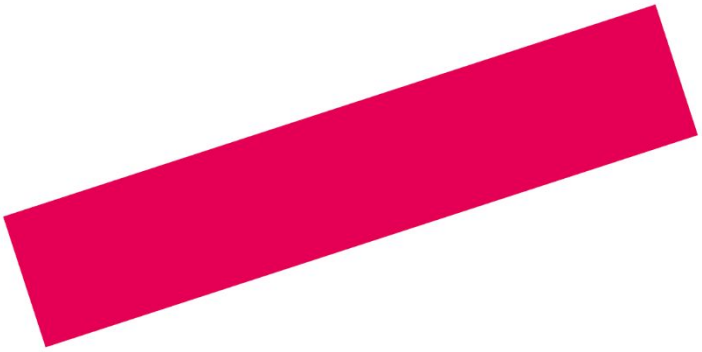

EVD3 Machine learning portfolio

By submitting this portfolio the authors certify that this is their original work, and they have cited all the referenced materials, in the forms of texts, models, and books properly.

Project Group:	Teun Nooteboom,	602598
	Stijn de Moed,	604868
	Renzo Broekman,	604593
Supervisor:	Jeroen Veen	
Date:	21.10.'20	

Document History

Version	Date	by	Change
V2.0	21.10.'20	Teun, Stijn, Renzo	Checked | The contents of the test and conclusion Added | Short introduction to chapter 2 Added | Conclusion of the project Translated | Test results to English
V1.5	19.10.'20	Teun, Stijn, Renzo	Checked | The documents flow and contents Added | Results of the created tests Added | Géron Machine Learning Checklist
V1.4	15.10.'20	Renzo	Added | Test results of the model training
V1.3	14.10.'20	Teun, Renzo	Added | Information to model training Added | Plan for testing the deployed system
V1.2	12.10.'20	Teun, Renzo	Added | Information to the model selection Added | Explanation how the model is trained
V1.1	01.10.'20	Teun, Renzo	Improved | Layout based on feedback
V1.0	28.09.'20	Teun, Stijn, Renzo	Check | Documentation for handing in Added | Deploy code block in appendix
V0.6	25.09.'20	Teun, Renzo	Added | Conclusion for the graphs Translated | Data acquisition and exploration to English
V0.5	24.09.'20	Teun, Stijn, Renzo	Check | If the content of the document is good and complete Added | The diagrams to data exploration Added | information about pipelines to data acquisition Translated | data acquisition and exploration to English
V0.4	23.09.'20	Teun, Renzo	Added | More information to data acquisition and exploration Translated | data acquisition and exploration to English
v0.3	16.09.'20	Teun, Stijn, Renzo	Added | More information to Problem statement Added | More information to data acquisition and exploration Translated | data acquisition and exploration to English
V0.2	10.09.'20	Teun, Stijn, Renzo	Added | Introduction Added | Information to Problem statement Added | Information to data acquisition and exploration
V0.1	09.09.'20	Teun, Stijn, Renzo	Added | Introduction Added | Bullet points
V0.0	05.09.'20	Renzo	Created | The document

Table of content

1. Problem statement **5**

2. Data acquisition and exploration **6**
2.1. Feature research 7
2.2. Conclusion 11
2.3. Pipelines 11

3. Model selection and training **13**
3.1. Model selection 13
3.2. Training 15

4. Deploy and Test **22**
4.1. Deployment 22
4.2. Testing 22

5. Conclusion **26**
5.1. Results of the project 26
5.2. Overview of the project 27
5.3. Takeaways from the project 28

Introduction

In this report, it is discussed how we can differentiate between 4 hand gestures by using embedded vision and machine learning. Machine learning is a way to classify images by training a system. The machine learning software learns how to recognize the hand gestures by comparing the features of the new image to the features that are learned in the database.

We learn how to create a machine learning system in the EVD3 course. EVD3 is relevant to the practical domains we are interested in for example AR for games. The relation of EVD3 to our minor project is that it could be used to recognize the different objects that the camera can see. For locating the objects x/y coordinates it would be slightly less useful.

The objectives of the project are to learn more about machine learning. Like how do you create a Python script that can turn data into a system that can recognise similarities in other images? How can you make the system lightweight enough to allow it to run on lower-end hardware like the Raspberry Pi? However, we will also learn about what the different workflows are and which are useful to us. And how do you handle your data, what data is useful and which is not?

Machine learning is a useful subcategory of embedded: vision; it helps with classifying the obtained data in an easier and more efficient method. It does this by comparing the learned data to the newly obtained image. However, in machine learning, you still need to tell which features it needs to look at.

The first chapter describes the problem which is going to be solved and what the given requirements for the problem are. The second chapter describes how the data is acquired and how useful information is extracted from the data. The third chapter describes which model is chosen for training and how the training process went. The fourth chapter describes how the system was deployed on the Raspberry Pi and how the test was made to see if the created model works. The last chapter concludes the tests and if the created system works like expected and what needs to change to improve the created system.

1. Problem statement

For the course EVD3, a system needs to be created that can recognise 4 different signatures. The system has a few requirements. The first requirement is the recognition of the 4 different hand signatures. These signatures are rock (a fist with the thumb over the other finders), paper (open hand with the fingers spread), scissors (index and middle finger up) and at last loose hand where the thumb and pink is held out. In figure 1 the gestures are shown. To make it easier for the system to recognize the gestures all gestures need to have the palm face towards the camera with a maximum angle of 45 degrees.
The system will not have a specific class to detect other objects or an empty image. However, there will be a threshold with how much the system needs to be certain. If it falls below this threshold it will be set as unknown.
The system needs to be able to recognize the different hand gestures with 80% accuracy.

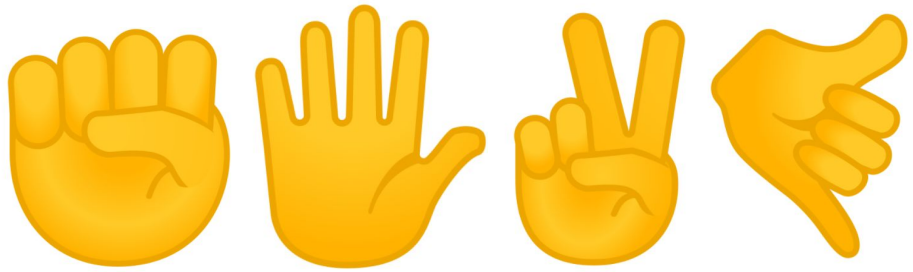

Figure 1: Visual representation on how the hand gesture look

Another requirement is, is that the system needs to be able to run on a Raspberry Pi. This means that the software needs to be lightweight enough it can run on a Raspberry Pi. The software also needs to be able to identify what the system is seeing within 1 second of obtaining the image.

Before we can start with recognising different hand gestures we need to create a database with labelled data. The database will consist of three different hand types however all hands are from a male humans (From Teun, Stijn en Renzo). For this reason the system will work mainly for those hand types. The hands need to be at a maximal distance of 40 cm from the camera and it is important that the entire hand and part of the wrist is visible.
The lighting circumstances will differ between the datasets from the different sources, however, as long as the background can be differentiated between the foreground and background this should not be an issue.
All photos are made using the Raspberry Pi camera v2. The most important reason is that this camera is the easiest to use together with the Raspberry Pi. The image quality is high enough to recognise the gestures. Only one type of camera is used because for training purposes the incoming data must be relatively similar to each other. For this reason the settings of the camera will also be the same. This will make the extraction of the features more reliable.

All the requirements are approved by Jeroen Veen. The system also needs to be working before the end of the first block of semester 7.

2. Data acquisition and exploration

To get the images a script was provided by Jeroen Veen called “aquire.py” this software makes an image and saves it in a labelled map whenever a corresponding key is pressed. The letters used are “H” for loose hand, “P” for paper, “F” for fist and “S” for scissors.

When enough data is collected the database is cleaned. All hand gestures that are not sharp enough because of too much movement, because the angle is not within specification or other problems are removed. This is done manually although there is a possibility bad data gets through, it should be a small enough percentage that it will have minimal impact on the deployed system.

After the database is cleaned from most outliers, unclear images or any other image that isn’t useful as training data for the algorithm. The features of the images will be plotted using the “explore.py” Python script. The features should make it easy to differentiate between different hand gestures. Each feature should be able to differentiate at least two different hand gestures. Multiple features are used because some features are better at differentiating between “rock” and “paper” while others might be better at differentiating “paper” and “loose hand”.

The features that are used are the Area, the Perimeter, the Aspect-ratio, Extent and Solidity. The feature area is the surface area of the hand in pixels. The Perimeter is the perimeter of the hand in pixels. The Aspect-ratio is the ratio between the max-width and the max height. The Extent is the surface area of the hand divided by the perimeter of the hand. And as of last, Solidity is the surface area divided by the area of the hull. The hull is the line around the object of which the angles are smaller or equal to 180 degrees. The hull area is the area within this line.

