feature selection.ipynb – Notebook with the feature selection steps.
8. Data Exploration D Pipeline design.ipynb – Notebook with pipeline design steps.
9. TrainTestSetFunction.py - Custom script that functions as a data processing pipeline.
10. Training and evaluation SVM.ipynb – Notebook containing the training and evaluation of the SVM.
11. Training and evaluation KNN.ipynb – Notebook containing the training and evaluation of the KNN.
12. Training and evaluation RF.ipynb – Notebook containing the training and evaluation of the RF.
13. Deployment.ipynb – Notebook used to deploy the model.
14. PredictNumber.py - Custom script that is used as a deployment.
15. matrix test outcomes.xlsx – Test outcomes.