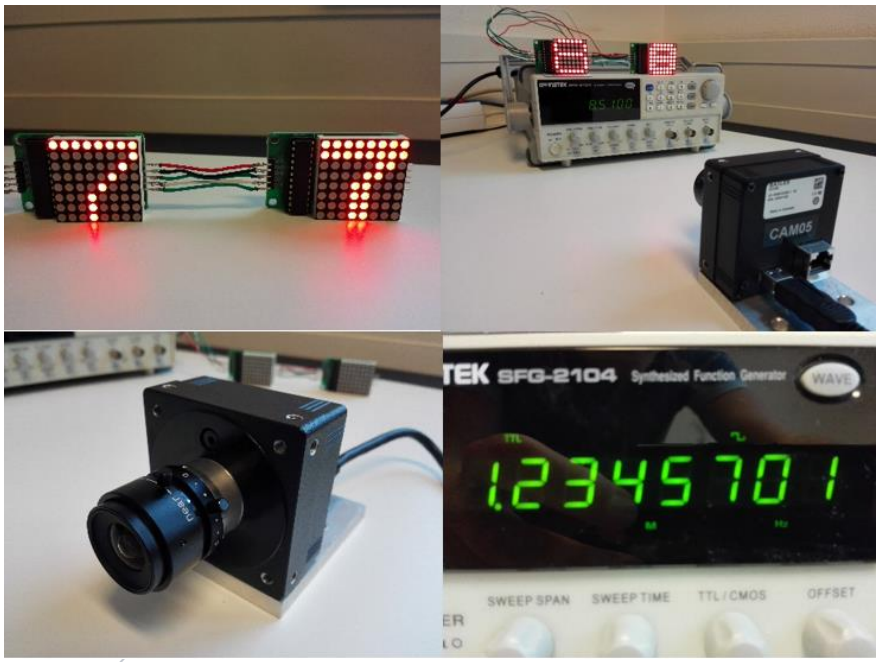
Automated number recognition software for digital LED displays

Graduation report

17-12-2015



DE HAAGSE HOGESCHOOL

Graduation counselors from the HHS:

Ing. T.J. Koreneef & Ir. A Le Mair

CARYA AUTOMATISERING

Graduation counselor from Carya Automatisering:

P. Batenburg

Author: Mark Schoneveld

Foreword

Before you lies the report “Automatic number recognition software for LED displays”. This is my report for my graduation internship for the study Mechatronics at The Hague University of Applied Sciences in Delft (HSS).

This report is written for Carya Automatisering and the HHS. It will illustrate the phases the project went through and presents the end result of the project that started the 26th of September 2015 and ended on the 17th of December 2015.

I would like to thank Peter Batenburg from Carya Automatisering for all the guidance, help and feedback on the project. Of course I would also like to thank all other employees of Carya that helped when it was necessary.

From the HHS I would like to thank Theo Koreneef for all the guidance during the project and Fidelis Theinert at the very beginning of the project when he helped me to formulate a good project assignment.

At last I would like to thanks my family and friends or all the support when it was needed.

I wish you pleasure with reading the report.

Mark Schoneveld, Nieuw-Vennep, 17-12-2015

Table of Contents

Summary

This project is about making making number recognition software for Carya Automation in Delft. The software must recognise numbers with different fonts from±

- 7 segment LED displays
- 5x7 dot matrix LED displays
- 8x8 dot matrix LED displays

The research question is “What is the best method for recognising numbers for this project?”. The demands of Carya were:

Nr	Requirements
1	The software needs to be able to recognise the numbers zero to nine from displays defined below.
	Displays:
2	- All displays are LED displays without backlight.
3	- The displays are 5x7 dot LED matrix, 8x8 dot LED matrix and 7 segment LED
4	A.
5	The software preferably makes use of machine learning to recognise numbers, but if another method is more reliable, it may also be used.
6	The student needs to make his own database with training and test images.
7	The software has to recognise numbers when the camera is angled with 45 degrees, rotated around the vertical y-axis and the horizontal x-axis that stands perpendicular to the display.
8	The lighting of the environment may be adjusted, but as least as possible.
9	The user needs to be able to set up the test environment within 5 minutes, including the lighting, normalisation of the image and setting the Region Of Interest (ROI) within the image.
10	The software needs to detect numbers as fast as possible. There is no time limit since Carya cares more about the method for recognising the numbers than the time it takes to do so, but a new prediction every 5 seconds is preferred.
11	The recognised numbers need to be shown on a computer screen, together with the percentage of how certain it is that the number is correctly recognised. The image does not need to be saved.
12	All documentation needs to be in English

Research

Machine learning

The research showed that the best way to recognize number was with non-linear regression methods, because it was not known whether the test data could be predicted correctly with linear regression. This means that the learning method can make a non-linear prediction line.

Classification methods were no option, because these return discrete values. Namely the predicted class. Regression methods give a percentage of how certain it is with its prediction.

The learning method also needs to be an eager learning method and not lazy. This means that the learning is done before the prediction starts instead of during the learning. Because of this, eager learning methods are faster than lazy learning methods during the prediction, but lazy methods have no training time before the prediction starts.

The chosen learning method is a Multi Layer Perceptron (MLP)

Pre-processing

The most important conclusions that was drawn, was that the better the input for the MLP was, the more basic the MLP could be without other kinds of modifications. The pre-processing contains:

- Thresholding, to filter out the numbers from the image
- Blurring, to remove noise and attach parts of numbers that lie close to each other
- Dilation, to attach the remaining parts of numbers that need to make a connection
- Erosion, to remove noise than could be dilated too and to smoothen the edges of the dilated numbers
- Connected Component Labelling, for detecting what pixels are connected to each other
- Skeletonization, so only the basic contours of the numbers remain
- Rotation, so all numbers stand straight up
- Cutting out all numbers from the images, so the numbers can be used for training the network or as input for the network that needs to be predicted.

Detailed design

The number recognition software can be divided in three main parts:

- Image acquisition
- Image per-processing
- Learning and predicting numbers

Realisation

In the realisation phase, the code is programmed and the test environment is made. The test environment can be divided in the following parts:

- Camera: which camera is used?
- Displays: which displays are used?
- User set-up: what does the user need to set-up before the software can be started?

The following camera and lens are used for the project:

- Camera: the Basler A312fc.
- Lens: a Navitar NMV-5WA.

The main advantage is that the lens has a manually changeable shutter and focus. Unfortunately the lens is also a wide angle lens, which causes dish eye/ barrel distortion in the image. Because the pre-processing already filters out a lot of the data from the image by cropping it, most is the barrel distortion is also filtered out.

The displays are a green 7 segment LED display from a function generator and two red dot LED matrices of the size 8x8. These can also produce numbers from 5x7 matrices.

For setting up the test environment, see the use case diagram in appendix D4.

Testing

For testing if the software meets all requirements, the following tests have been done.

Nr	What
1	Has machine learning been used in the recognition software?
2	Can the software recognize the numbers zero to nine from 5x7 dot matrix LED displays, 8x8 dot matrix LED displays and 7 segment LED displays?
3	Can the software recognize the numbers zero to nine from all the above displays when the camera is rotated with 45 degrees?
4	Can the software recognize the numbers zero to nine from all the above displays when the camera is positioned between 15-30 cm?
5	Can the software recognize numbers without light adjustments?
6	Can the test environment be set up within 5 minutes?
7	Does the software show the original image, together with predicted value of the number and a percentage of how certain it is that the number has that value?
8	Can the recognition software check numbers every 5 seconds?

The software has passed every test, but the last requirement has not been checked yet because of the time that was left for the project. This was the least important requirement for Carya. They cared more about the proof of concept than the amount of time it took to recognise the numbers. The software recognises 98.1% of all test images.

Conclusion

The answer to the research question “What is the best method for recognising numbers for this project?”, is a non-linear learning method, the MLP, that makes use of images that have been pre-processed to create an as good as possible input for the MLP.

Introduction

Carya Automatisering VOF is a company located in Delft. They are specialised in automating processes in many kinds of industries. The company is founded in 2003 and at the moment there are five persons leading the company and three employees.

A few years ago, a client wanted Carya to automate a number recognition process for them. The client frequently checked how much electromagnetic radiation different kinds of displays could handle till numbers shown on the displays were impossible for humans to read. This process was done by pointing an electromagnetic radiation resistant camera towards the display and sending the resulting images to a computer screen. This computer screen was then checked by some people sitting in front of it and they wrote down for what frequencies the display would show signs of electromagnetic interference.

The project was to automate this. Carya wrote recognition software with LabVIEW for seven segment LED and LCD displays. Afterwards the customer also wanted numbers that where presented on other kinds of displays to be recognised, but it turned out not to be feasible in the time allocated to the project.

The goal of this graduation project is: “Write recognition software that is able to recognise numbers with different fonts that are presented on digital displays”.

To reach this, the following question needed to be asked: “What is the best method for recognising numbers for this project?”. This project is a proof of concept. Carya wants to use the recognition method for recognising numbers and/or other objects in future vision-projects. Now follows an outline of this report.

- The first chapter discusses the assignment.
- The second chapter describes the research phase.
- The third chapter shows the detailed design of the software. This is mainly done with drawings and different kinds of SysML-diagrams.
- The fourth chapter describes the realisation of the software and the test environment.
- The fifth chapter is about the tests that have been done with the software.
- The sixth chapter gives a summary of all products that are finished and can be delivered to Carya. At the end the answer will be given whether the main goal has been reached.
- The final chapter contains the conclusions of the project and the recommendations for Carya about further implementation of the recognition software.

1. Assignment

The scope of the assignment has been divided in multiple parts:

- Goals
- Requirements
- Boundaries
- Deliverables

These are described in this chapter and at the end of the report it is checked whether everything within the scope has been done and implemented.

1.1 Goals

The goal of the project is to write software that is able to recognise numbers with different fonts from different kinds of displays. To reach this goal, research is done to find the best recognition method and find the best way to pre-process every image.

1.2 Requirements

Carya has set a number of requirements for this project. These requirements can be divided into three main types:

- Recognition software
- Test environment
- Result analysis

These requirements can be found in Table 1.

Table 1:Requirements Carya has set for the project

Nr	Requirements
1	The software needs to be able to recognise the numbers zero to nine from displays defined below.
	Displays:
2	- All displays are LED displays without backlight.
3	- The displays are 5x7 dot LED matrix, 8x8 dot LED matrix and 7 segment LED
4	A.
5	The software preferably makes use of machine learning to recognise numbers, but if another method is more reliable, it may also be used.
6	The student needs to make his own database with training and test images.
7	The software has to recognise numbers when the camera is angled with 45 degrees, rotated around the vertical y-axis and the horizontal x-axis that stands perpendicular to the display.
8	The lighting of the environment may be adjusted, but as least as possible.
9	The user needs to be able to set up the test environment within 5 minutes, including the lighting, normalisation of the image and setting the Region Of Interest (ROI) within the image.
10	The software needs to detect numbers as fast as possible. There is no time limit since Carya cares more about the method for recognising the numbers than the time it takes to do so, but a new prediction every 5 seconds is preferred.
11	The recognised numbers need to be shown on a computer screen, together with the percentage of how certain it is that the number is correctly recognised. The image does not need to be saved.
12	All documentation needs to be in English

1.3 Boundaries

To prevent the assignment from getting to big, certain boundaries have been set. These are shown in Table 2.

Table 2: Boundaries of the project

Nr	Boundaries
	The program does not have to recognise numbers from pictures fully automated. The user is allowed to predefine:
1	- the distance from the camera to the display
2	- the regions of interest, per number, where the user wants to find a number
3	- where the decimal seperator is located
4	- how many characters there are to be recognised
5	- what the colour of the number is.
	However, boundary 1 to 5 may be automated if there is time to do so.
6	The software will be written in Python 2.7.10 with the "CV2", "numpy", "skimage", "math", "random", "glob" and "os" library.
7	The software will recognise numbers from Appendix A presented on displays defined in Table 1.
8	The minimum and maximum distance between the camera lens and the display is 15 to 30 cm.
7	The recognition software will work with the camera that has been used for all tests.

1.4 Deliverables

The goal of this project is to make recognition software that is able to recognise numbers from different displays with different sizes and different fonts. The deliverables are:

- The recognition software
- A validation report of the recognition software

The validation report can be found in chapter “5. Testing”.

2. Research

To find a way to recognise the numbers from displays, the research was focussed on the following two areas:

- What is the best method for recognising the numbers in the image?
- What kind of pre-processing[1] needs to be done to the image for above methods before number recognition can be applied?

An explanation of the research of each point will be shown in this chapter, together with the outcome. First, the method for recognising the numbers will be dealt with. After this is chosen, it can be determined what kind of input the recognition method needs and what kind of pre-processing needs to be done.

2.1 Method for recognising the numbers

Carya set the requirement (requirement 5, Table 1) that machine learning needed to be used, unless a better method was found. After research it became clear that there are two main methods for character recognition:

- Template matching
- Machine learning

What they have in common is that they both look at characteristics of the object that needs to be recognised. In this case it is an image of a number. The difference lies in how they do this.

With template matching, an image is compared to a database of other images with possible appearances. In the case of this project, an image of an unknown number would be compared to many images of numbers from 0 to 9 with different fonts, one by one. The most similar image in the database is expected to have the same value as the number that is presented to the software. This is used in many programs for number recognition, but in many cases it only works good when it expects a number with a specific font that is also in the database. Otherwise it is possible that the software does not recognise the number or worse, gives a wrong estimation of the value of it.

When the font is not known, machine learning could be used. A table with advantages and disadvantages of template matching and machine learning is presented below (Table 3).

Table 3: Advantages and disadvantages of template matching and machine learning

	Template matching	Machine learning
Recognition method	Compares the input image to many images in a database, one by one.	Compares the input image to one or more models, created with images from a database.
Advantages	Has no training time.	Can learn patterns
		Is faster than template matching with predicting an outcome
Disadvantages	Only works well when it is known what the object in the image looks like.	Needs to train one or more models. This could take a lot of time.
	Needs to process every image in the database every time there is some new input.	

[1] Pre-processing: making adaptions to an image, so the software can further use the data from it. Like thresholding the image, cropping or removing noise.