The images are created on the Raspberry Pi using the Raspberry Pi camera V2.1. all pictures are taken with a top-down view. For each hand gesture, 200 images are created for a total of 4 * 200 = 800 images. After the images are created they are transported to the pc using a vnc viewer. This is done because the pc has more processing power which makes extracting the features and training the algorithm faster. The algorithm can be moved to the Raspberry Pi after the training is completed.

2.1. Feature research

In chapter 2.1 the acquired data is researched to see if the created features are useful for differentiating between the different classes. This is done by creating different types of diagrams that showcase how the feature interacts with the classes and with other features. The main graphs used for this are the scatter plot and the box plot.
The scatter plot plots two different features against each other depending on how well the features work together the less overlap the classes have. For the box plot the feature is checked on how well it works on its own. The bigger the distance between the boxes the better the feature is for differentiation between classes.
The other graphs that are drawn is a simple overview which shows how much data for each class is available. Another graph is the distribution graph which shows how the data is distributed over the data set. And the last graph is a heatmap which shows how much overlap different features have with each other the closer to 1 the more similar they are.

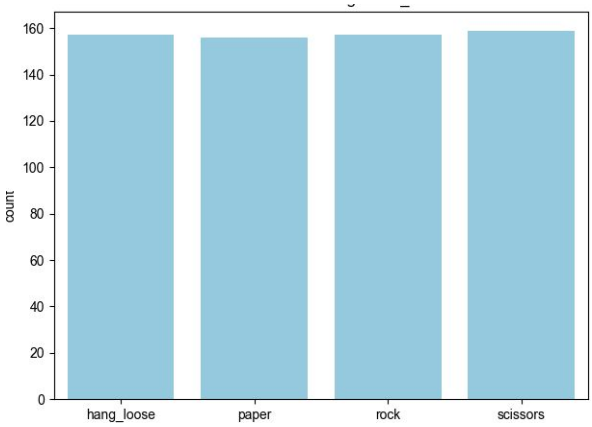

Figure 2: The amount of training data per hand gesture.

Figure 2 shows the amount of images that are used for training purposes. This is 75% of the total data set. In total there are 210 images per hand gestures in the data set meaning that there are about 52 images per hand gesture for the test set.

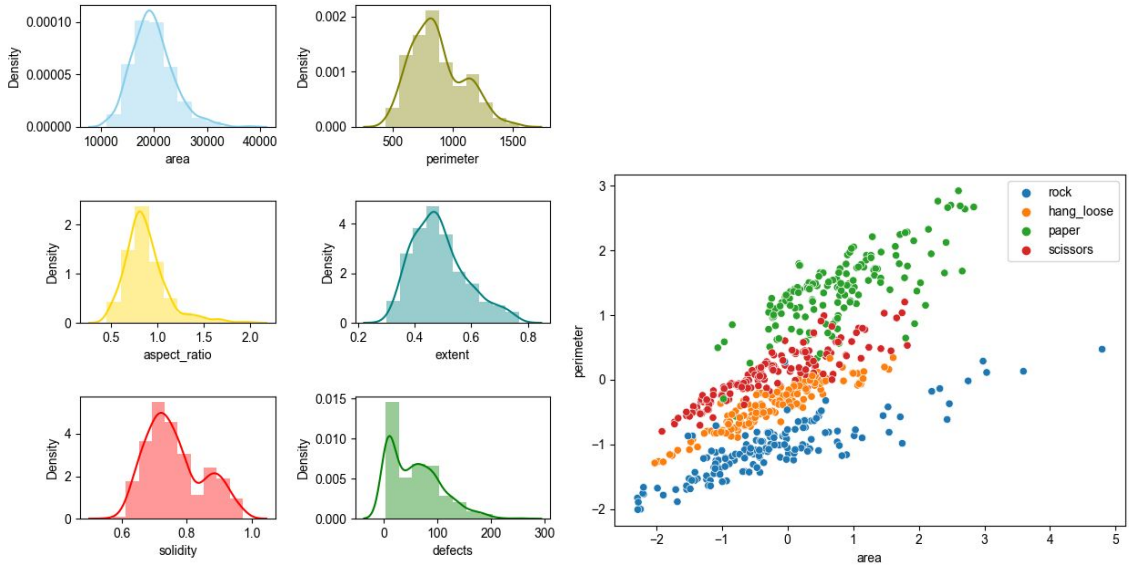

Figure 3 A and B: Distribution of the features over the data set Scatter plot area against perimeter

Figure 3A shows the values of the features that are used and shows how they are divided. From this diagram, it is easy to see if there are any obvious outliers within the created data set. The order of the features shown are area, perimeter, aspect ratio, extent and the solidity.
Figure 3B shows the relation between the perimeter and the area. This relation shows a clear distinction between each of the hand gestures. However, there is still some overlap between some of the gestures so more features need to be used.

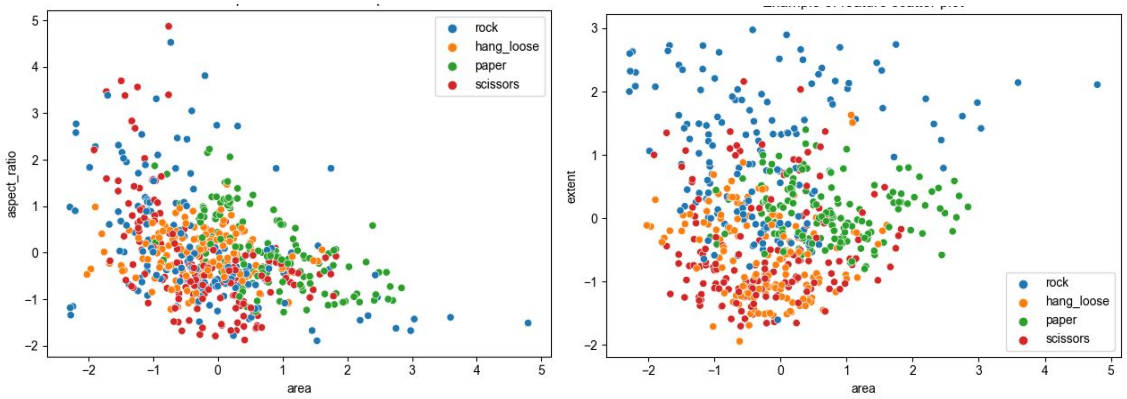

Figure 4: A and B: The scatter plot area against aspect ratio and area against the extent

Figure 4A shows the relation between the area and the aspect ratio. Here you can see that not all relations will give a clear distinction between the gestures.
Figure 4B shows the relation between the area and the extent. However, the difference is not clear enough to use this information to differentiate between the hand gestures.

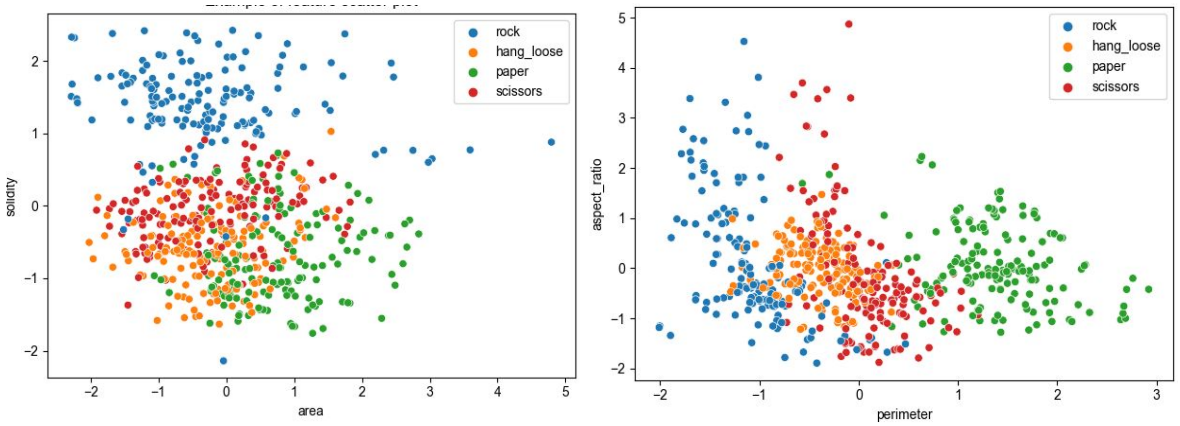

Figure 5: A and B the scatter plot area against solidity and perimeter against aspect ratio

Figure 5A shows the relation between the area and the solidity. From this graph, the rock is easily differentiated from the other hand gestures.
Figure 5B shows the relation between the perimeter and the aspect ratio. From the graph, it is possible to differentiate the different hand gestures. However, it would be mainly useful to see the difference between paper and rock