Machine learning has the big advantage that it can learn what each number looks like. This is done by looking at all numbers in the database with different fonts, rotations, sizes, whatever a project requires. It learns what a number looks like and does not compare it to a rigid template.

This project is a proof of concept for software being able to recognise numbers presented on displays. It needs to be as fast as possible during the recognition process and in real life it may not always be known what font is used for the numbers. Because of this, machine learning is the best option for the project.

2.1.1 Types of machine learning

A machine learning algorithm is able to learn from its inputs and predict something with the processed information from previous obtained inputs.

To learn an algorithm what a number looks like, it will need examples that are extracted from a database. The database can differ in size. This depends on what the learning method needs and how similar the objects are that need to be recognised.

Machine learning contains many kinds of methods. These can be classified into four main groups:

These main learning methods all have different ways of learning. The most important ones are described in Table 4.

Table 4: Advantages and disadvantages of the main learning methods

	Supervised learning	Unsupervised learning	Semi-supervised learning	Reinforcement learning
Advantages	Knows when an answer is wrong Knows how wrong an answer is	Can sometimes find unsuspected patterns No need for labelling data (could save time)	Can label clustered data automatically Only a small amount of data needs to be	Knows it when an answer is wrong
Disadvantages	All data needs to be labelled	Does not know when an answer is wrong	Does not know when an answer is wrong When unlabelled data looks a lot like some labelled data with another value, the unlabelled data could be clustered incorrectly	Does not know how wrong an answer is All data needs to be labelled

Before the choice can be made between these four types of learning, two questions need to be considered:

- continuous or discrete values?
- eager or lazy methods?

Continuous or discrete value

The result from every of the above method can be given in two different forms. The result can be a *continuous* value or a *discrete* value. A continuous value can be any value and a discrete value can only be a specific value. For example, the separation of numbers in supervised learning, semi-supervised learning and reinforcement learning can be done in two ways.

- Classification (discrete)
- regression (continuous).

With classification a separation line is “drawn” between the plotted data of the different numbers and the data is classified. By looking at what side the data lies, it can be predicted to what class a number belongs. With regression, a predictive line is calculated with which it can be predicted how likely it is that a number has a specific value. In Figure 1 (Rossant, 2014), a classification and a regression is showed. The red lines are linear and therefore of the form ”$y = ax + b$”.

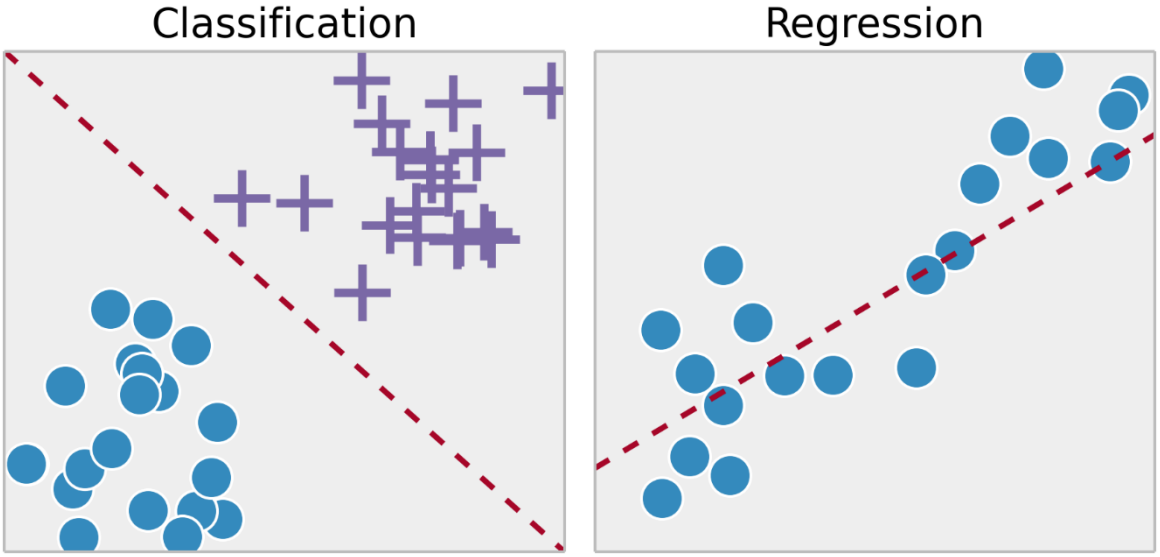


Figure 1:). [A red classification line in the left image and a red regression line in the right image], Reprinted from GIthub website, by iPython-books, retrieved from http://ipython-books.github.io/featured-04/

Carya demands to get a percentage of how big the chance is that a number is estimated correctly (Requirement 12, Table 1), so the outcome needs to be a continuous value. Regression is chosen for the project.

Regression can be linear and non-linear. In this project, different numbers with different fonts are presented to the learning algorithm. Non-linear regression methods can form linear regression lines, but not the other way around. Since it is not known whether the data from different numbers can be separated or predicted by a linear function, a non-linear regression learning method is the best choice to start the project with. At the end of chapter 5 an evaluation is done whether a non-learning method was necessary or that a linear learning method would have been sufficient.

Eager of lazy methods
The last big choice that needed to be made was whether the learning method is *eager* or *lazy*.

Eager methods search for the best general function to predict a target by looking at all training data before it starts. A lazy method only starts to search for a general function as soon as it needs to do so. This makes lazy methods a lot slower during the recognition process. Lazy methods also use more memory because they have to process all the images in the training set, every time the recognition software starts to recognise something. The advantage of these methods is that they are able to make multiple different prediction models per learning round Instead of one model for all cases. Therefore these methods can find a the best prediction model for the specific input.

This does not mean that eager methods are not accurate. When there is a good trainings set, the results can be almost as good as with lazy methods, if not equal. The advantages and disadvantages of both methods are presented in Table 5.

Table 5: Advantages of eager and lazy learning methods

	Eager learning	Lazy learning
Advantages	Fast during recognition	Better accuracy in most cases
	Uses less memory	No learning process before the recognition starts
Disadvantages	Needs a relatively long time to learn from the database compared to lazy learning methods	Slower during recognition, because it needs to make a prediction model every time it wants to recognise a number
	Less accuracy in most cases	Uses more memory

The digital displays in this project will present numbers with high contrast to the environment and pre-defined shapes, so with the trainings set it is possible to make a good prediction model. Because Carya wants the software to be as fast as possible during the recognition of numbers (Table 1, requirement 10), the best choice for this project is an eager learning method.

2.1.2 Choosing a learning method
The learning method needs to be eager and the result needs to be a continuous value. Now that these choices are made, it can be decided which of the four learning method is most appropriate:

- unsupervised learning
- semi-supervised learning
- supervised learning
- reinforcement learning.

Unsupervised learning
With unsupervised learning, there is no labelled data. Labelled data is not always available because the labelling can be very time consuming. Standard unsupervised learning method make use of classification and are therefore not suitable for this project, as stated before with classification methods. Nevertheless, there are some types that can calculate how likely it is that a number belongs to a certain cluster: soft or fuzzy clustering.

A big disadvantage of clustering is that because the software does not know the value of each number in the database. It may be the case that the same numbers with different fonts have bigger

differences than different numbers. Than the classification could go completely wrong. For example, a blocked three and nine look a lot like each other, but a smaller three can be very different (see Figure 2).

Figure 2: Two threes that show significant differences in shape and a three and a nine that share the same shape wide shape in a block form

It would be plausible that these are not classified correct, because the right three and the nine share almost the same shape. The three on the left however is a lot thinner, more round and a bit thicker. The threes could be put in different clusters and there is no way the software can know this.

Unsupervised learning lacks to ability to be steered in a certain direction with setting its variables, but this can also be an advantage. It is possible that because it is not steered in a certain direction, a new unsuspected pattern can be found.

Semi-supervised learning

Semi-supervised learning makes use of clustering, just like unsupervised learning. By looking at the few labelled data in the database and comparing them with other unlabelled data, it can be predicted where the unlabelled data belongs to. With these kind of learning methods, the user does not have to label all clusters afterwards as with unsupervised learning. Nevertheless, the disadvantage is that is does not know when unlabelled data is incorrect, just as with unsupervised learning. When some unlabelled data with a specific number on it looks a lot like some labelled data with another specific number on it, it is possible that all that unlabelled data is classified incorrectly.

If the user want to make a database with some labelled numbers, he has to make sure that the data is labelled correctly. This can be a very time consuming process.

Supervised learning

With supervised learning, a prediction is made and the software gets told if the answer is right or wrong and when it is wrong, it gets told what it should be. It can adjust its parameters in such a way that it learns to recognise numbers by getting the correct answer. A big problem is the labeling of data. This can be time consuming.

Reinforcement learning

These learning methods use labelled databases just like supervised learning methods, but does not get told what the correct answer should be when it is wrong. An disadvantage that result from this is that it can sometimes be slower than supervised learning methods. This is because it need to search for a correct answer instead of just getting the correct answer.

Choice of learning method for the project

A labelled database would not be a problem to acquire, because the numbers on the predefined displays that been used for making images can be set manually and therefore the labelling can be done automatically together with making the image. This can be done by making the image and naming the image after the value presented on the image. (more on this in chapter 3).

Clustering with unsupervised or semi-supervised data could be a possibility, but some numbers may look a lot like each other when presented on the digital displays. This could be a problem for unsupervised learning because it may not find the difference. And for semi-supervised learning this may be a problem concerning the possibility that numbers are labelled wrong after clustering.

Reinforcement learning would be a good choice, just like supervised learning, but the fact that it may be slower in some cases than supervised learning methods, makes supervised learning the best choice in this project.

2.1.3 Supervised learning with non-linear regression

There are many supervised learning methods to choose from that make use of non-linear regression and eager learning methods. But there are two methods however that come up almost everywhere on the internet, in books and in papers.

- Support Vector Machine (SVMs)
- Multi Layer Perceptrons (MLPs)

Although the SVM is a classifier. The SVM that can make predictions with regression is called Support Vector Regression method (SRV).

These are not the only two learning methods. Other methods include for example decision trees and K-nearest neighbour are also called a lot, but the first two methods have already been used in many scientific research to character recognition with much success, sometimes almost 100%. Some of these papers can be found in the sources. In most papers these two methods excel and can compete with each other. What the best method is, is unfortunately not possible to predict. This depends on the training and test data and the machine learning method processes it.

There were no good reasons to choose for or an SVR or an MLP. The way they work differs, but both have shown good results in OCR. The choice fell for an MLP because Carya was already a little familiar with this method and getting to understand MLPs is relatively easy for people from Carya that need to work with it in the future, because it is based on the human brain. An explanation of the MLP is given in Appendix B1.

It appears to be the case that people made combinations of machine learning methods and other mathematical tricks in the learning methods to be able to predict very difficult problems. Like input images with a lot of noise or with numbers a lot of different handwritten numbers. The more irregular the image was, the more complex the machine learning method becomes to handle this.

Numbers on digital LED displays without backlight have very clear contours and much contrast. Therefore the numbers are relatively easy to filter out of images with few noise and clear contours. On top of that, numbers shown on digital displays almost always present clear characteristics so they are easily readable for humans. A relatively easy learning method without extra adjustments would satisfy if the pre-processing of the image is good.

2.2 Pre-processing the image

The pre-processing of the image is very important for the recognition. When the pre-processing is done in a good way, the machine learning method can be kept relatively basic. The pre-processing will consist of the following parts:

- Filtering out the numbers from the image
- Finding the position of the numbers in the image

To know what pre-processing needs to be done, the method for finding the numbers needs to be known. Therefore the acquisition of the images will come first and after that the way of retrieving the necessary information from the image with pre-processing.

2.2.1 Finding numbers in the image

At the beginning of the project, the main idea was to use histograms of the x- and y-axis about the amount of pixels per row and per column to find the numbers. This proved to be a problem in some cases when the image was rotated (more about this in Appendix C). With Connected Component Labelling (CCL) all pixels with a value above zero that are connected can be given a certain value. This works as follows:

Every pixel is checked in the image. It starts with the left upper corner and works its way down to the bottom right corner. When a pixel is black, it skips that pixel. When a pixel is anything else than black, it gets a label.

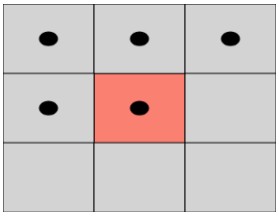


Figure 3: [square 8 connectivity]. Reprinted from Wikipedia website, by DHull003, 2010, Retrieved from https://en.wikipedia.org/wiki/Connected-component_labeling#/media/File:Square_8_connectivity.png

The label it gets depends on the situation. Let's take a look at Figure 3 (Dhull003, 2010). Imagine the red pixel is white. That means it gets a label. If one of the pixels around the red one with a black dot already have a label, the red pixel gets the lowest of them. If there are no other labels to be found around the red pixel, it will get a new label that has not been assigned to another pixel yet.

When the label is assigned, the software continues to pixel right of the red pixel and does exactly the same. This process continues till all pixels are checked and labelled if they are white. The result of this process might look as in Figure 4.

At the same time a pixel is labelled, the software also keep a list with all labelled values that connect with each other. By finding, for example, that two pixels with a three and a seven touch each other and two pixels with a label of three and six, then it can be concluded that pixels thee, seven and six belong to the same object, due to a similarity in the connecting label. The same can be seen from Figure 4 and Table 6.

0	1	1	0	0	2	2	0	0	3	3	0	0	4	4
1	1	1	1	1	1	1	1	0	0	3	3	3	3	0
0	0	1	1	1	1	0	0	0	3	3	3	3	0	0
0	1	1	1	1	0	0	0	3	3	3	0	0	3	3
1	1	1	0	0	1	1	0	0	0	3	3	3	0	0
0	1	1	0	0	0	0	0	5	3	0	0	0	3	3
0	0	0	0	0	6	6	5	3	0	0	7	3	3	3

Figure 4: Dhull003. (2010). Example of an array where connected region labeling is to be carried out. 1 represents the region pixel, and 0 represents the background pixel. Retrieved from https://en.wikipedia.org/wiki/Connected-component_labeling#/media/File:Screenshot-Pixel_Region_(Figure_1).png)

Table 6: [connected label table]. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Connected-component_labeling

Set ID	Equivalent Labels
1	1,2
2	1,2
3	3,4,5,6,7
4	3,4,5,6,7
5	3,4,5,6,7
6	3,4,5,6,7
7	3,4,5,6,7

On the bottom row, there are two labels with a value of six and one with a value of seven. Because these labels touch the pixels with label three, it can be concluded that all labels connecting with label three belong to the same object in an image and are indirectly touching label seven. By creating a connected label table, the pixels that belong to the same object are grouped.