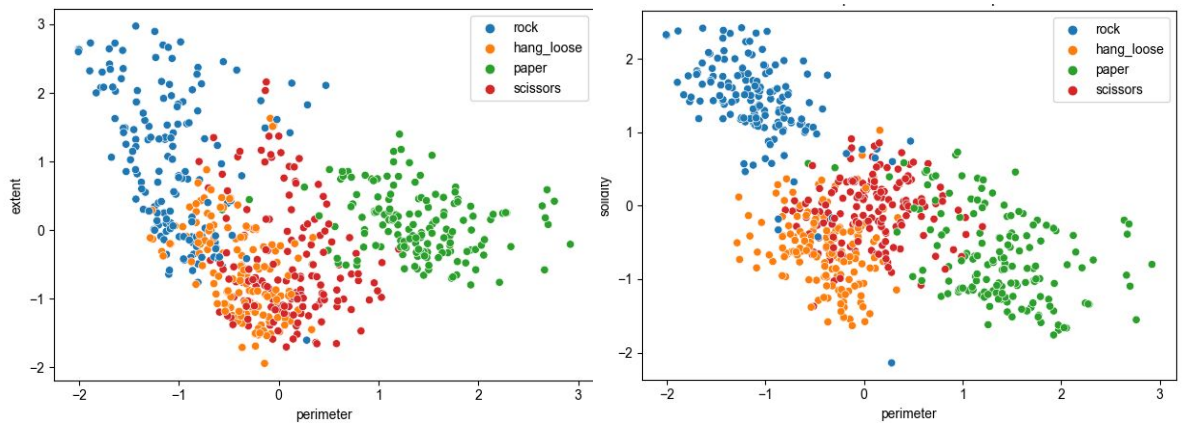

Figure 6: A and B the scatter plot perimeter against extend and perimeter against the solidity

Figure 6A shows the relation between the perimeter and the extent. The graph again shows a good way to separate paper from rock. However, hang loose and scissors are still difficult to separate.
The same can be seen in figure 6B where the relation is shown between perimeter and solidity. Although it is slightly better here.

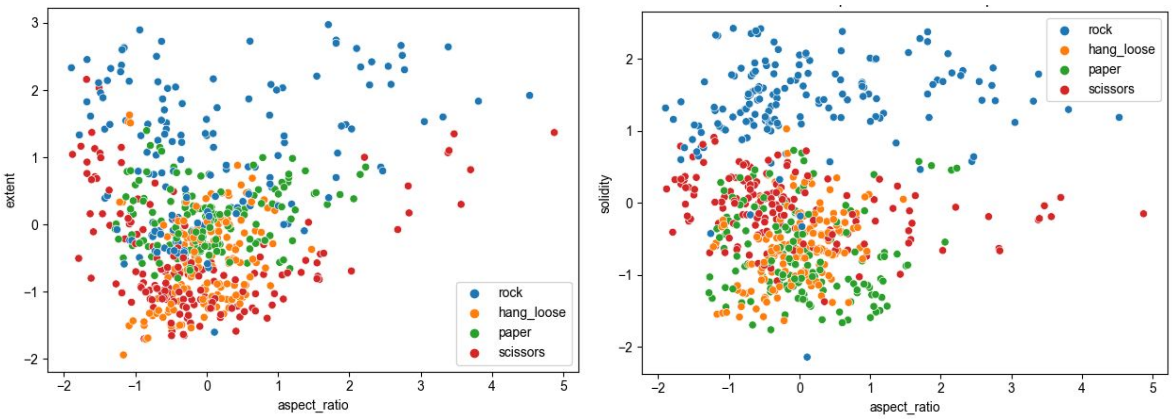

Figure 7: A and B the scatter plot aspect ratio against extent and expect ratio against solidity

Figure 7A the relation between aspect ratio and extent. From this graph, only limited information can be obtained.
Figure 7B shows the relation between aspect ratio and solidity. Again from this graph, a clear distinction between rock and the other hand gestures can be found.

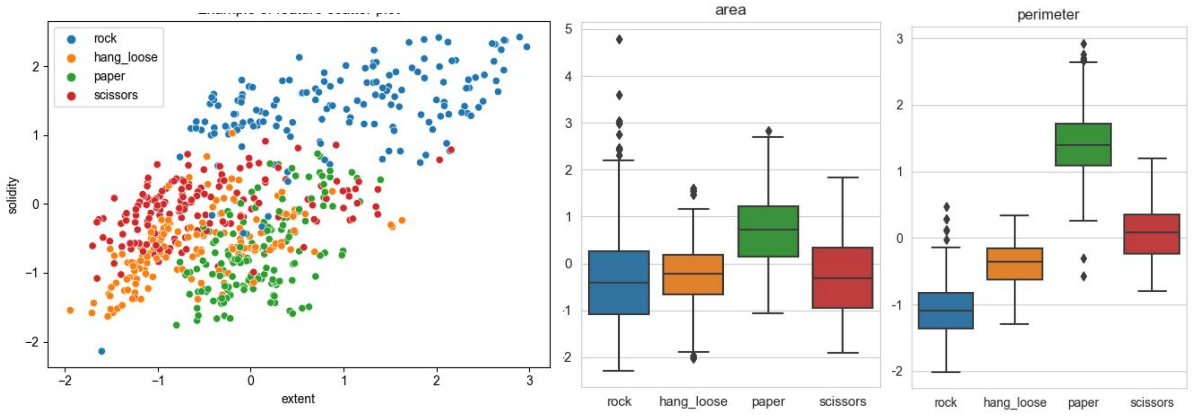

Figure 8: A and B the scatter plot extent against solidity and the boxplots of area and perimeter

Figure 8A shows the relation between the extent and the solidity. Rock and hand are again easily differentiated from the other hand gestures.
Figure 8C shows the boxplot of the perimeter. Here it is easy to differentiate between rock and paper. Figure 8B shows the boxplot of the area for the different hand gestures from this graph. It is possible to make it possible to see a slight difference between paper and the other gestures.

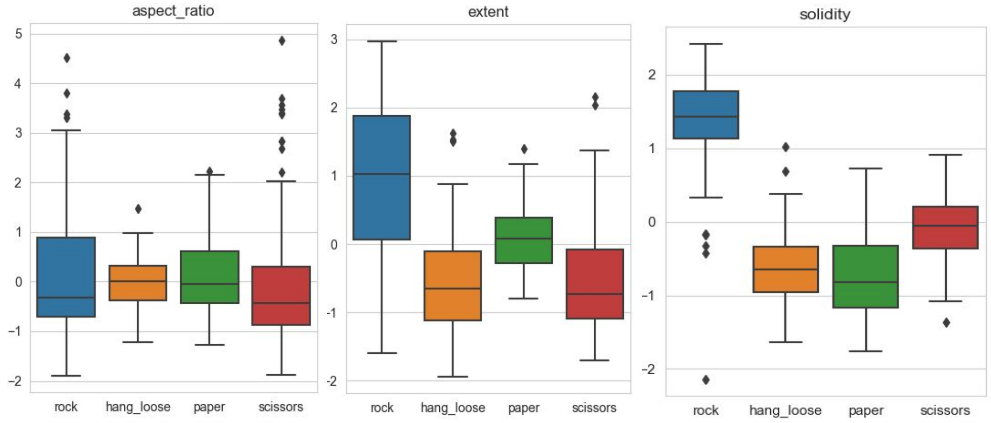

Figure 9: A, B and C: The boxplots for aspect ratio, extent and solidity

Figure 9A shows the boxplot of the aspect ratio no real distinction can be made between the different hand gestures.
Figure 9B shows the boxplot of the extent again it is possible to differentiate rock from the other gestures.
Figure 9C shows the boxplot of the solidity which again shows that rock is easily differentiated from the other hand gestures. Scissors is also slightly able to be differentiated from the other gestures.

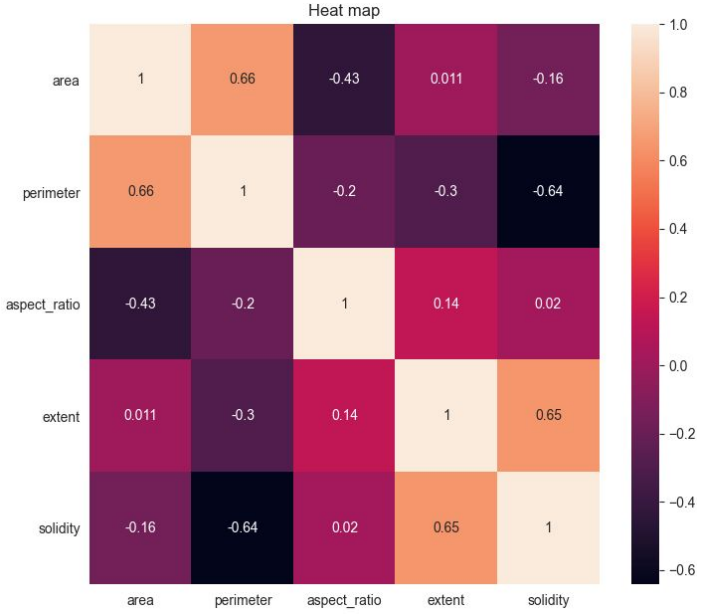

Figure 10 Heatmap correlation between the different features.

Figure 10 shows the correlation between the different features which means that if two values have a high correlation the value would be closer to 1. So to ensure that there are no features that are exactly the saem the value should be as far as possible from 1.
The main reason for this is because otherwise the algorithm will be trained thinking that this attribute is twice as important compared to the other features.

2.2. Conclusion

From the research above it is decided that aspect ratio will be removed because none of the graphs that contained the aspect ratio added any new information to differentiate between the different gestures.
With the current features it is also quite difficult to differentiate scissors from hang loose. For this another feature needs to be added like counting the amount of holes in a hull. Or for example the largest distance between the contour and the hull. However, because of the limited amount of time no extra research will be made towards finding out how these features could be calculated. That said with the current features it should still be possible to achieve the 80% guess rate which was described in the problem statement.

2.3. Pipelines

To control the data flow within the software pipelines are used[1]. The setup for the pipelines are as follows:

```
steps1 = [('scaler', StandardScaler()), ('SVM', SVC())]
steps2 = [('scaler', StandardScaler()), ('decsT', DecisionTreeClassifier())]
steps3 = [('scaler', MinMaxScaler()), ('SVM', SVC())]
steps4 = [('scaler', MinMaxScaler()), ('KNearest', KNeighborsClassifier())]
pipe1 = Pipeline(steps1)
pipe2 = Pipeline(steps2)
pipe3 = Pipeline(steps3)
pipe4 = Pipeline(steps4)
```

First, a variable “steps” is created which holds the steps that the program will go through. The pipeline only consists of two steps scaling of the data that ensures that data with larger scales do not influence how the algorithm performs. This can be done in different ways like with MinMaxScaler or with StandardScaler. Both ways have their pros and cons. However, for this project, the StandardScaler is more likely to be used. This is because it is less affected by outliers. Because of the limited data size outliers would be a bigger problem. The second part of the pipeline is the model used. In the next chapter, the best model will be chosen however, above a few are shown like SVM, Decision tree classifiers and K-nearest classifiers.

```
parameters1 = {'SVM__C': [10, 100, 10e5], 'SVM__gamma': [0.1, 0.01]}
parameters2 = {'decsT__criterion': ['gini', 'entropy'],
        'decsT__max_depth': np.arange(3, 15)}
parameters3 = {'KNearest__leaf_size': np.arange(20, 25),
        'KNearest__n_neighbors': np.arange(1, 4), 'KNearest__p': [2, 3]}
```

[1] (n.d.). sklearn.pipeline.Pipeline — scikit-learn 0.23.2 documentation. Retrieved September 24, 2020, from http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

After that, the possible parameters that can be changed or are the most important are tested. This happens automatically however the system does expect some suggestions. These suggestions will be changed stepwise to ensure that the most optimal values are used.

```
grid1 = GridSearchCV(pipe1, param_grid=parameters1, cv=10)
grid2 = GridSearchCV(pipe2, param_grid=parameters2, cv=10)
grid3 = GridSearchCV(pipe3, param_grid=parameters1, cv=10)
grid4 = GridSearchCV(pipe4, param_grid=parameters3, cv=10)
```

The pipeline together with the parameters are sent into a grid-search this will help with finding the best parameters. To do this it uses cross-validation 10 times. This means that it will check how well the created algorithm works against its training data.

```
grid1.fit(trainX, trainY)
print("score = %3.2f" % (grid1.score(testX, testY)))
print(grid1.best_params_)

grid2.fit(trainX, trainY)
print("score = %3.2f" % (grid2.score(testX, testY)))
print(grid2.best_params_)

grid3.fit(trainX, trainY)
print("score = %3.2f" % (grid3.score(testX, testY)))
print(grid3.best_params_)

grid4.fit(trainX, trainY)
print("score = %3.2f" % (grid4.score(testX, testY)))
print(grid4.best_params_)
```

After the training is done it will score how well it does against the testing data. And shows which parameters it decided were the best.

3. Model selection and training

This chapter is divided into the selection of the model and the training of the chosen model. The model selection compares different classifier algorithm and choses the one with the best results based on different criteria,
The training gives more information about how the training is done and what the end result of the training was.

3.1. Model selection

For the model selection there can be chosen from SVM, K-nearest, decision tree and many more. First start with dividing the training, testing and validation. The testing set needs to be completely separated and won't be used until the end.

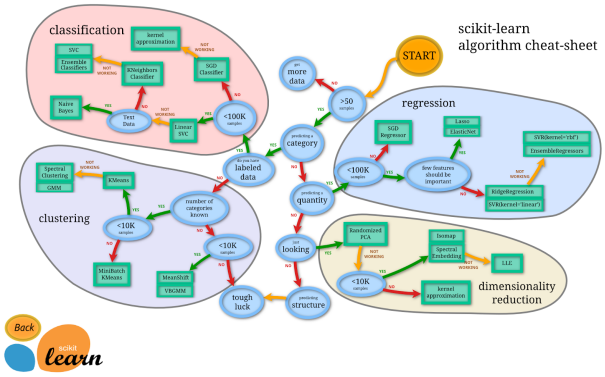

Figure 11: Cheat-sheet which algorithm should be chosen[2]

In the image above figure 11 shows which algorithm should be chosen for this project. This graph shows that first the linear SVC should be used if this doesn't work the KNeighborsclassifier and as last the SVC ensemble classifiers. However, for this project to see the effect of different classifiers they are tested at the same time to see which one is the best for the project.

The ones tested are SVC in rbf and linear mode, KNeigboursclassifier and decision tree classifier. For all these algorithms the standard scaler is used instead of MinMax because from the previous chapter it was shown that there are still some outliers within the dataset. When using the standard scaler these outliers shouldn't have too much influence during the training phase. Scaling is necessary for this dataset because the different features are in different scales which can create biases when training the algorithm.

Because the dataset is quite small, splitting the dataset in different ways can cause relatively big differences during the training process. To counteract this the learning curves are drawn of the different algorithms. These learning curves split the data set in different ways to ensure that the learning curve is the same no matter how many times the program is used. It is also chosen to keep the hyperparameters as default for this comparison. Although this doesn't exactly show how well these algorithms work it is the easiest and fastest way to compare them.

[2] (n.d.). Choosing the right estimator — scikit-learn 0.23.2 .... Retrieved October 9, 2020, from http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

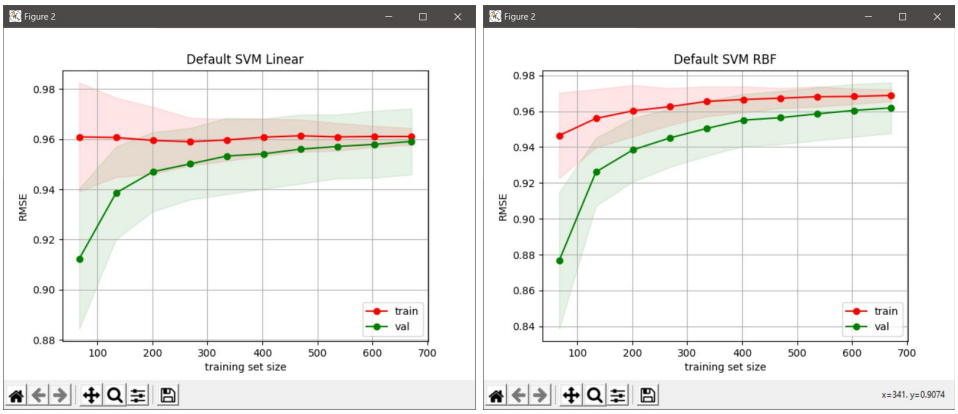

Figure 12: Learning curves of SVM linear and RBF with default hyperparameters

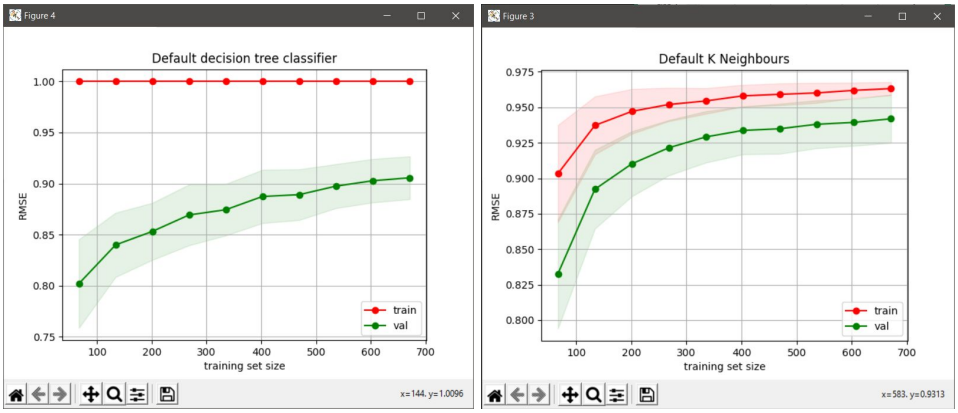

Figure 13: Learning curves of DTC and K-neighbour with default hyperparameters

The learning curves shown above are drawn using the code provided by Jeroen Veen.