When all white pixels are labelled, the image is processed again all connecting labels will the lowest connecting label value. When this is completely processed, the numbers are separated from each other and the exact region in which every number lies is known. An example of how this might look is given in Figure 5 (Dhull003, 2010).

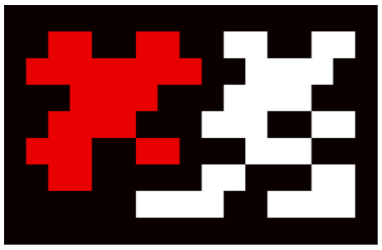


Figure 5: Dhull003. (2010). Result of connected region labeling using two-pass raster scan [Drawing]. Retrieved from https://en.wikipedia.org/wiki/Connected-component_labeling#/media/File:Screenshot-Figure_1.png

To be able to use this method, there are two important features that the image needs to have:

- All images need to be binary, which means that there are only black (value=0) and white (value=255) pixels.
- All parts of the numbers need to be connected, because every loose object is labelled different.

This means that the pre-processing has to lead to the above result. Unfortunately, the number presented on the DOT matrix display and 7 segment display are, as the names already say, segments and dots. These points need to be filtered out of the image and connected to each other.

2.2.2 Filtering numbers out of image

To make the CCL work, all segments that present a number must be connected. But these need to be filtered out of the image first. All displays are LED displays without backlight. This means that all numbers that are presented, are bright light sources. This can be used for filtering out the numbers from the image. When the light source is bright, the aperture of the camera can be closed almost completely (more about this in chapter 4). When the aperture is almost closed, a lot of noise from the background is removed while the numbers are still visible in the image. All steps that are taken in the pre-processing of the image are documented in this chapter with a short explanation why it was used. These steps are:

- Thresholding, to filter out the numbers from the image
- Blurring, to remove noise and attach parts of numbers that lie close to each other
- Dilation, to attach the remaining parts of numbers that need to make a connection
- Erosion, to remove noise than could be dilated too and to smoothen the edges of the dilated numbers
- Connected Component Labelling, for detecting what pixels are connected to each other
- Skeletonization, so only the basic contours of the numbers remain
- Rotation, so all numbers stand straight up
- Cutting out all numbers from the images, so the numbers can be used for training the network or as input for the network that needs to be predicted.

Thresholding

For filtering out the numbers from the image, thresholding needs to be done. When thresholding is used, specific colours are filtered out of an image. There are different ways to process all values above and under a set threshold. Since a black and white image is the best input for the CCL, everything below a specified threshold becomes black and everything above it becomes white.

Figure 6: 8x8 dot matrix LED display and 7 segment LED display

The colour of the LEDs is either green or red and this is filtered out of the image (Figure 6).

Images loaded with the CV2-library, that is parts of the OpenCV-library are using the colour space Red, Green and Blue (RGB). The disadvantage of RGB is that is can be very hard to filter certain colours out of images if they are not 100% red, green or blue. This is because all other colours are always a combination of red, green and blue. For humans it is hard to say what combination of colours is used for, let's say, all shades of green or in this case, the very bright red and green from the displays. Therefore, the thresholding in the code is done by converting all colours to the HSV colour space, which stands for Hue, Saturation and Value of the brightness. Figure 7 ((RGB cube, n.d). and ((“[HSV cone],” n.d.)). shows a cube that illustrates the possible colour combination in RGB and a cone that illustrates how the HSV colour space works.

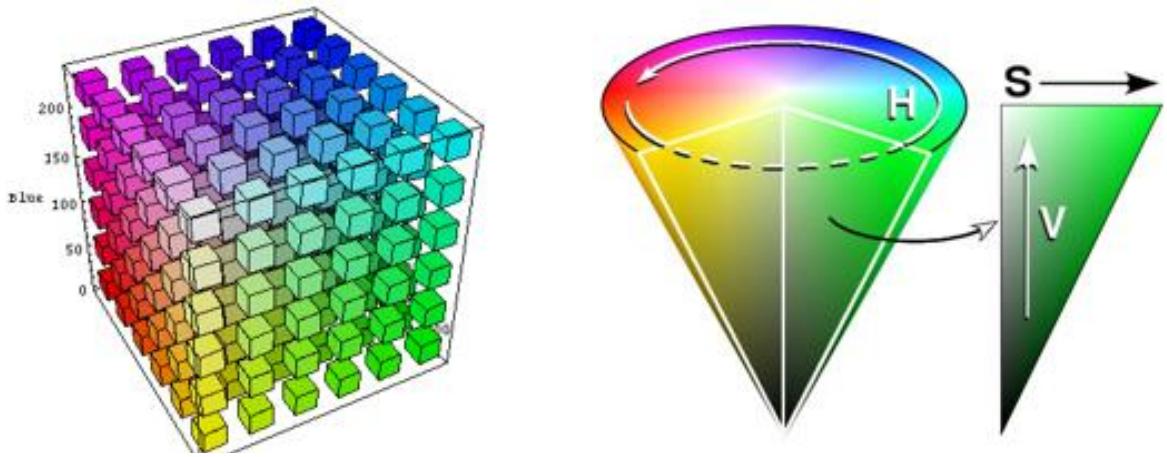


Figure 7: RGB color cube seen from white (255,255,255) [Drawing]. (n.d.) Retrieved from https://engineering.purdue.edu/~abe305/HTMLS/rgbspace.htm and [HSV cone] [Drawing]. (n.d.). Retrieved from https://upload.wikimedia.org/wikipedia/commons/f/f1/HSV_cone.jpg

- The Hue is the colour. In Python, all shades of red are 0 to 60, green is 60 to 120 and blue is 120 to 180.
- The Saturation has a range from 0 to 255 and the lower the Saturation, the more mat the colour is.
- The Value also has a range from 0 to 255 and the lower the Value the darker the colour.

Filtering all red values out of an image with almost completely closed aperture, gives a result like Figure 8.

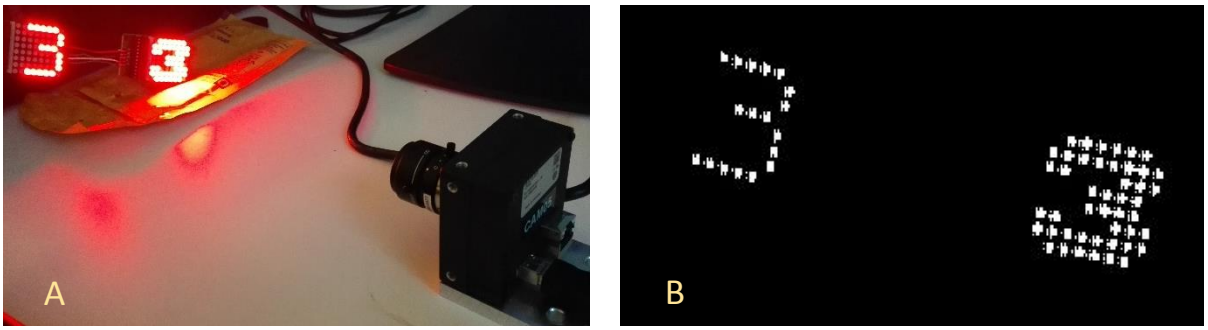


Figure 8: Making an image of two threes of a DOT matrix LED display and thresholding them with HSV and almost closed aperture. (A) Picture of test environment. (B) Thresholded image made by the camera.

The conversion between HSV and RGB can be done with a standard function from the OpenCV-library and can be converted in both ways.

Blurring

Noise in images is always a problem in vision. To filter this out of the image, a technique called blurring is most commonly used. Blurring the image is a technique where all pixels are compared to the ones around it and made more equal to each other. For example, the edge of one of the white dots in Figure 8 touches one or more black pixels. By blurring the image, the black pixels around the white ones become a bit more white, but not the maximum value. The exact opposite goes for the white pixels at the edges. When the right image of Figure 8 is blurred, the result will be like in Figure 9.

Figure 9: Blurred version of Figure 8B

In a good test environment, blurring is not necessary for removing noise. As in Figure 8 and Figure 9 there is no noise, but blurring can still help with pre-processing. Not only is the noise (partially) removed while blurring, but components of numbers that lie close to each other are connected and sharp edges are faded. The CCL needs binary images to process, so the image is thresholded again. Every pixel that is not black (having a value above zero), can be seen as a part of the number and can be made white. The result is shown in Figure 10.

Figure 10: The blurred image is thresholded

Almost all dots are connected, but there are still some pretty large gaps between some parts of the three on the left.

With only blurring and thresholding this appeared to be a problem. It needed to be done several times before the parts where connected (Figure 11), but the middle stripe of the three on the right touched the upper and bottom stripe of the three because these are so close to each other and surrounded by other white pixels. Another method needed to be used to connect the loose parts.

Figure 11: Image blurred and thresholded a second time. The gap between the parts of the left three became slightly smaller, while the three on the right already became a lot thicker.

Dilation and erosion

Dilation makes all white spots in a thresholded image bigger, so when a black image with white numbers is dilated, all numbers become thicker. When looking at Figure 10, it becomes clear that dilation is necessary to connect all parts of the number to each other. Erosion has the exact opposite effect. It makes white object in images smaller. This can be used for making noise in images smaller or even letting them disappear when it is small enough. Erosion and dilation in combination with blurring the image and thresholding the image are powerful tools for image pre-processing to remove noise and make other object easier to process.

The dilation makes all white pixels in the image bigger, regardless of the surrounding pixels. This is a big advantage in opposition to blurring the image. When dilating Figure 11, the result look like in Figure 12.

Figure 12: Dilated version of figure 12

This is exactly what the CCL needs. All components of every image are connected and the image is binary. The only disadvantage is that noisy pixels that may not be filtered out yet are also dilated. These need to be filtered out again. Since a blur will not satisfy because the noise points became a lot

bigger too, erosion is necessary. Every group of white pixels become smaller, but since there is a connection between all points of the numbers, the connection remains. Only the small dots will be removed when the parameters are set correctly for the erosion. A result of erosion might look as in Figure 13.

Figure 13: Erosion after dilation

All noise is removed, the image is binary and all parts of the numbers are connected. This image can be used as input for the CCL. Nevertheless, the process appeared to be slow. It took a couple of seconds to label all pixels, calculate the connected label table and then make every label in the image another value. The reason was that there were a lot of pixels to process since all numbers where relatively thick after the blurring, thresholding and dilating. Even though everything was eroded at the end. The numbers needed to be made thinner, so less pixels needed to be labelled, the connected label table was smaller and the relabelling went faster.

Skeletonize

Skeletonization means that only the basic feature of a binary image is kept See Figure 14 (“[Skeletonization]”, n.d.).

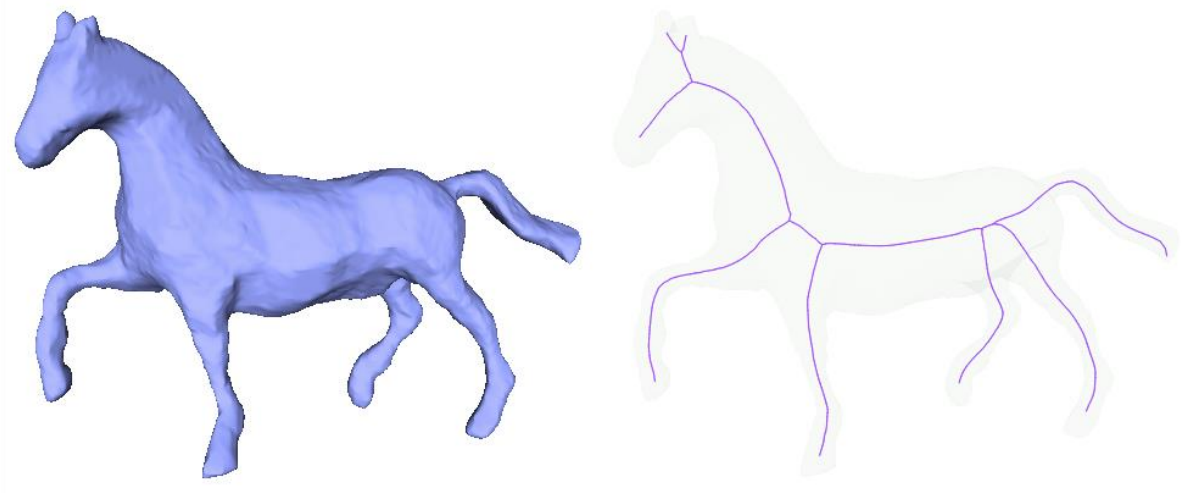


Figure 14: [Skeletonization] [Drawing]. (n.d.) Retrieved from http://doc.cgal.org/latest/Surface_mesh_skeletonization/main_image_suggestion.png

With this function, the basic form of all numbers could be kept, while still processing the numbers, because all components remain connected. They only become smaller. When the numbers from Figure 13 are skeletonized, the figure look like presented in Figure 15.

Figure 15: Numbers in the image after skeltonization

The processing with the CCL now only takes about half a second, what is a big improvement. An important thing to keep in mind is that when there are more numbers to be processed, is will take more time to do so. It needs to process more labels.

The other advantage of skeletonizing all numbers is that is does not matter how thick a number is. They will all become one pixel thick and can be cropped to be prepared for the recognition software. If not all numbers are the same width, the recognition software may have problems with recognising the number. The input always needs to be of the same form so it can learn from its input. The MLP learns the global shape of a number. If it is too stick, it will learn that almost all pixels can be white for a certain number and if it is too small, it will learn only specific pixels can be white if a number is presented.

Figure 16: Numbers from different digital displays without skeletonizationn in the process.

This would not matter if all numbers on the displays would have the same thickness, but this is not the case as can be seen from resulting images in Figure 16. Therefore the skeletonization is a big advantage in the learning process.