From the figure 12 and figure 13 it is possible to see which one performs the best for the given dataset. The decision tree classifier performed the worst with performing well on the training dataset but performing relatively bad on the validation set. The line also seems to straighten out meaning that having a larger dataset will most likely not improve the algorithm further and it could only improve further with better hyperparameters and better features. K-Neighbours performed pretty well although relatively speaking worse than the SVM algorithm. The last two SVM linear performed well as its validation set scored the same as the training set meaning that it is reliable in its predictions. However, it's overall score is quite low, maxing out on 0.96. Compared to RBF that has an accuracy on the training set of 0.97. By fine tuning the hyperparameters the RBF algorithm will most likely be able to perform better than the linear model. Besides that the linear SVM algorithm has also been slower to calculate the hyperparameters when running on a laptop compared to the RBF SVM algorithm.

For this reason it has been chosen to use the SVM with the RBF kernel. After fine tuning this model should perform the best on our training set.

3.2. Training

For training it is chosen to use the workflow represented below in figure 14. Here it is shown that the data set is split into three smaller sets: the training, validation and test set, the training and validation set is used to find which model should be used and which hyperparameters. After all this is done the test set is used to evaluate the algorithm. This is done to prevent overfitting of the algorithm and see how well it generalizes on nieuw data.

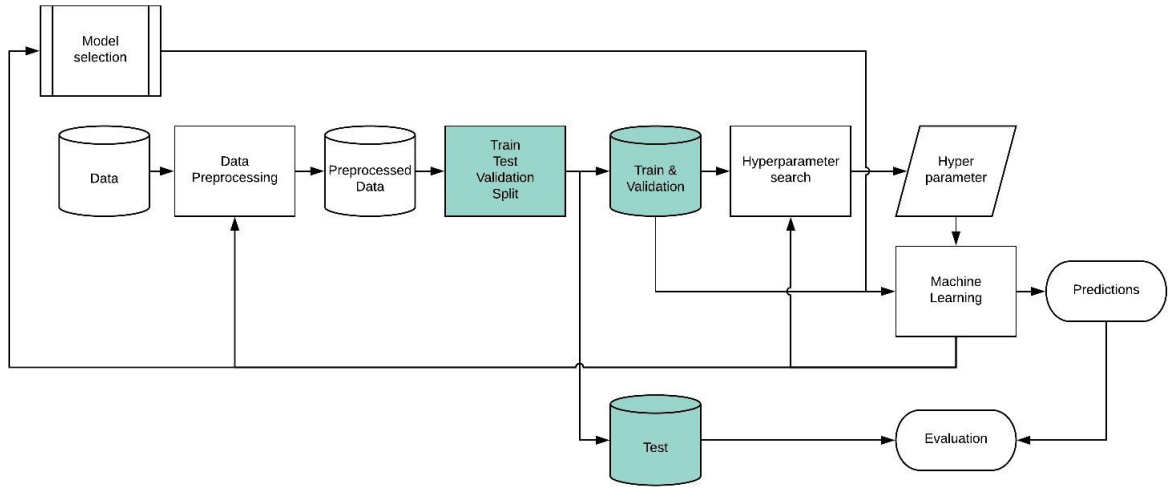

Figure 14: The workflow for this project[3]

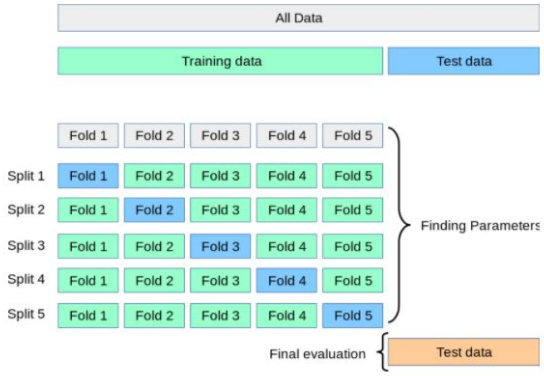

Figure 15: Standard way grid cross validation.

After splitting the dataset there are a few ways to find the hyperparameters that function the best. One of these ways is to use Grid Cross Validation shown in figure 15.
However, for this project shuffle splitting was used for the cv value. With 100 fold split this way there is a lower chance that the algorithm gets stuck in a local optimum because of a specific training set.

To do this the following code was used.

```
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=48)
grid1 = GridSearchCV(pipe1, param_grid=parameters1, cv=cv)
```

[3] (2018, December 20). (My) Machine Learning Workflow. Epistemic status ... - Medium. Retrieved October 12, 2020, from https://medium.com/datadriveninvestor/my-machine-learning-workflow-7576f7dbcef3

Besides having a lower chance of getting stuck in local optimum, this project has a relatively small dataset. Therefore, there is a higher chance that splitting the dataset in a specific way causes the algorithm to get better or worse results. This is shown below in figure 16. Here if we use the same dataset but split it using two different seeds one algorithm performs way better on the validation set compared to the other. To avoid this using the shuffle splitting method is also preferred.

	precision	recall	f1-score	support
hang_loose	0.98	0.98	0.98	53
paper	1.00	1.00	1.00	52
rock	0.98	1.00	0.99	52
scissors	1.00	0.98	0.99	53
accuracy			0.99	210
macro avg	0.99	0.99	0.99	210
weighted avg	0.99	0.99	0.99	210

	precision	recall	f1-score	support
hang_loose	0.98	0.94	0.96	53
paper	1.00	0.94	0.97	52
rock	0.98	1.00	0.99	52
scissors	0.91	0.98	0.95	53
accuracy			0.97	210
macro avg	0.97	0.97	0.97	210
weighted avg	0.97	0.97	0.97	210

Figure 16: Classification_report seed 48 and 86.

Plotted using the following code

```
predictY = grid1.predict(testX)
print(classification_report(testY, predictY, target_names=gestures.unique_targets))
```

Plotting learning curves can give a good overview on how well the algorithm performs. To plot these SKlearn has provided their users some software[4]. These learning curves show how well the algorithms perform for a given set of data. When drawing these learning curves the shuffle splitting is also used to smooth out the line. The learning curve represents the training and the validation. The closer they are together at these lines are at the end of the training dataset the better the algorithm performs on the validation set compared to the training set.

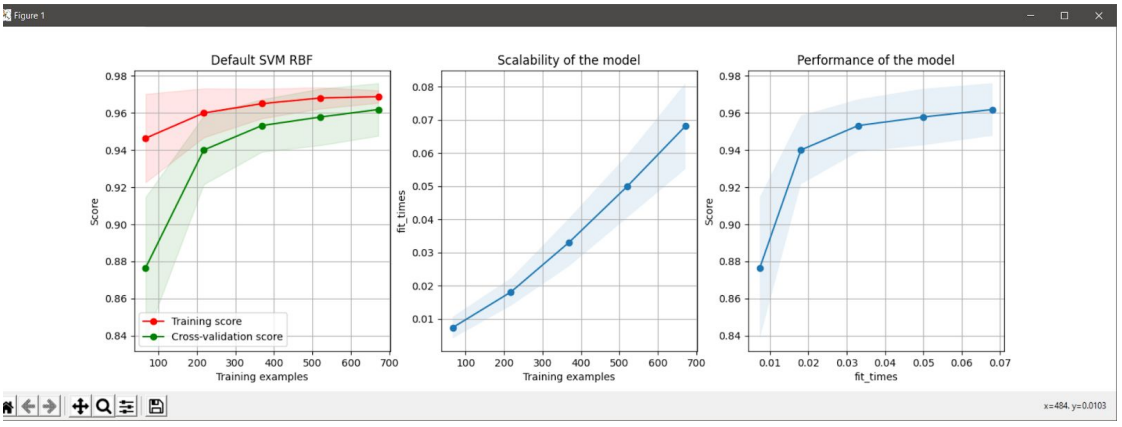

Figure 17: Learning curve of the default algorithm

[4] (n.d.). Plotting Learning Curves — scikit-learn 0.23.2 documentation. Retrieved October 9, 2020, from http://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html

	precision	recall	f1-score	support
hang_loose	0.98	1.00	0.99	53
paper	1.00	0.96	0.98	52
rock	1.00	1.00	1.00	52
scissors	0.96	0.98	0.97	53
accuracy			0.99	210
macro avg	0.99	0.99	0.99	210
weighted avg	0.99	0.99	0.99	210

Figure 18: Classification report of the default algorithm

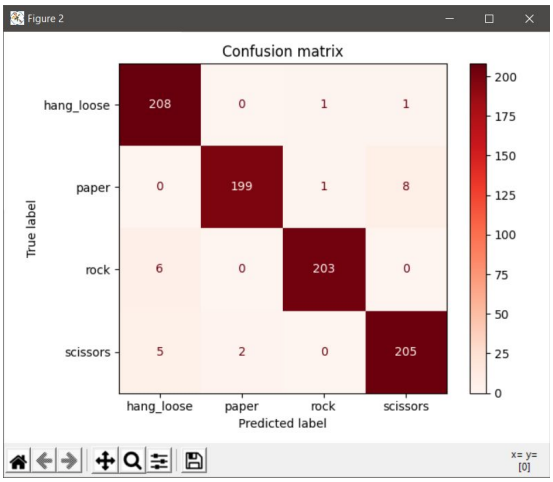

Figure 19: Confusion matrix of the default algorithm

In the figure 17 above, the learning curve is shown. The plots in the second column show the times required by the models to train with various sizes of training dataset. The plots in the third column show how much time was required to train the models for each training size. figure 18 figure 19 shows the confusion matrix this plot shows which one was wrongly predicted and what it did predict instead.