Figure 17: Numbers from different digital display after skeletonization and dilation

After skeletonization and then dilation, the numbers are all slightly bigger than the skeletons and the sharpest edges are roughly smoothened. With an end result as in Figure 17, the images can be used for the MLP training to learn what every number looks like and the MLP prediction to predict the value of a number.

Rotating

In Figure 16 and Figure 17**,** the numbers were standing straight in front of the camera when the picture was taken. Carya wants the software to recognise the numbers, even when the camera is rotated with 45 degrees. Figure 15 is a good example of when the camera is rotated.

At first, the rotation was tried to be acquired by looking at the median value of where the numbers were. This appeared to be problematic in some cases (See appendix C1). A more reliable way appeared to be finding the position of every outer rotated number and find the exact middle of it (Figure 18).

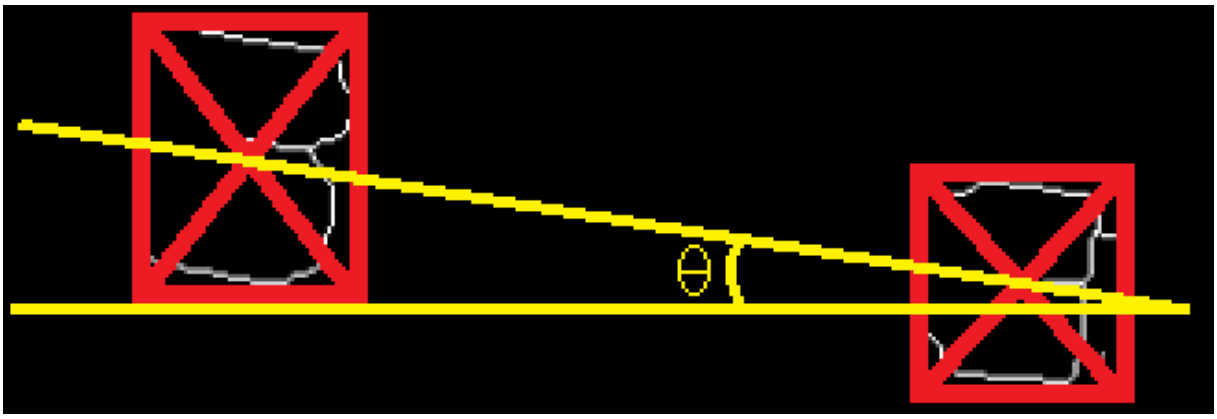


Figure 18: A rectangle around the found numbers and a line drawn between them

A line was drawn between these centre points of the two outer numbers in the image. This line makes a certain angle with the horizontal axis when the numbers are rotated in the image. The image is then rotated by this angle so all numbers are straight in the resulting image (Figure 19).

Figure 19: Rotated skeleton

When the camera is rotated around the axis pointing out of the display, this will always work. But when the camera is rotated around the x-axis and y-axis parallel to the display, the numbers are deformed slightly. This is also visible in Figure 19, where the right number is smaller than the left number. The image could be skewed, but this is an extra process and the software may also be able to learn that numbers are sometimes shown with a little skewness. This of course means there is a need for enough training material, but there can be made plenty to test with.

Besides that, the MLP needs cropped images because too many pixels mean a lot slower processing time. When all images are cropped enough, the most basic contours are kept and small rotation disappear in the resulting images. The only thing that matters that that the width and height of the numbers are kept proportional to each other so the number presents a possible shape and that the height of the numbers in all images is the same in the cut out image of the numbers. Cropped images of the numbers from Figure 19 are shown in Figure 20.

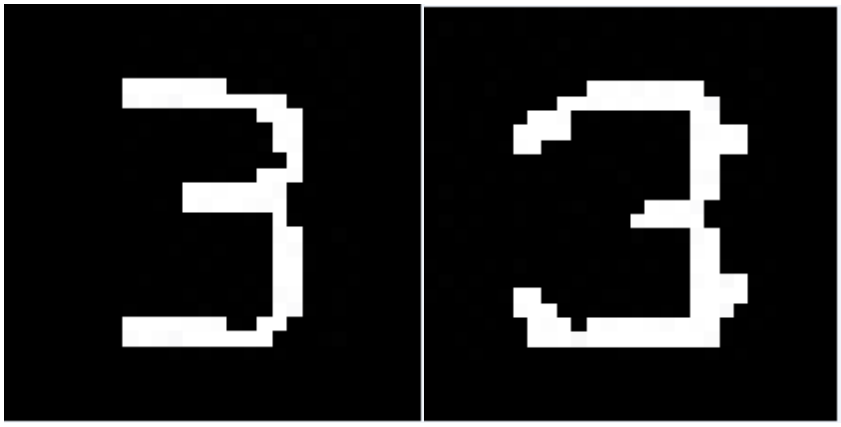


Figure 20: The two resulting images from the pre-processing of the camera image

If the image pre-processing is done, the image from the figure above can be used for training the network or can be put in the network to determine what numbers are presented.

2.3 Conclusion

Learning method

Classification is no option because Carya want a percentage of how good the recognition works. And because it is not known whether all data will be linearly separable, the choice to start with is a learning method that can produce non-linear regression lines which can predict the numbers.

The learning method needs to be eager and not lazy, because eager methods are faster during recognition than lazy methods because they train before recognition starts. Lazy methods do it during the recognition.

SVR and MLP are both great machine learning methods that have proven themselves to be good with OCR in many research papers. There are no hard cons or pros for selecting one of both methods, because they are both able to learn and recognise almost anything in the right circumstances. Carya was already a little familiar with MLPs and is relatively easy to understand because it is based on the human brain. With an eye on the fact that people from Carya need to work with it in the future, the choice fell on the Multi Layer Percepton as learning method.

Pre-processing

The better the pre-processing is, the more basic the machine learning method can be.

For pre-processing the following steps will be taken:

- Thresholding, to filter out the numbers from the image
- Blurring, to remove noise and attach parts of numbers that lie close to each other
- Dilation, to attach the remaining parts of numbers that need to make a connection
- Erosion, to remove noise than could be dilated too and to smoothen the edges of the dilated numbers
- Connected Component Labelling, for detecting what pixels are connected to each other
- Skeletonization, so only the basic contours of the numbers remain
- Rotation, so all numbers stand straight up
- Cutting out all numbers from the images, so the numbers can be used for training the network or as input for the network that needs to be predicted.

After all of these steps are processed, the images can be used for training the MLP or testing the MLP.

3. Detailed design

In the previous chapter the MLP has been chosen as the learning algorithm and all different steps have been defined that will be used for the image pre-processing. This chapter will be about:

- the structure of the code for image retrieval
- the structure of the code for the MLP
- the structure of the code for IPP
- the way all these codes work together to let the number recognition work

This explanation is mainly done with SysML-diagrams. The basic Block Definition Diagram (BDD) of the number recognition software is shown in in Figure 21.

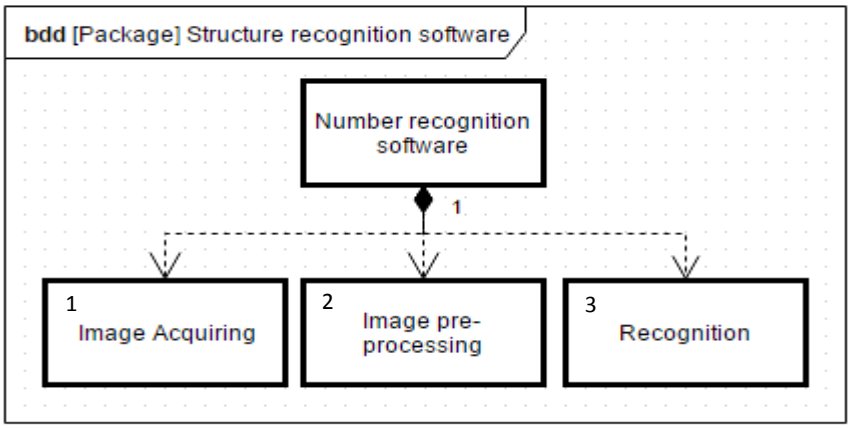


Figure 21: BDD of the number recognition software

3.1 Structure of the code

3.1.1 Image acquiring

For acquiring images, the software needs to receive camera images of numbers on the displays and show these on the screen so the user can to see whether all numbers are in the view of the camera. But this is not enough, because the numbers also need to be visible for the software without too much noise from other object. Therefore the user also sees a thresholded image where, if set correctly, only the numbers in the image are visible.

This means that the Image Acquiring can be divided in the following parts (Figure 22):

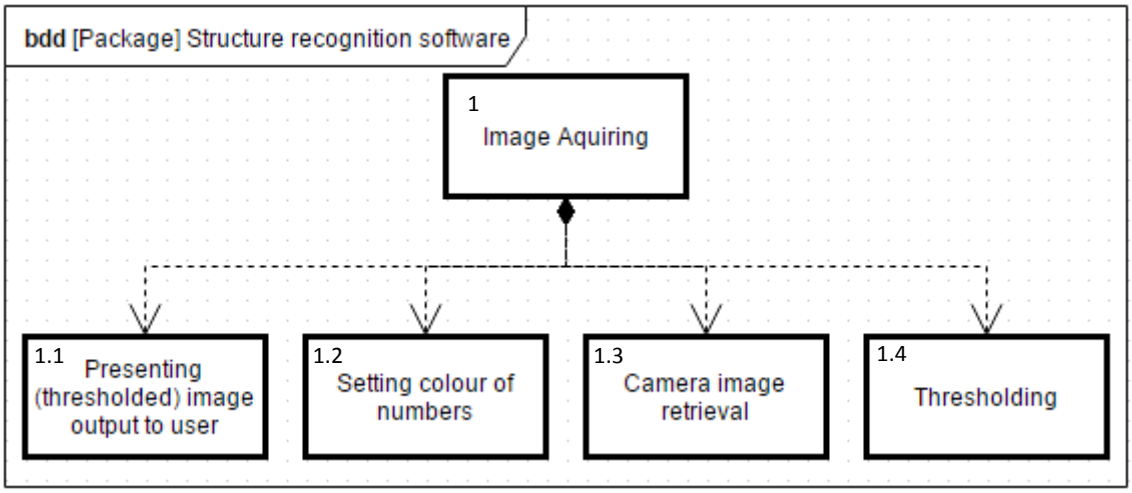


Figure 22: BDD of pre-processing the image

The colour of the numbers need to be set by the user for thresholding. The user has to do this by setting the HSV values. The easiest way is to select the green, red or blue spectrum with the Hue and select all possible values (0-255) with the Saturation and the Value. By looking at the thresholded image from the camera the user can see whether the values are correct or that they need to be adjusted. Further adjustments can also be done with the camera, but more about this in chapter 4.

3.1.2 Image pre-processing

The image pre-processing happens after the image acquiring and before the recognition software is used. The numbers are filtered out of the image and processed in such a way that all numbers with different fonts are presented in the same way to the recognition software. As said before in paragraph 2.2, if the pre-processing is optimized as good as possible, the machine learning method can be kept simple. That is why the pre-processing is a very important part of the number recognition software.

The main structure of the pre-processing software is presented in the BDD in Figure 23. For a complete BDD of the image pre-processing, see Appendix D1: block definition diagram of the Image pre-processing.

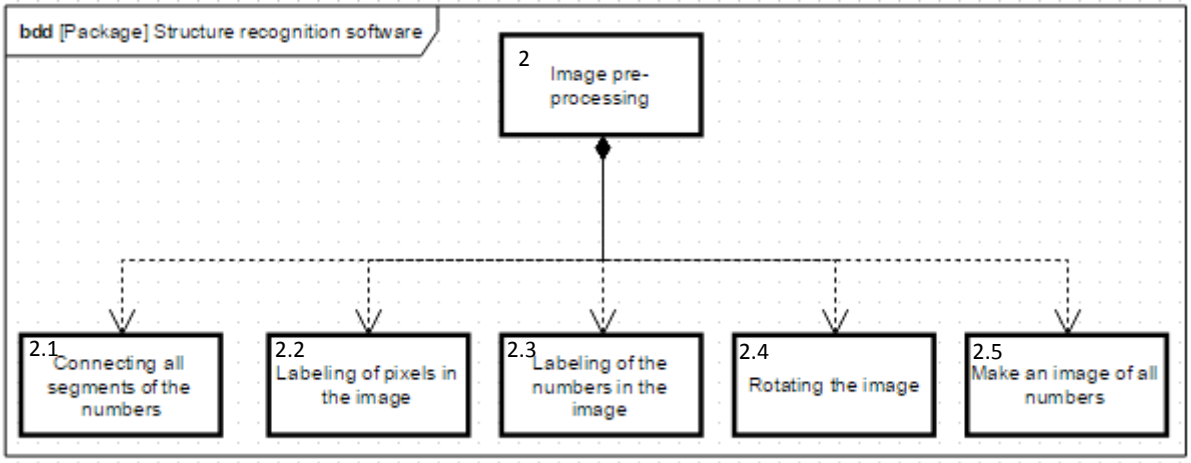


Figure 23: BDD of image pre-processing

All blurring, thresholding, dilation, erosion and skeletonization described in paragraph 2.2, is done to connect all segment of the numbers. The labelling of the pixels can be divided in two parts.

- Labelling all pixels in the image
- Labelling the numbers in the image

The second step means that all connected labels are, thanks to connecting all dots and stripes of every number, labelled the same. It is then possible to filter out everything from the image with a certain label.