Plotted with the following code:

```
def show_confusion_matrix(estimator, x, y, label):
   fig0, ax0 = plt.subplots(1, 1)
   plot_confusion_matrix(estimator, x, y, ax=ax0, display_labels=label,
cmap="Reds")
   ax0.set_title('Confusion matrix')
   plt.tight_layout()
```

To tune the algorithm the following steps were taken. First a relatively wide range was given, this range was 1, 10, 100, 1000, 10000 for the C hyperparameter and 1, 0.1, 0.01, 0.001 for gamma. This was done to find which parameter would be best on average. These values were used in the gridsearchcv function. This function returned C=10 and gamma=0.1. With this information the next possible parameters were given. It was chosen 5, 10, 15 for C and 0.15, 0.1, 0.05 for gamma. Again by using the gridsearchcv function it returned C=6 and gamma=0.15. After repeating this process a gain the final result gave C=6 and gamma= 1.49. These hyperparameters were chosen for the tuned model which can be seen graphically below.

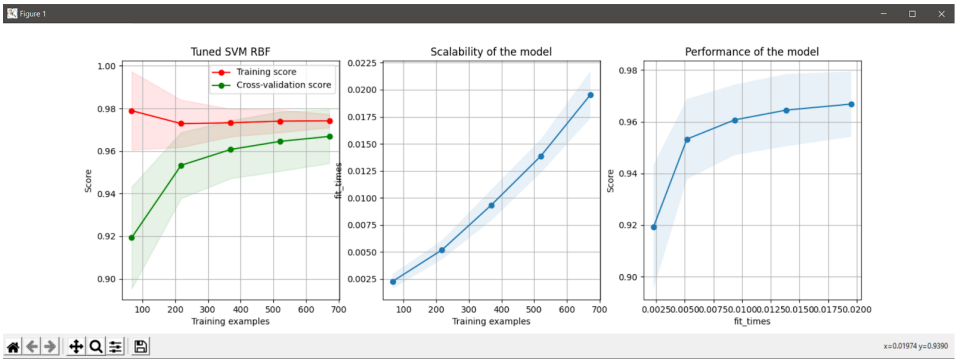

Figure 20: Learning curve of the tuned algorithm based on the validation set

	precision	recall	f1-score	support
hang_loose	0.98	0.98	0.98	53
paper	1.00	0.96	0.98	52
rock	0.98	1.00	0.99	52
scissors	0.96	0.98	0.97	53
accuracy			0.98	210
macro avg	0.98	0.98	0.98	210
weighted avg	0.98	0.98	0.98	210

Figure 21: Classification report of the tuned algorithm based on the validation set

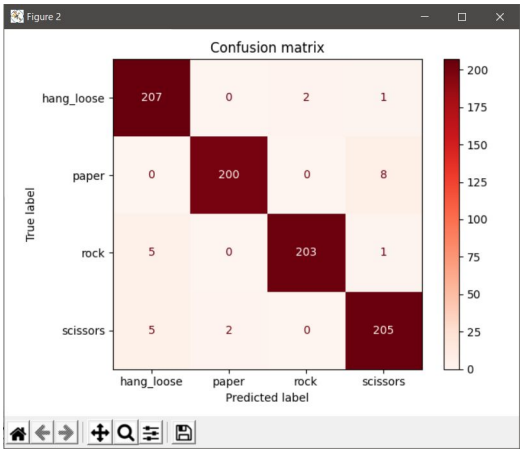

Figure 22: Confusion matrix of the tuned algorithm based on the validation set

From the plots shown above in figure 20 the tuning didn't do too much to the overall working of the algorithm. From the classification report it seems to perform slightly worse. However, from the learning curve it shows that the algorithm does perform marginally better on the validation set compared to before the tuning process.
With the way the hyperparameters are measured it is possible that a local optimum is found instead of a global. This means there might be better hyperparameter options. However, finding the global optimum would be relatively difficult. This could be solved by using the randomizedsearchcv instead of the gridsearchcv. However, although randomizedsearchcv has a higher chance of finding the global optimum because the dataset is relatively small with a limited number of features, the gridsearchcv should perform relatively well.

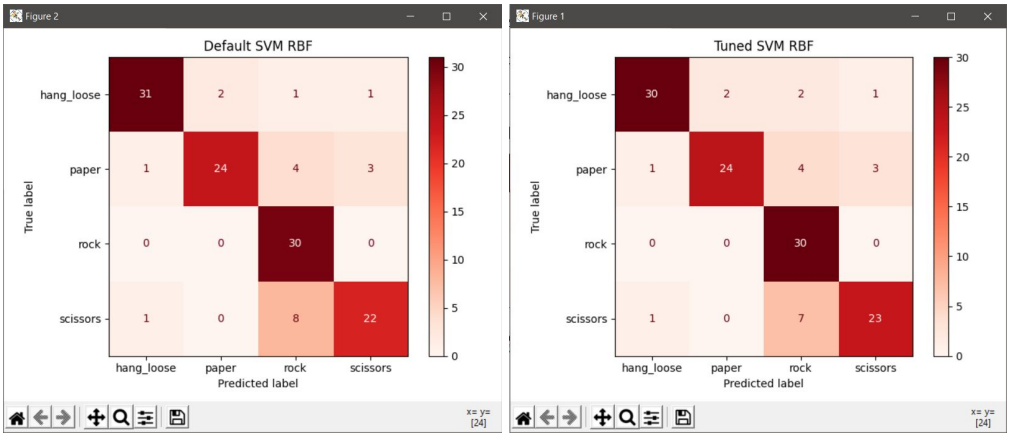

Figure 23: Comparison performance of algorithm on the testing set using Confusion matrix

	precision	recall	f1-score	support
hang_loose	0.94	0.89	0.91	35
paper	0.92	0.75	0.83	32
rock	0.70	1.00	0.82	30
scissors	0.85	0.71	0.77	31
accuracy			0.84	128
macro avg	0.85	0.84	0.83	128
weighted avg	0.86	0.84	0.84	128

Figure 24: Default algorithm C=10, gamma="scale"

	precision	recall	f1-score	support
hang_loose	0.94	0.86	0.90	35
paper	0.92	0.75	0.83	32
rock	0.70	1.00	0.82	30
scissors	0.85	0.74	0.79	31
accuracy			0.84	128
macro avg	0.85	0.84	0.83	128
weighted avg	0.86	0.84	0.84	128

Figure 25: Tuned algorithm C=6, gamma=0,149

Figure 23 shows the difference between how well the algorithm performed on the test set. For both the tuned and default algorithm. From this it is possible to see that the tuned algorithm works slightly better. However, there will most likely not be any noticeable difference when deployed. The same can be seen in figure 25 where there is also not a lot of difference in the classification report.

From these results it is also possible to see that the performance drops from 0.96+ down to 0.74+ for the tuned model. Although this ovefitting error is expected because the hyperparameters are tuned for the validation test. Some of the images in the test set might also be less than ideal because the angles might be larger than 45 degrees.

4. Deploy and Test

4.1. Deployment

The tests from the training chapter 3.2 shows that the model performs relatively well and can be deployed for further testing. To deploy the system a short program has been created. This software can be found back in appendix 1. The code uses JobLib to load the model and displays the image that is taken, how well it is able to segment the foreground from the background shown in figure 26A and the image with the algorithm's prediction and confidence level. The prediction with the highest confidence is displayed in green shown in the figure 26B below.

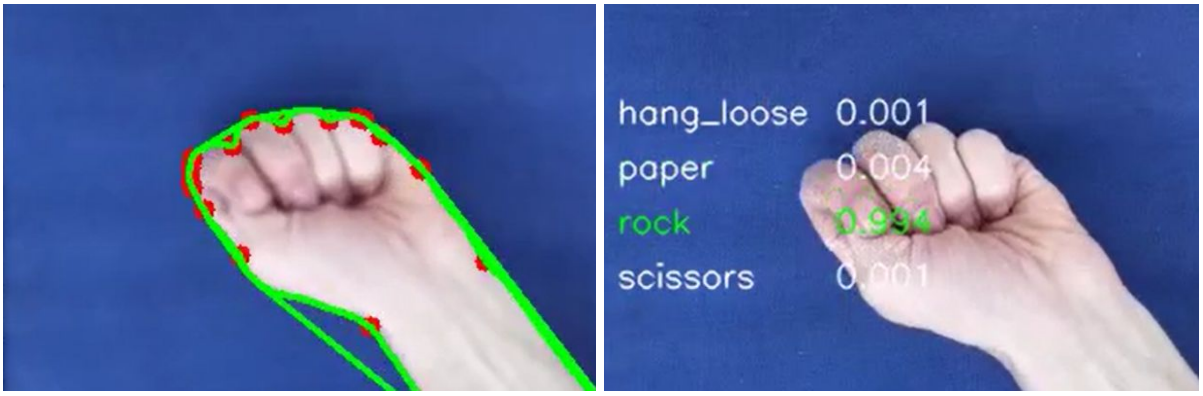

Figure 26: Display segmentation and prediction of the algorithm.

4.2. Testing

The performance of the system is tested as follows. After deploying for each hand gesture the pitch, the roll and the yaw angles are individually tested. For each possible rotation the angles from -60° to 60° is tested with increments of 15°. The tables below show for which angles the different hand gestures predicted correctly. With this data it is possible to see if the 80% accuracy is achieved for the different hand gestures.

The test data is collected by creating a video of all the hand gestures from different angles. The capture program made a video of the entire screen however, the code used to test the algorithm was as in appendix 1. Each angle is shown for 5 seconds before moving on to the next one. The angles that are measured are from -60° to 60°. This was chosen because in the requirements the system needed to be able to get from -45° to 45° degrees, by going 15° further it is easier to check if this is achieved. The steps between each angle will be 15° because otherwise it would be too difficult to differentiate between the angles. The hand movement is limited to rotation of the wrist.

Test results

The angles for the table below are defined as follows. The Pitch is the rotation of the hand from left to right, the Roll is rotation of the hand from back to front and the Yaw is the rotation of the hand from side to side. In the table the percentage on how confident was on predicting the class based on the angle. The table also shows in colour if the algorithm chose the correct gesture.

Table 1:
Pitch angle test

Hand Gestures	-60°	-45°	-30°	-15°	0°	15°	30°	45°	60°
Loose hand	28%	48%	66%	77%	96%	95%	86%	72%	-
Paper	99%	99%	99%	99%	100%	100%	99%	96%	86%
Rock	98%	99%	99%	99%	100%	100%	99%	99%	99%
Scissor	69%	88%	91%	95%	95%	97%	97%	92%	86%

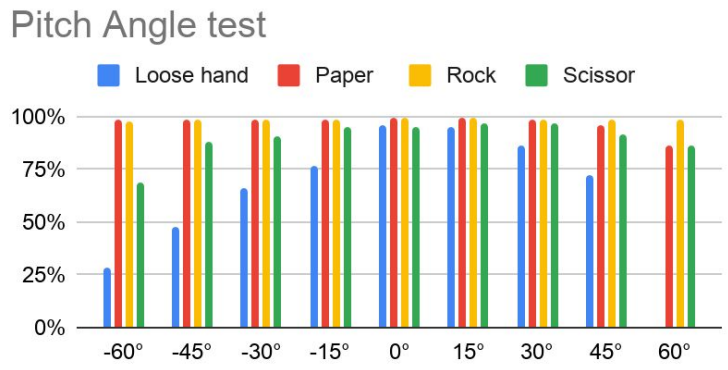

Figure 27: Graph of the Pitch Angle test

Table 1 and the corresponding graph in figure 27 show that for Paper, Rock and Scissors are all detected with an accuracy of at least 69%. However, the prediction confidence from the Loose hand drops significantly when the angle changes. However, although the confidence drops it still predicts correctly.

Table 2:
Roll angle test

Hand Gestures	-60°	-45°	-30°	-15°	0°	15°	30°	45°	60°
Loose hand	1%	52%	54%	77%	73%	60%	62%	72%	88%
Paper	81%	92%	93%	98%	99%	99%	98%	84%	51%
Rock	99%	98%	99%	99%	99%	99%	99%	99%	99%
Scissor	95%	90%	94%	92%	96%	94%	92%	82%	-

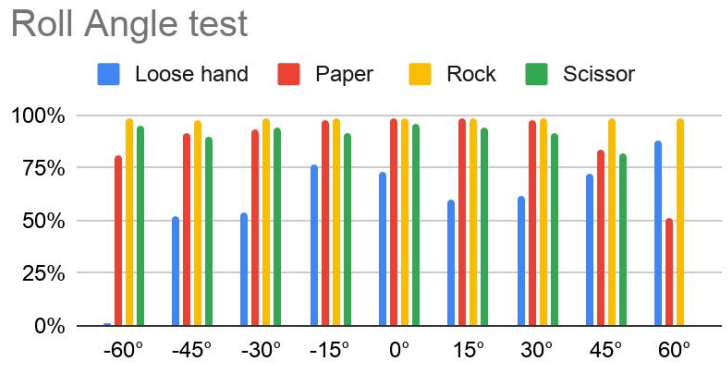

Figure 28: Graph of the Roll Angle test

Table 2 and the corresponding graph in figure 28 show similar results to the first test. The confidence did go down with the lowest confidence level of 51%. However, on average it seems about the same. The prediction confidence of the Loose hand however, drops even further compared to the previous test going as far as 1% on its worst guess. This is however, only at the 60° angle which is outside the requirements.

Table 3:
Yaw angle test

Hand Gestures	-60°	-45°	-30°	-15°	0°	15°	30°	45°	60°
Loose hand	-	13%	13%	15%	47%	72%	73%	72%	52%
Paper	-	98%	99%	99%	96%	92%	91%	90%	97%
Rock	-	99%	99%	99%	99%	99%	99%	98%	98%
Scissor	-	94%	94%	92%	94%	96%	91%	93%	94%

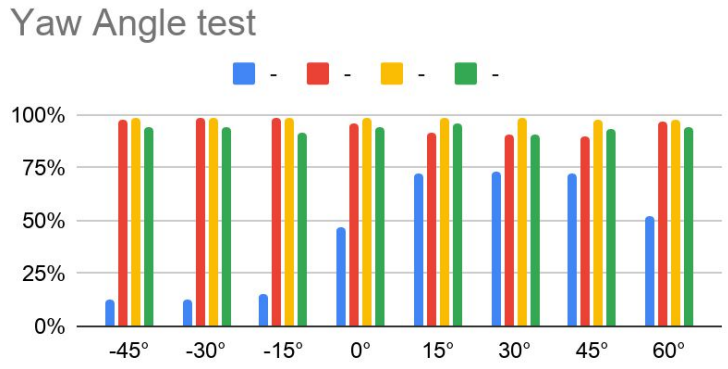

Figure 29: Graph of the Yaw Angle test

Table 3 and the corresponding graph in figure 29 show similar results to the other test for the Paper, Rock and Scissors. However, loose hand prediction confidence and accuracy went down further. Instead of detecting loose hand the algorithm predicted scissors.

Conclusion Test

From the test results it appeared that the distance between the fingers had a large influence on how well the algorithm performed.
In most angles the algorithms had a high accuracy and prediction confidence. Only loose hand lacked behind. Loose hand was usually confused with either rock or scissors. We think this is because of how the features are created. We mainly use features that are based on the surface area and on the hull convexity. So when an angle becomes too big the side of the loose hand will look similar if not the same as the scissors or rock.
So to improve this more and better features need to be created to allow the algorithm to differentiate the different classes.

5. Conclusion

5.1. Results of the project

The project was divided into four main parts. These parts were: the finding of the features used in the project, which model was used for this project, which training method was used for this project and the testing of the algorithm.

The first part finding the features was done in chapter 2. Here it was decided to use the following features: the area, the perimeter, the extent and the solidity. These were chosen because from the scatter plot and box plot they came out as good ways to differentiate between the different classes.

The second part choosing the model was done in chapter 3. Here it was decided to use the SVM model with the RBF kernel. This was chosen over the other options SVM in Linear, K-Neighbour and Decision tree classifier because the learning curve of the SVM with RBF kernel showed the most promise. The overall accuracy was high and the distance between the accuracy on the training set and validation set was relatively low compared to the other options. Although SVM with a linear kernel performed also pretty well the time to train it felt quite long which is not ideal for something that needs to be prototyped quickly.

The third part training the model was also done in chapter 3. Here it was decided to use the GridSearchCV function to check which HyperParamater was the best for this model with the given data set. After tuning there wasn't much difference between the default parameters and the tuned ones. However there was a slight improvement. There was also an option to use the RandomizedSearchCV function; this function is faster and has a smaller chance to get stuck in a local optimum. However, because of the size of the project implementing the GridSearchCV was easier.