3.1.3 Recognition

Now that the images are made and the pre-processing is done, the recognition software can be used to detect what numbers are present in the image. The image recognition can be divided in the following steps presented in Figure 24 (to see the complete diagram of the recognition part of the software, see “Appendix D2: block definition diagram of the Recognition”:

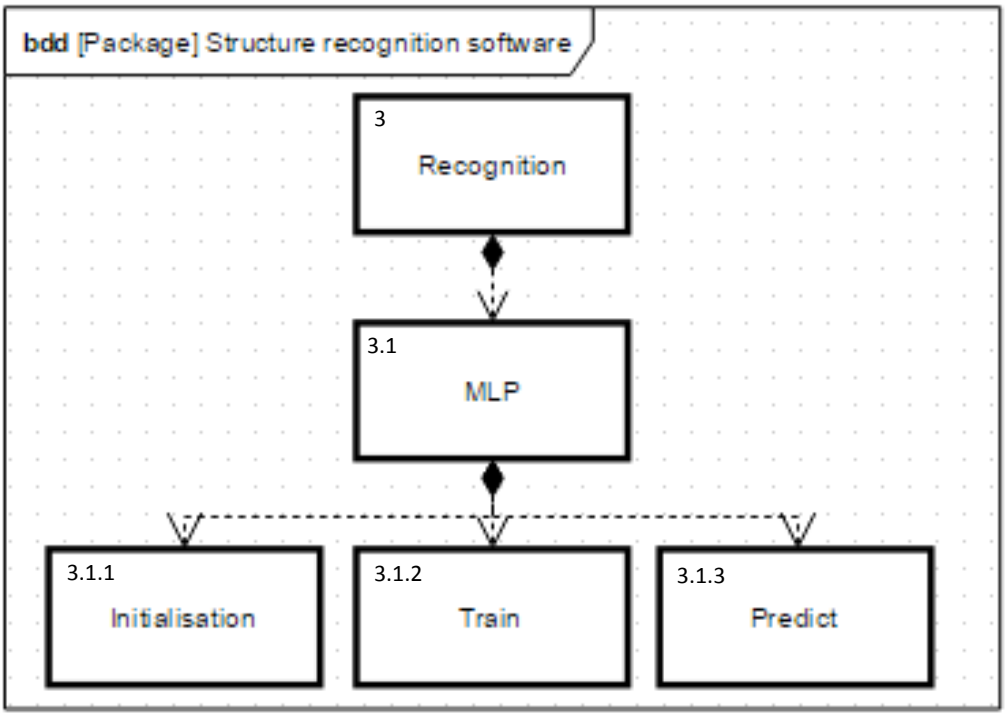


Figure 24: BDD of the number recognition

The recognition software consists of three main parts.

- Initializing the weights and biases at the start. These are determined randomly when the software is being trained
- Training the MLP by adjusting the weights and biases to get a result that is as good as possible.
- Predicting numbers with the MLP. The weights and biases from the training that gave the best result are used to predict the outcome of an image form a camera.

3.2 Process of the code

The above codes with the image acquiring, image pre-processing and the learning method combined can make the complete number recognition software. When all blocks (for more info in deeper layers in the BDD, see Appendix D) are used for the structure of the code, the classes and functions inside it will look like in the class diagram in Figure 25.

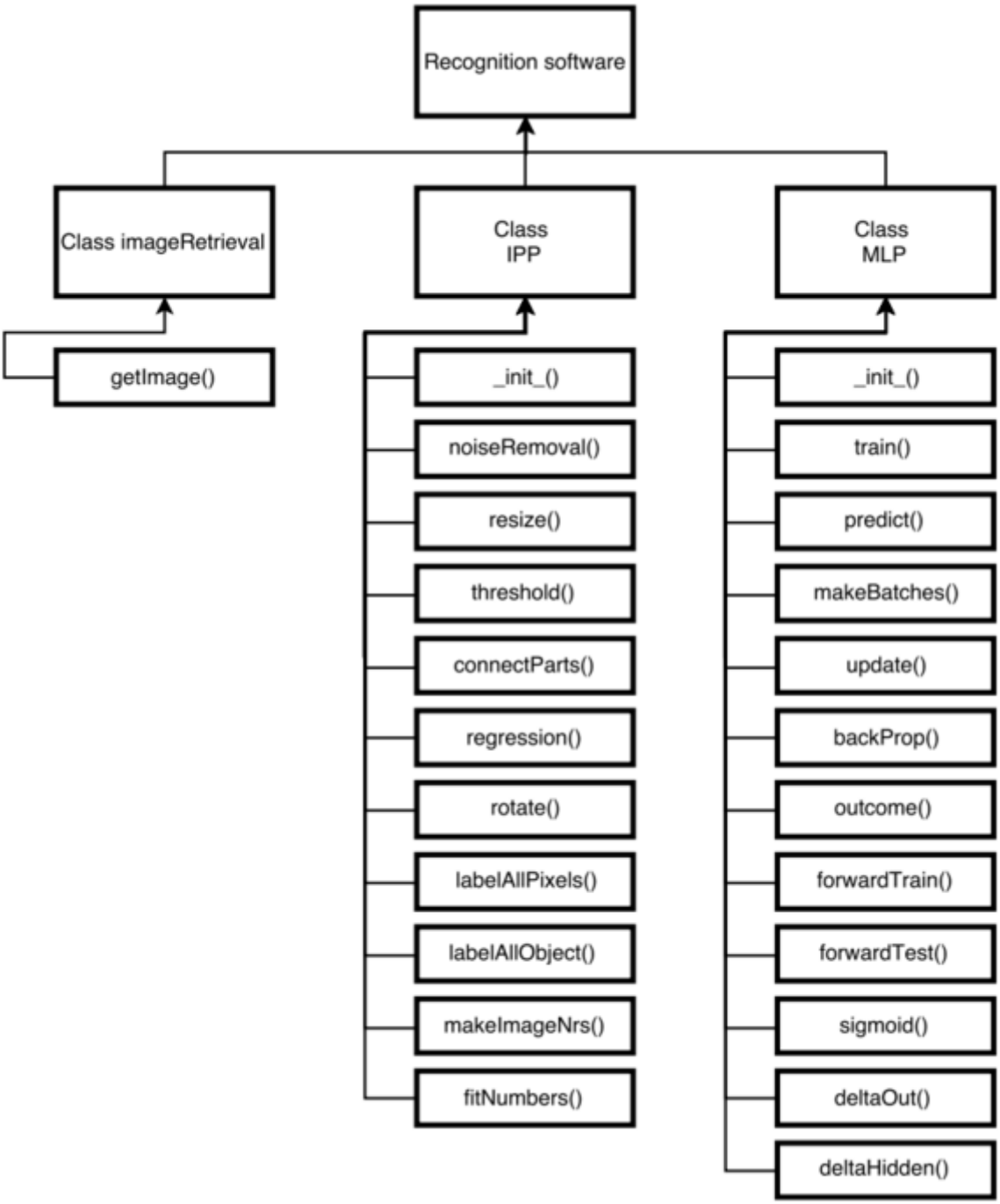


Figure 25: A class diagram of the number recognition code. This is made with the block definition diagrams.

An activity diagram is shown in Figure 26. For a sharper image, see “Appendix D3: Activity diagram of the number recognition software”. It illustrates how all parts of the recognition software work together to get the software to learn numbers or recognise numbers.

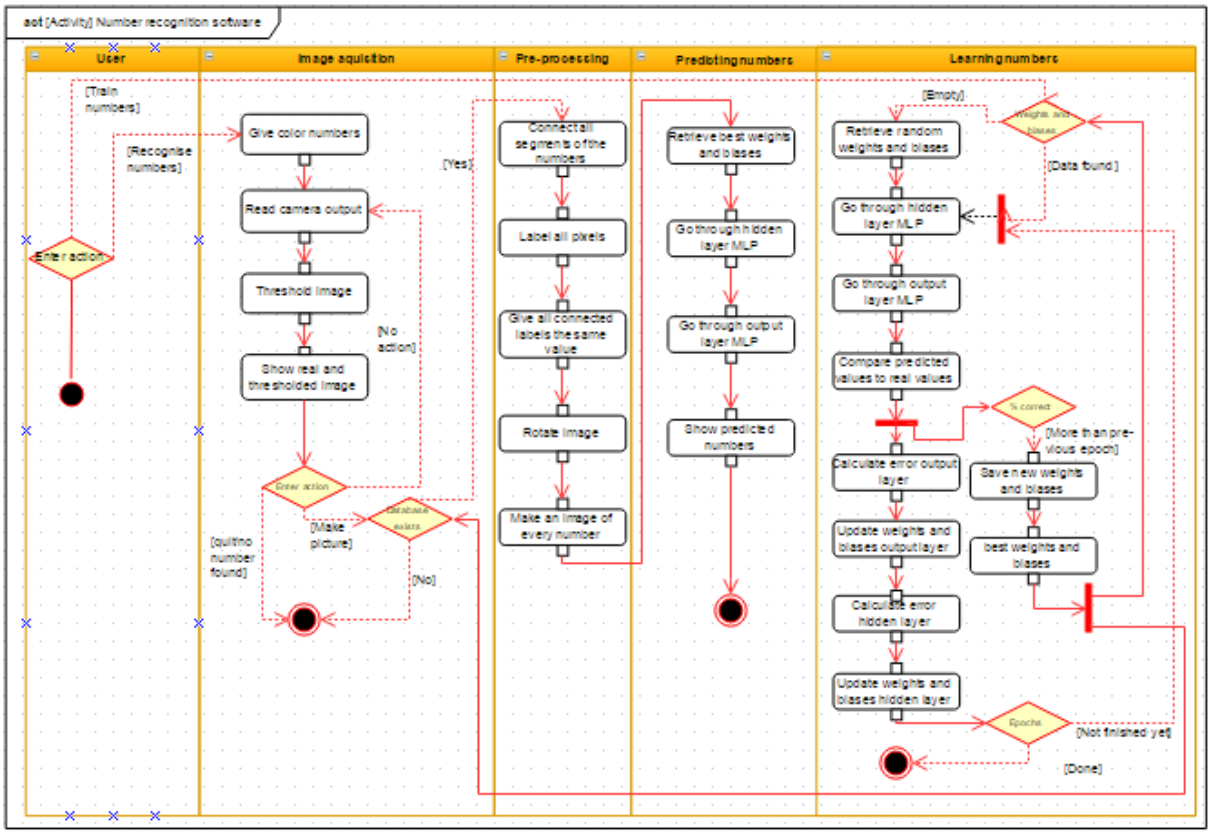


Figure 26: Activity diagram of training the learning method

Shortly explained, the user can choose whether he wants to train the software or to use it to predict a value of a number. If the user wants to train the network, it directly goes to the training parts and retrieves trains the biases and weights. If the user wants to predict numbers with the network, then the software first retrieves an image from the camera and then the software goes to the number recognition part, where it retrieves the best weights and biases.

The first block in the image acquisition part says the user has to give the colour of the numbers. This needs to be done so thresholding is possible. After the threshold is given by the user (in HSV), the software returns the real camera image and a thresholded image continuously to the user on the computer screen, with which the user can see whether the threshold values are correct.

If everything the threshold good, the user can press the “p” to take a picture and continue to the pre-processing of the image. If “q” is pressed, the program stops and the user can set the threshold values again.

Summary

The number recognition software can be divided in three main parts:

- Image acquisition
- Image per-processing
- Learning and predicting numbers

The user has to predefine whether he want to train or test the network. If he wants to train, the network needs no further input from the user and goes on with the training directly. If the user wants to recognise numbers from a display, the user has to set the threshold values that can be checked with the live feed from the camera with the thresholded live feed that is presented on a computer screen. If the threshold needs to be adjusted, the user can quit by pressing “q” and adjust them. If he is satisfied and want to start the prediction of numbers, the user can press “p”.

4. Realisation

If the user wants to predict numbers from displays, he needs to set the threshold values in the code. This is not the only thing that has influence on the resulting image. A very important parts is the way the camera and the display are positioned, the lighting in the room, the aperture of the camera, and so on.

In the realisation phase, the code is programmed and the test environment is made. The test environment can be divided in the following parts:

- Camera: which camera is used?
- Displays: which displays are used?
- User set-up: what does the user need to set-up before the software can be started?

At the end of the chapter a budget analysis is given whether the budget analysis of the Plan of Approach was estimated correctly.

4.1 Test environment

4.1.1 Camera

While being in search of a camera, the Hague University of Applied sciences in Delft proposed the following camera and lens:

- Camera: the Basler A312fc.
- Lens: a Navitar NMV-5WA.

The lens has a focal length of 4.5 mm and has an manually adjustable aperture to set the amount of light that passes through onto the light sensor in the Basler-camera. The focus is also manually adjustable to sharpen or blur the image. Figure 27 shows an image of the Basler camera with the Navitar lens. For the specifications of the camera and the lens, see “Appendix E Specifications of the Basler A312fc camera” and ”Appendix F Specifications of the Navitar NMV-5WA camera lens”

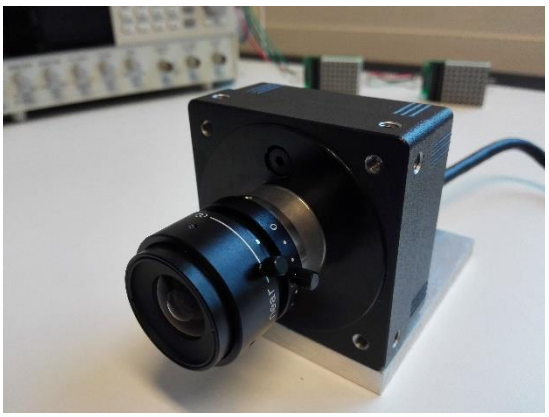


Figure 27: Basler-camera used for the project

The aperture can be set between f/1.4 and f/16, with f/1.4 almost completely open and f/16 almost completely closed (see Figure 28 (Hill, 2010)).

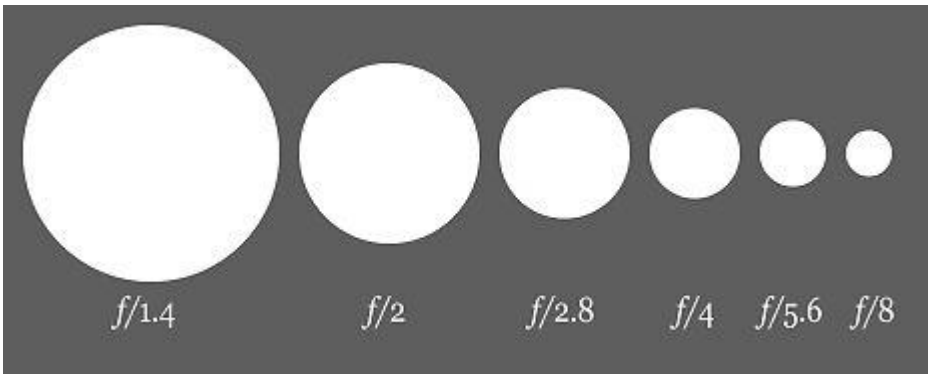


Figure 28:Peter Hill. (2010). [Apertures with different sizes][Drawing]. Retrieved from http://www.redbubble.com/people/peterh111/journal/5725038-the-easy-guide-to-understanding-aperture-f-stop

The Basler camera has a relatively high frame rate. It makes 53 frames per second (fps), where most good, expensive webcams mostly only have 30 fps. Some standard rules for cameras are that:

- the more the aperture is closed, the less light can go through and reach the sensor in the camera that captures the light
- the higher the frame rate, the less light can be let through the lens per frame per second and the darker the image will be with the same amount of light.