The last part was testing the deployed system; this was done in chapter 4. Here it was decided to make a video to monitor the different angles of the different hand gestures. From this we concluded that the accuracy was higher than the 80% for Paper, Rock and Scissors however, Loose hand lacked behind. It is expected that it was mainly caused by a lack of features that can properly differentiate the class. Because the features that are currently used mainly use the area of the object which changes quite drastically when rotating the hand.

That said we are quite pleased with the results. If we had to do it again with more time. We would have gathered more data for the different angles. With this data we would find better and more features to differentiate the classes. After doing this it is expected that the program would be able to easily detect all the different classes with an accuracy of 90%.

5.2. Overview of the project

To ensure that the project went as smoothly as possible the general machine learning project checklist from the book (Géron, 2019) has been used. This list contains the following elements.

1. Frame the problem
2. Get the data
3. Explore the data to get insight
4. Prepare the data
5. Explore different models
6. Fine tune the models
7. Present the solution
8. Launch monitor and maintain the system

To frame the problem we wrote chapter one Problem statement here the problem and the limitations were defined. By doing this the problem was framed.
The data was acquired by using the same camera that was going to be used for the deployed system. However, because of Corona it wasn't possible to get the data from all our hands meaning that the system was eventually only trained on one of our members.
The data was also explored in chapter two different graphs like the boxplot the scatter plots were drawn and compared. From this it was clear which features were going to be useful and which would be unimportant.
In chapter three different models were compared. Although the sklearn cheat sheet showed which order we should try we decided for learning purposes it made more sense to compare the different ones we have learned about during the EVD3 lessons.
Training was done in a relatively standard way using the gridsearchcv though the randomizedsearchcv would have probably been a better choice in actual application. This is because randomizedsearchcv is faster and usually doesn't get stuck in local optimums. The randomizedsearchcv could have also been combined together with the gridsearchcv when a global minimum was found.
As last the system was deployed and tested on the Raspberry Pi, it worked surprisingly well. However, to ensure that the project was within spects some tests were made.
And after that the system was ready to be deployed.
So in generals lines the checklist has been followed. Which allowed for a rather smooth ML experience. However, because the checklist is quite general some of the steps could have been executed with more detail.

5.3. Takeaways from the project

The biggest thing that came forward from doing this project was how important lighting is for reliable data acquisition. The moment the lighting changes too much it can become almost impossible to properly segment the foreground form the background. Besides that it became quite early on how important it is to have a large amount of data. With more data the algorithm became more reliable with its predictions.
Lastly, the important thing was finding good features. To find good features it was important to look at objects differently. Not only what is an object but also how do you know it is this object.

Figure List

Figure 1: Visual representation on how the hand gesture look 6
Figure 2: The amount of training data per hand gesture. 8
Figure 3 A and B: Distribution of the features over the data set Scatter plot area against perimeter 8
Figure 4: A and B: The scatter plot area against aspect ratio and area against the extent 9
Figure 5: A and B the scatter plot area against solidity and perimeter against aspect ratio 9
Figure 6: A and B the scatter plot perimeter against extend and perimeter against the solidity 10
Figure 7: A and B the scatter plot aspect ratio against extent and expect ratio against solidity 10
Figure 8: A and B the scatter plot extent against solidity and the boxplots of area and perimeter 10
Figure 9: A, B and C: The boxplots for aspect ratio, extent and solidity 11
Figure 10 Heatmap correlation between the different features. 11
Figure 11: Cheat-sheet which algorithm should be chosen 14
Figure 12: Learning curves of SVM linear and RBF with default hyperparameters 15
Figure 13: Learning curves of DTC and K-neighbour with default hyperparameters 15
Figure 14: The workflow for this project 16
Figure 15: Standard way grid cross validation. 16
Figure 16: Classification_report seed 48 and 86. 17
Figure 17: Learning curve of the default algorithm 17
Figure 18: Classification report of the default algorithm 18
Figure 19: Confusion matrix of the default algorithm 18
Figure 20: Learning curve of the tuned algorithm based on the validation set 19
Figure 21: Classification report of the tuned algorithm based on the validation set 19
Figure 22: Confusion matrix of the tuned algorithm based on the validation set 19
Figure 23: Comparison performance of algorithm on the testing set using Confusion matrix 20
Figure 24: Default algorithm C=10, gamma=”scale” 20
Figure 25: Tuned algorithm C=6, gamma=0,149 20
Figure 26: Display segmentation and prediction of the algorithm. 22
Figure 27: Graph of the Pitch Angle test 23
Figure 28: Graph of the Roll Angle test 24
Figure 29: Graph of the Yaw Angle test 24

References

1. (n.d.). sklearn.pipeline.Pipeline — scikit-learn 0.23.2 documentation. Retrieved September 24, 2020, from http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

2. (2018, December 20). (My) Machine Learning Workflow. Epistemic status ... - Medium. Retrieved October 12, 2020, from https://medium.com/datadriveninvestor/my-machine-learning-workflow-7576f7dbcef3

3. (n.d.). Choosing the right estimator — scikit-learn 0.23.2 .... Retrieved October 9, 2020, from http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

4. (n.d.). Plotting Learning Curves — scikit-learn 0.23.2 documentation. Retrieved October 9, 2020, from http://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html

5. (n.d.). Plot classification probability — scikit-learn 0.23.2 .... Retrieved October 5, 2020, from http://scikit-learn.org/stable/auto_examples/classification/plot_classification_probability.html

6. Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. Sebastopol, Canada.: O'Reilly Media.

CODE appendices

Appendix 1 Deployment code

```
from imutils.video import VideoStream
import numpy as np
import time
import os
import cv2 as cv
import joblib
from fetch_data import fetchDataFromImage


if __name__ == '__main__':
   model_path_test = 'C:\\Users\\renzo\\Downloads\\gesture_data\\model\\finalized_model.sav'
   local_data_path_test = 'C:\\Users\\renzo\\Downloads\\gesture_data\\test'
   loaded_model = joblib.load(model_path_test)
   gestures = ['hang_loose', 'paper', 'rock', 'scissors']

   frame_size = (320, 240)
   vs = VideoStream(usePiCamera=False, resolution=frame_size).start()
   time.sleep(1.0)

   while True:
       frame = vs.read()
       if frame is None:
           continue

       k = cv.waitKey(1) & 0xFF
       # quit the program when q or esc is pressed
       if k == ord("q") or k == 27:
           break

       # save the current frame for analysis
       elif k == ord("t"):
           test_path = os.path.sep.join([local_data_path_test, "{}.png".format(int(time.time()))])
           print("[INFO] saving frame: {}".format(test_path))
           cv.imwrite(test_path, frame)

       gesture = fetchDataFromImage(frame)

       # Trying to figure out how to avoid crashing when nothing is detected
       if gesture.data is not None:
         predictionScore = loaded_model.predict_proba(gesture.data)

         text1 = gestures[0]
         text2 = gestures[1]
         text3 = gestures[2]
         text4 = gestures[3]

         valueList = [round(predictionScore[0][0], 3), round(predictionScore[0][1], 3),
                      round(predictionScore[0][2], 3),  round(predictionScore[0][3], 3)]

         text11 = str(valueList[0])
```

```
        text22 = str(valueList[1])
        text33 = str(valueList[2])
        text44 = str(valueList[3])

        colourHigh = (000, 255, 000)
        colourLows = (255, 255, 255)

        colourList = [colourLows, colourLows, colourLows, colourLows]
        colourList[valueList.index(max(valueList))] = colourHigh

        cv.putText(frame, text1, (10, 375), cv.FONT_HERSHEY_SIMPLEX, 0.5, colourList[0], 1, cv.LINE_AA)
        cv.putText(frame, text11, (110, 375), cv.FONT_HERSHEY_SIMPLEX, 0.5, colourList[0], 1, cv.LINE_AA)

        cv.putText(frame, text2, (10, 400), cv.FONT_HERSHEY_SIMPLEX, 0.5, colourList[1], 1, cv.LINE_AA)
        cv.putText(frame, text22, (110, 400), cv.FONT_HERSHEY_SIMPLEX, 0.5, colourList[1], 1, cv.LINE_AA)

        cv.putText(frame, text3, (10, 425), cv.FONT_HERSHEY_SIMPLEX, 0.5, colourList[2], 1, cv.LINE_AA)
        cv.putText(frame, text33, (110, 425), cv.FONT_HERSHEY_SIMPLEX, 0.5, colourList[2], 1, cv.LINE_AA)

        cv.putText(frame, text4, (10, 450), cv.FONT_HERSHEY_SIMPLEX, 0.5, colourList[3], 1, cv.LINE_AA)
        cv.putText(frame, text44, (110, 450), cv.FONT_HERSHEY_SIMPLEX, 0.5, colourList[3], 1, cv.LINE_AA)
    cv.imshow("Frame", frame)
    time.sleep(0.1)
```