When the aperture is closed almost completely, only the light from bright light sources is let through. Like light from the LED displays. An almost closed aperture in combination with the high frame rate makes sure that most of the background is filtered out in the resulting image, but that the numbers shown on the LED displays are still visible on the image. For the 7 segment display however, this might not always be the case since these LEDs shine less bright than the dot matrix displays and the lines of the numbers are thinner than the dots of the other display. It may be needed to open the aperture a little bit during the testing, but this also depends on other factors, like the light in the test environment.

Fish eye

The lens of the camera is a 4.5 mm lens and therefore a wide angle lens. An advantage is that the lens can make very sharp images at short distances and has a very wide viewing angle.

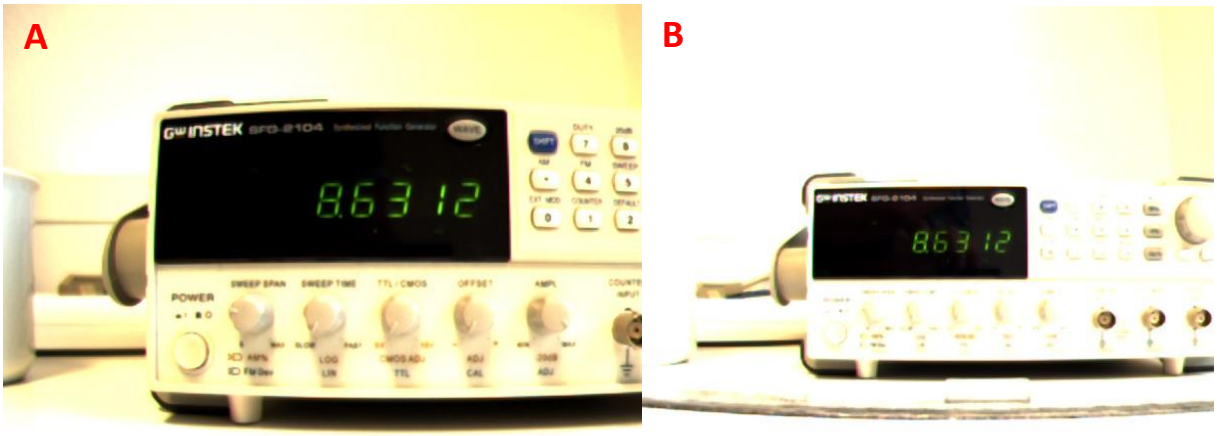


Figure 29: A picture taken from close by the display (A) and a picture taken a bit further from the display (B). When looking at the black part of the display, the fish eye distortion is clearly bigger in figure A than in figure B.

A big disadvantage however, is that it suffers from fisheye distortion/barrel distortion. This means that all objects are made more round in the image. The closer the object to the lens, the worse this gets (see Figure 29). The closer an object is at the sides of the view range of the lens, the worse the distortion gets. In the exact centre of the picture, there is no distortion.

All cut out numbers are cropped so much that in most cases this is not visible any more, but the user has to keep in mind that the closer the camera gets to the display, the more the fisheye distortion will be.

The fisheye could also be removed in the pre-processing, since the specifications of the camera lens are known and the distance between the lens and the object are known. However, after cropping all images of the numbers in the pre-processing it was barely visible that they were distorted in comparison to the ones taken with the camera standing on a further distance from the display. (see chapter 5). Since this straightening of the image would be an extra step in the pre-processing, and extra steps take more time, this was left out of the pre-processing.

Because of the relatively high frame rate, the manual changeable focus and manually changeable aperture, the Basler camera in combination with the Navitar lens are suitable for this project. The only disadvantage is the fish eye effect, but due to the cropping in the pre-processing, this hardly has any effect on the images that are made of the numbers.

4.1.2 Displays

The displays where the numbers needed to be recognised from are:

- 7 segment display
- 8x8 dot matrix LED display
- 5x7 dot matrix LED display

7 segment display

The 7 segment display was acquired from an function generator of Carya that was also used in the project that formed the basis for this project. A GW Instek SFG-2104. Figure 30 shows an image of the numbers presented on the function generator.

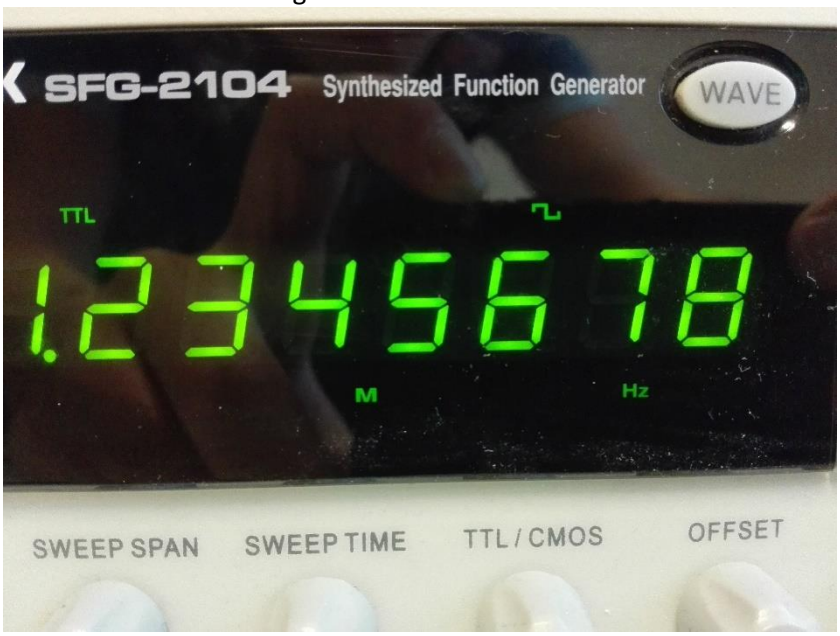


Figure 30: Numbers 1 to 8 on the function generator that has been used for the test environment

An advantage was that all numbers were presented a little bit sheared to the right side. The numbers could easily be projected with the same structure on the dot matrix displays, but standing straight up. This means that there is more variety in the learning images and the learning software will learn a bigger variation of numbers.

8x8 and 5x7 DOT matrix displays

Machines or measuring equipment with 8x8 and 5x7 dot matrix LED displays where not available at Carya, so these needed to be ordered. An Arduino has been used for presenting the numbers on the dot matrix displays, so the numbers could be defined manually by setting every LED separately. The reason an Arduino has been chosen, is that there was already one present for testing. Any other micro controller board equal to the Arduino board could have been used.

Two 8x8 dot matrices had been ordered for the test environment. The specifications of the display are presented in “Appendix H1 specifications of the 8x8 dot matrix LED display”. These displays had certain advantages:

- The display could also be used for presenting numbers from 5x7 dot matrices. (Figure 31, picture C).
- The numbers were red, so the recognition software was not only being tested on the green numbers from the 7 segment display of the function generator.

The displays were ordered together with a two micro controllers (MAX7219). Not only was this almost as expensive as buying only the display, this was also a way to reduce the amount of needed ports on the Arduino. Instead of a port on the Arduino for every row, column, a voltage supply and a ground, there were only five ports used on the Arduino:

- Load
- Voltage supply
- Ground
- Clock
- Input signal

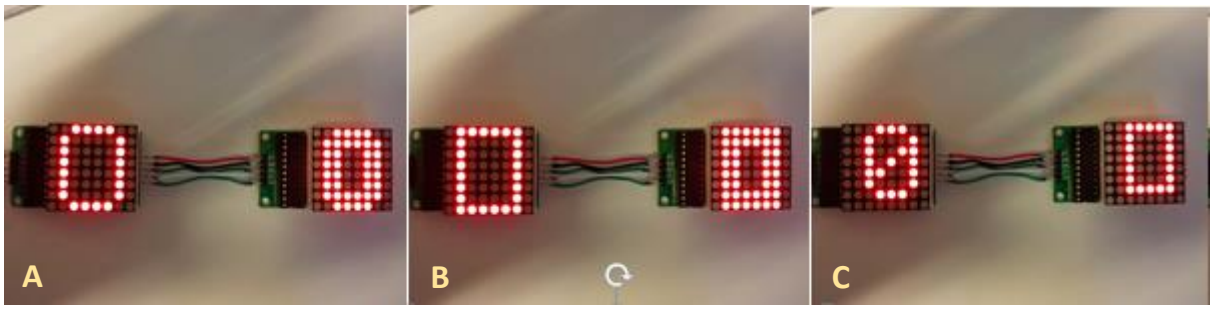


Figure 31: numbers presented that are common on 8x8 (A and B) and 5x7 DOT matrices (C)

For programming the Arduino, the library LedControl was used. This library can be retrieved from Github with the source https://github.com/wayoda/LedControl

4.1.3 Setting up the test environment

It was already said that the user needs to set the threshold values so the numbers from the displays could be filtered by colour. This is very important to do the filtering, but not the only thing There are also a lot of other factors from the test environment that play an important role. In Figure 32, a use case diagram is shown where these steps are presented.

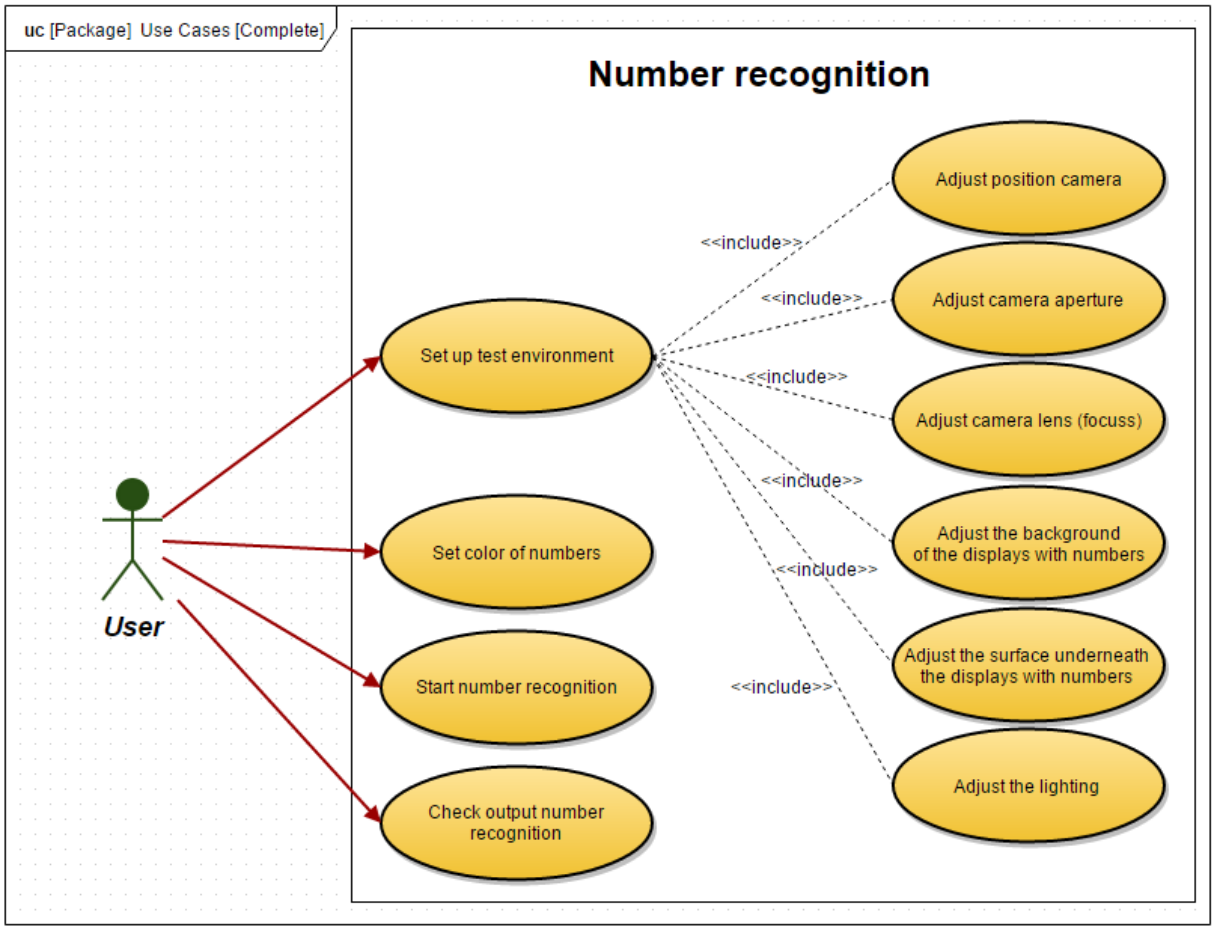


Figure 32: Use case diagram of the number recognition software

Setting up the test environment is equally as important, if not more important than the exact right threshold values. If setting up the test environment is done correctly, the amount of noise in the image is reduced and the displays will be presented better to the camera.

Aperture

Carya wanted the number recognition software to work with as few light adjustments as possible.

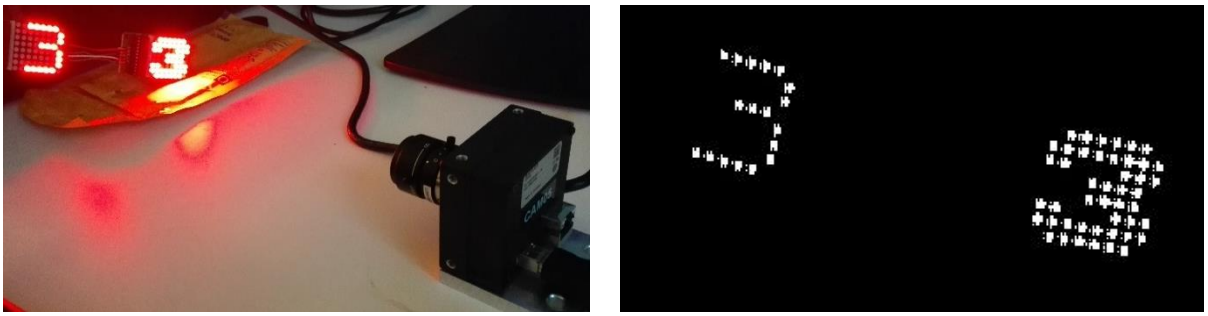


Figure 33: Making an image of two threes of a DOT matrix LED display and thresholding them with HSV and almost closed aperture. (A) Picture of test environment. (B) Thresholded image made by the camera.

When the aperture is closed almost completely (f/16), only very bright light can be seen on the image from the camera. Since the displays are LED displays, they remain visible on the image when showing a number, while the background light is almost complete filtered out (Figure 33).

In case the environment is still very bright, it might be possible that there is still some noise from the environment coming through the aperture and can be spotted on the thresholded image.

The focus

The focus of the camera should be as much optimized as possible, but when the aperture is almost closed, the difference between a sharp and blurry image is significantly low, because there is very few light that comes through. When the aperture is open, it becomes important that the image is as sharp as possible. Beyond a distance of 30 cm from the display, this can become a problem because the lens is a wide angle lens that is specialized in making sharp images from objects close by the lens.

Surface underneath the displays

As can be seen from the image above, there is also a lot of light on the surface where the numbers are shining above. If a matt surface is used, like paper, the reflecting light is filtered because the light is not bright enough. It is important that the surface is not mirroring the light, because the light intensity then remains the same as when it comes directly from the display.

Background of the displays

The background behind the display should preferably be black and no shiny surface. When the background is a light colour, for example white, the background produces noise because it is not filtered out by the aperture in rooms with lighting. If the background must be a bright colour, like white, than it is best to make the room dark.

Distance between the camera and the display

Carya wanted the software to work for distances with a camera distance of 15 to 30 cm and with a maximum angle of 45 degrees with respect to the display in x, y and z direction. For these distances the software has been tested.

Lighting

Lighting is one of the most important parts with vision, because too much light can for example cause noise in the image or too much over exposure, which makes the image almost completely white. With LED displays, the LEDs are bright light sources themselves and it is preferable that there is no light from the environment. Carya however wanted the software to work with as few light adjustments as possible. In a normal workspace with TL lights at the ceiling, the software should work for the LED displays that were used in the tests.

Conclusion

The user needs to make sure the aperture, focus, lighting, background of the display, surface underneath the display and the distance between the display and the camera is set correctly to make sure the number recognition software works properly.

The software is persistent against TL light in offices, but the background always needs to be dark and not shiny. The same goes for the surface underneath the displays, so that the surface can not reflect any light from the displays into the camera lens.

4.2 Budget analysis

The budget plan that was made at the beginning of the project, is shown in Figure 34, together with the budget analysis.

A

What	Price
Firewire	€ 39,99
Cable	€ 14,99
7 segment display	€ 4,00
alphanumeric display	€ 4,00
5x7 DOT matrix display	€ 4,00
8x8 DOT matrix display	€ 4,00
Arduino	€ 0,00
Camera	€ 0,00
Computer	€ 0,00
Computer screen	€ 0,00
Wires for connecting displays	€ 10,00
Payment intern (total)	€ 935,00
Total	€ 1.015,98

budget analysis

B

What	Price
Firewire	€ 39,99
Cable	€ 14,99
7 segment display	€ 0,00
(2x) 8x8 DOT matrix display	€ 13,90
Arduino	€ 0,00
Camera	€ 0,00
Computer	€ 0,00
Computer screen	€ 0,00
Wires for connecting displays	€ 0,10
Payment intern (total)	€ 935,00
Total	€ 1.003,98

Figure 34: The budget analysis and the true costs

The dot matrix displays were a little more expensive than estimated, because they were ordered together with the MAX7219 and other components like capacitors, resistors and diodes and there were some additional shipment costs. The 7 segment display however did not need to be ordered because there was an function generator present with a 7 segment display that could be set with any number that was needed and the alphanumeric display was also not ordered. It was not parts of the project and was meant to be done when there was time left.

The cables were less expensive. This was because not all rows and columns of the displays needed to be attached to the Arduino, but only 5 of them and some cables to connect the two matrix displays. Approximately only around 10 cents of cable has been used for the project. The total cost of the project are 12 euro cheaper than estimated.

5. Testing

When the software and the test environment where ready, the test could be done to see how rigid the number recognition software was. Not all tests could be done.

According to the requirements that Carya set for this project, a table could be made with all points that the software has to meet. See Table 7 for the points where the software will be tested for.

Table 7: Table to confirm the requirements of Carya

Nr	What
1	Has machine learning been used in the recognition software?
2	Can the software recognize the numbers zero to nine from 5x7 dot matrix LED displays, 8x8 dot matrix LED displays and 7 segment LED displays?
3	Can the software recognize the numbers zero to nine from all the above displays when the camera is rotated with 45 degrees?
4	Can the software recognize the numbers zero to nine from all the above displays when the camera is positioned between 15-30 cm?
5	Can the software recognize numbers without light adjustments?
6	Can the test environment be set up within 5 minutes?
7	Does the software show the original image, together with predicted value of the number and a percentage of how certain it is that the number has that value?
8	Can the recognition software check numbers every 5 seconds?

Test results

1. *Has machine learning been used in the recognition software?*

Yes. The Multi Layer Perceptron has been used for the recognition of numbers. The learning method has been kept as basic as possible. The pre-processing has to process the images in such a way that these can be implemented directly in the MLP. The better the input in the MLP, the more basic the MLP can be.

2. *Can the software recognise the numbers zero to nine from 5x7 dot matrix LED displays, 8x8 dot matrix LED displays and 7 segment LED displays*

Yes. A database consisting of a total of 4591 images of all numbers and displays was made. These images were made by manually placing the camera with distances from 15 to 30 cm from the displays and with angles with a maximum of 45 degrees. After training the MLP multiple times with 1/8 of this database (random), the best result came after 88 training round (epochs). The software was able to recognise 98.1 of all numbers in the test set. The output of the code was:

> "Epoch 88 : 563 out of 574 correct (98.0836236934 %)"

In "Appendix I Best test results of the number recognition software", it can be seen that the software predicted the 1, 2, 4 and 5 without problems.

Whereas the recognition software was able to predict one, two, four and five without mistakes, the software showed that the six, eight and nine were the hardest numbers to predict. The software sometimes predicts the big block numbers wrong. With this, I mean the numbers like in Figure 35.

Figure 35: the numbers eight, eight, six and nine after pre-processing. They show small differences between each other

The differences between the block numbers are very subtle. Mostly only a small stripe. For example, a nine is sometimes seen as an eight and vice versa. Although this difference is small, it must still be able to be recognised. To improve the learning algorithm, it could be tried to make the database bigger. The prognosis is that this will improve the test results, because the perceptron will get more pictures to use for the training. It does however specifically need the block numbers because these appear to be difficult for the learning software.

3. *Can the software recognise the numbers zero to nine from all the above displays when the camera is rotated with 45 degrees?*

Yes. The images in the database are taken with random angles and since the recognition software is only sometimes having some trouble with certain numbers, it is safe to say that the angle of the camera was no problem in the number recognition.

4. *Can the software recognise the numbers zero to nine from all the above displays when the camera is positioned between 15-30 cm?*

Yes, but after 25 cm the image pre-processing sometimes is not able to filter the number from the image, because the image is not sharp enough after this distance. Especially when it is angled. This is a characteristic of the lens and this could be solved by using a bigger lens. Keep in mind that a bigger lens also means a smaller view angle.

5. *Can the software recognise numbers without light adjustments?*

This depends on the situation. The software is robust against TL light when the background of the image and the underlying ground is non-reflecting. As long as this is the case, then no, light adjustments are not necessary.

6. *Can the test environment be set up within 5 minutes?*

Yes. The training of the weights and biases of the MLP however might take a little bit longer. This depends on the speed of the computer, the amount of training images, the amount of test images, the parameters of the MLP and so on. On a Toshiba Satellite C50, it is possible to do the training in 5 minutes. During the training the test environment can be set up.

7. *Does the software show the original image, together with the predicted value of the number and a percentage of how certain it is that the number has that value?*

Yes. It also shows for every other number what the certainty is. The number with the highest certainty is the predicted value.

8. *Can the recognition software check numbers every five seconds*

Because of the fact that there was limited time for the project and this was one of the least concerns for Carya, this has not been tested yet. A prognosis would be yes, but this should be measured to be able to say it with certainty.

Assumptions that have been made

At the beginning of the project, two assumptions have been made where the whole project was built upon. These assumptions where:

- The learning method can be kept basic, if the image pre-processing is good

The database consists of the numbers zero to ten with six different fonts per number. The fact that 98.1% of these images is estimated correctly with a relatively small database, means that using a basis learning algorithm with good input images can result in almost an optimal result and that the assumption was correct.

- Linear learning methods may not be able to separate all data

The fact that there is still two percent of the test images that is not recognised, means that the software is not able to make a non-linear function (with the current training database) to produce a prediction line for the numbers that will work for 100%. A MLP can easily be converted to a single layer perceptron by removing the hidden layer, thereby making it a linear learning method instead of a non-linear learning. The best result is:

"Epoch 117 : 526 out of 574 correct (91.637630662 %)"

This shows that only a little less than a tenth of the test database could not be estimated be a linear separator after the single perceptron is trained. The main problem appears to be recognising the five, The software mostly thinks it is a six or a nine, because these are basically the same with the exception of one stripe. (see "Appendix J Best test results of the single (linear) perceptron".

The data from the training set is not linearly separable, because it is not able to find a linear function that can predicted all numbers. Although it comes really close.

6. Deliverables

The deliverables of the project where:

- The recognition software
- A validation report of the recognition software

All of the deliverables are handed over to Carya Automatisering and The Hague University of Applied Sciences on the 17th of December 2015. The validation report can be found in this report in chapter 5.

As appendices of this report, additional information about certain parts of the project is presented and all datasheets of equipment used in this project, like the displays and the camera.

7. Conclusion and recommendations

In this chapter, the conclusion of the project is given and a recommendation in case Carya wants to use the software and test environment or future project.

Conclusion

For the project, Carya wanted to have vision software that was able to recognise numbers from 7 segment LED displays, 7x5 dot matrix LED displays and 8x8 dot matrix LED displays. The matrix displays have red LEDs and the 7 segment display has green LEDs. The project was divided in a practical assignment and a research assignment.

Research question

The research question was: "*What is the best method for recognising numbers for this project?*". Carya wanted the software to be able to show how big the chance was that a predicted value is the correct value. Because of this, the learning method needed to be a regression method and not a classification method, which only shows the class a number belongs to.

The assumption that was made during the beginning of the project, was that the numbers were not linearly predictable by a line. The research therefore showed that the best method was a non-linear supervised learning algorithm. The Multi Layer Perceptron (MLP) was chosen to be used as the learning algorithm.

The research also showed that the worse the test input was, the more complex the learning algorithm needed to be to recognise all numbers. A basic learning algorithm can be used without any other tricks for better recognition, as long the input images are good enough. Therefore the course for the practical assignment was to use a basic version of the MLP and to focus mainly on the image pre-processing.

The results that the recognition software showed, made clear that this assumption was correct. 98.1% of the test images could be recognised when the algorithm learning the numbers from the training input. The bar that was set was a minimum of 95%, so the learning method performed better than expected.

Assumptions that have been made during the project

At the beginning of the project, there were some things that could not be determined with certainty. To continue the project, some assumptions were made where the project was built upon. These assumptions where:

- The learning method can be kept basic, if the image pre-processing is good.

The database consists of the numbers zero to ten with six different fonts per number. The fact that 98.1% of these images is estimated correctly with a relatively small database, means that using a basis learning algorithm with good input images can result in almost an optimal result and that the assumption was correct.

- Linear learning methods may not be able to separate all data

The fact that there is still two percent of the test images that is not recognised, means that the software is not able to make a non-linear function (with the current training database) to produce a prediction line for the numbers that will work for 100%. The assumption that not all data can be linearly separated, it true.

Assignment

The project assignment was "*Write recognition software that is able to recognise numbers with different fonts that are presented on digital displays*.". All demands of Carya are met, except for the one about the time that could not be tested yet. This however was not very important to Carya, long as the software worked, since the project was a proof of concept.

The number recognition software was able to recognise numbers from the test set with a certainty of 98.1% and meet all important requirements, and can therefore be called successful. Here must be noted that the digital displays are only the earlier defined displays in the project.

Recommendations

Camera

Carya can better use another lens for future vision projects that have the same requirements as this project. The Basler-camera its manually changeable aperture and focus were very handy and could be changed within a second. The lens however has the problem that is creates barrel distortion in the image because it is a wide angle lens. If Carya wants to use a camera for the same test environment as with this project, I would recommend to use a bigger lens of 8 mm or 12 mm to prevent this kind of distortion.

Database with numbers

The database with images of numbers probably needs to be bigger than the 4500 images that are now in the database. It is enough to predict 98.1% of the test images correct, but this could probably be better. Especially since the mistakes are clearly made because of numbers that look like each other. With more training data, this can possibly be solved. Note that all parameters for the Multi Layer Perceptron need to be changed again to get the best result.

Programming language

Carya want to implement the code in LabVIEW, what I think will be a lot faster than Python. Python does the trick, but it is not very fast in comparison to languages like C and C++, because it is an interpreter and not a true programming language. I do not know how fast LabVIEW codes are however.

Test environment

Carya wants to adjust as few as possible to the light in the environment while testing. It is possible to filter out the numbers with the software if there is no over exposure, but I would advise to take the pictures in a dark space if LED displays are being checked. Light from any other source means that a lot of settings need to be changed in the test environment and that there is a bigger chance that there will be noise in the image. On top of that, the pre-processing will take less time if some filtering steps can be skipped.

References

Carya Automatisering (n.d.). [Logo Carya] [Image]. Retrieved from https://www.carya.nl/nl/

[connected label table]. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Connected-component_labeling

Cyrille Rossant. (2014). *[A red classification line in the left image and a red regression line in the right image*] [Drawing]. Retrieved from http://ipython-books.github.io/featured-04/

Dedgaonkar, S. G., Chandavale, A. A., & Sapkal, A. M. (2012). *Survey of Methods for Character Recognition* (ISSN: 2277-3754). Retrieved from http://www.dsi.unifi.it/NNLDAR/Papers/01-NNLDAR05-Liu.pdf

De Haagse Hogeschool (n.d.). *Haagse Hogeschool* [Image]. Retrieved from

https://bldng360.nl/bedrijven/Haagse%20Hogeschool

Dhull003 (2010). [square 8 connectivity] [Drawing]. Retrieved from https://en.wikipedia.org/wiki/Connected-component_labeling#/media/File:Square_8_connectivity.png

Dhull003. (2010). Result of connected region labeling using two-pass raster scan [Drawing]. Retrieved from https://en.wikipedia.org/wiki/Connected-component_labeling#/media/File:Screenshot-Figure_1.png

Dhull003. (2010). Example of an array where connected region labeling is to be carried out. 1 represents the region pixel, and 0 represents the background pixel. Retrieved from https://en.wikipedia.org/wiki/Connected-component_labeling#/media/File:Screenshot-Pixel_Region_(Figure_1).png

Driessen K. (2015, september 2). *Structuur plan van aanpak (PvA) voor het hbo*. Retrieved from

https://www.scribbr.nl/scriptie-structuur/structuur-plan-van-aanpak-pva-voor-het-hbo/

DVS (n.d.). [Numbers recognised by software and displayed] [Photograph]. Retrieved from

http://www.dvs-vision.de/sites/default/files/images/Bilder/Zeichen%202.png

Liu, C. (n.d.). Classification and Learning for Character Recognition: Comparison of Methods and Remaining Problems. Retrieved from http://www.dsi.unifi.it/NNLDAR/Papers/01-NNLDAR05-Liu.pdf

Mustapha, A. (2013, August 7). What are the differences between machine learning, pattern recognition and data mining? [Forum post]. Retrieved from https://www.researchgate.net/post/What_are_the_differences_between_machine_learning_pattern_recognition_and_data_mining

Nielsen, M. (2015, June). Neural Networks And Deep Learning. Retrieved December 17, 2015, from http://neuralnetworksanddeeplearning.com/index.html

Peter Hill. (2010). [Apertures with different sizes][Drawing]. Retrieved from http://www.redbubble.com/people/peterh111/journal/5725038-the-easy-guide-to-understanding-aperture-f-stop

PMWIKI (n.d.). *Plan van Aanpak*. Retrieved from www.ipma.nl/wiki/kennis/plan-van-aanpak

projectsmart (n.d.). *The stages of a Project*. Retrieved from http://www.projectsmart.com/project-management/the-stages-of-a-project.php

RGB color cube seen from white (255,255,255) [Drawing]. (n.d.) Retrieved from https://engineering.purdue.edu/~abe305/HTMLS/rgbspace.htm and [HSV cone] [Drawing]. (n.d.). Retrieved from _https://upload.wikimedia.org/wikipedia/commons/f/f1/HSV_cone.jpg

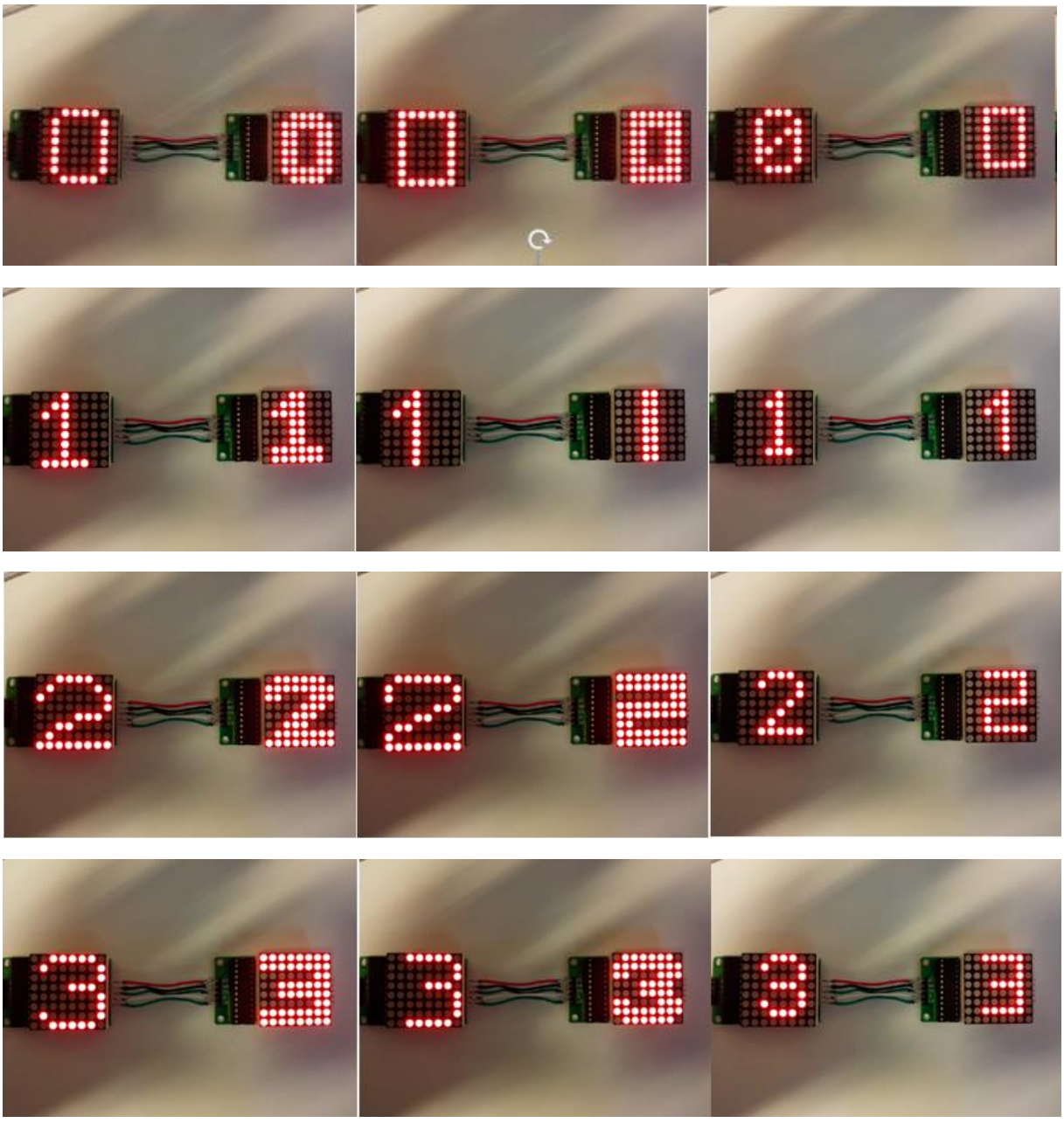
Semi-supervised learning. (2015, June 18). Retrieved December 17, 2015, from https://en.wikipedia.org/wiki/Semi-supervised_learning

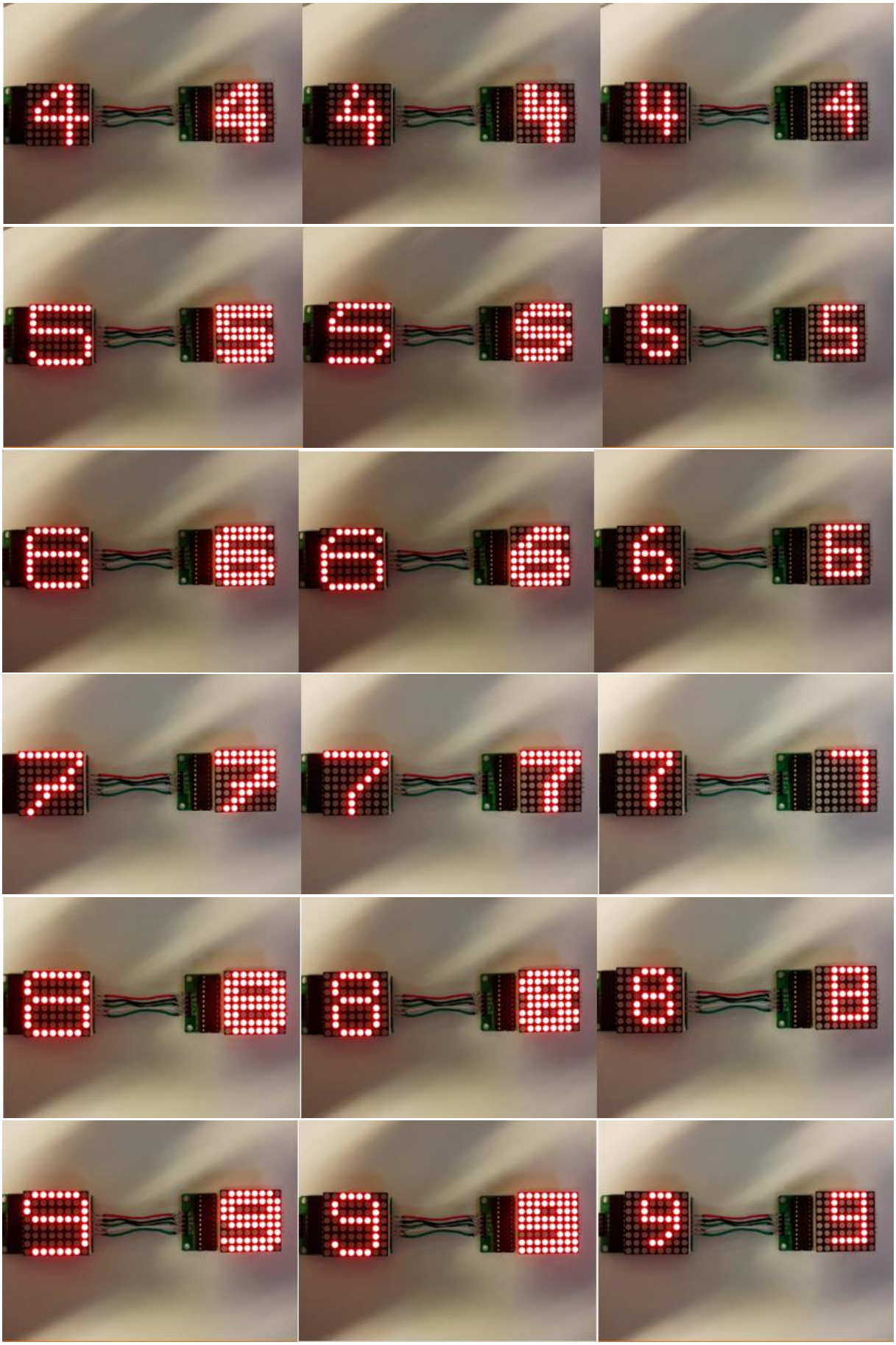
Semi-Supervised. (n.d.). Retrieved from http://scikit-learn.org/stable/modules/label_propagation.html

Template matching. (2015, November 20). Retrieved December 17, 2015, from https://en.wikipedia.org/wiki/Template_matching

Appendix

Appendix A Images of all numbers






TTL
1.2345678
M
Hz


09
Hz

Appendix B Multi Layer Perceptron

Perceptrons are models of neurons in brains. Perceptrons get certain inputs *X* (see Figure 36*)*. These inputs are called nodes. Some nodes are more important than others for reaching a specific results.

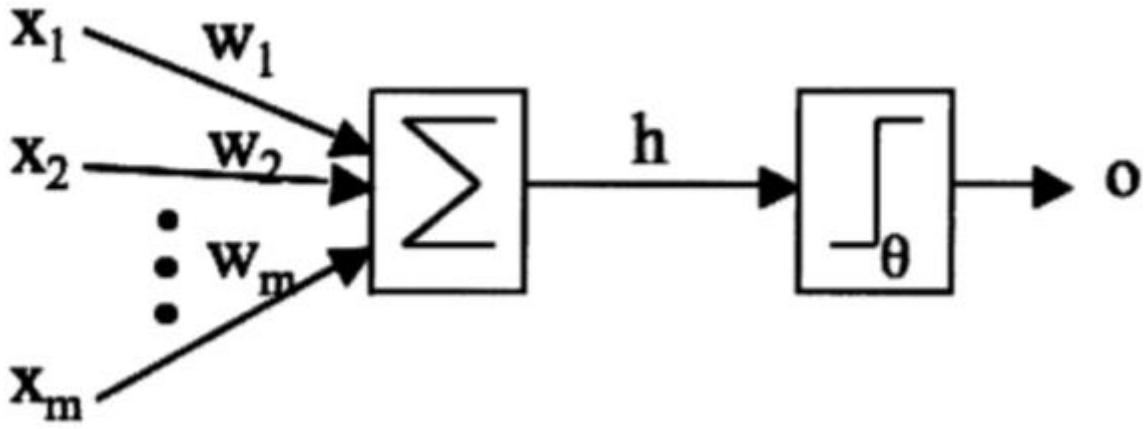


Figure 36: Architecture of a perceptron

(Source: Stephen Marsland, Machine Learning: An Algorithmic Perspective)

This importance is given with a certain weight *ω* (*W* in the image). All inputs are multiplied with their corresponding weight and the results are summed. The outcome *h* is put in an activation function (in this case a threshold) that says the perceptron has to fire or not, depending on the value of *h* is. When speaking of firing with a threshold, 0 is a no and 1 is a yes. The formula to calculate h is:

$$\mathrm{h} = \sum_{i=1}^{m} \omega_i x_i$$

One perceptron is able to make linear regressions. Simple problems with one type of output can be solved with a perceptron, like OR problems. For example, with an OR problem there can be two inputs. When they both are zero, the *h* will be zero too. If threshold is <1, this means that one or both inputs is a 1, the output after the threshold will also be a one. If both outputs are 0, the output will also be zero.

With an XOR this is not the case. It is not possible to solve this problem with only one perceptron, because the problem is not linearly solvable. Multiple perceptrons will be needed, which can produce non-linear solutions.

The Multi Layer Perceptron consists of multiple perceptrons and is build out of multiple layers. There are as many nodes in the input layer as there are input variables and the same goes for the output. The middle layer is a hidden layer. The amount of nodes can differ per situation (Figure 37).

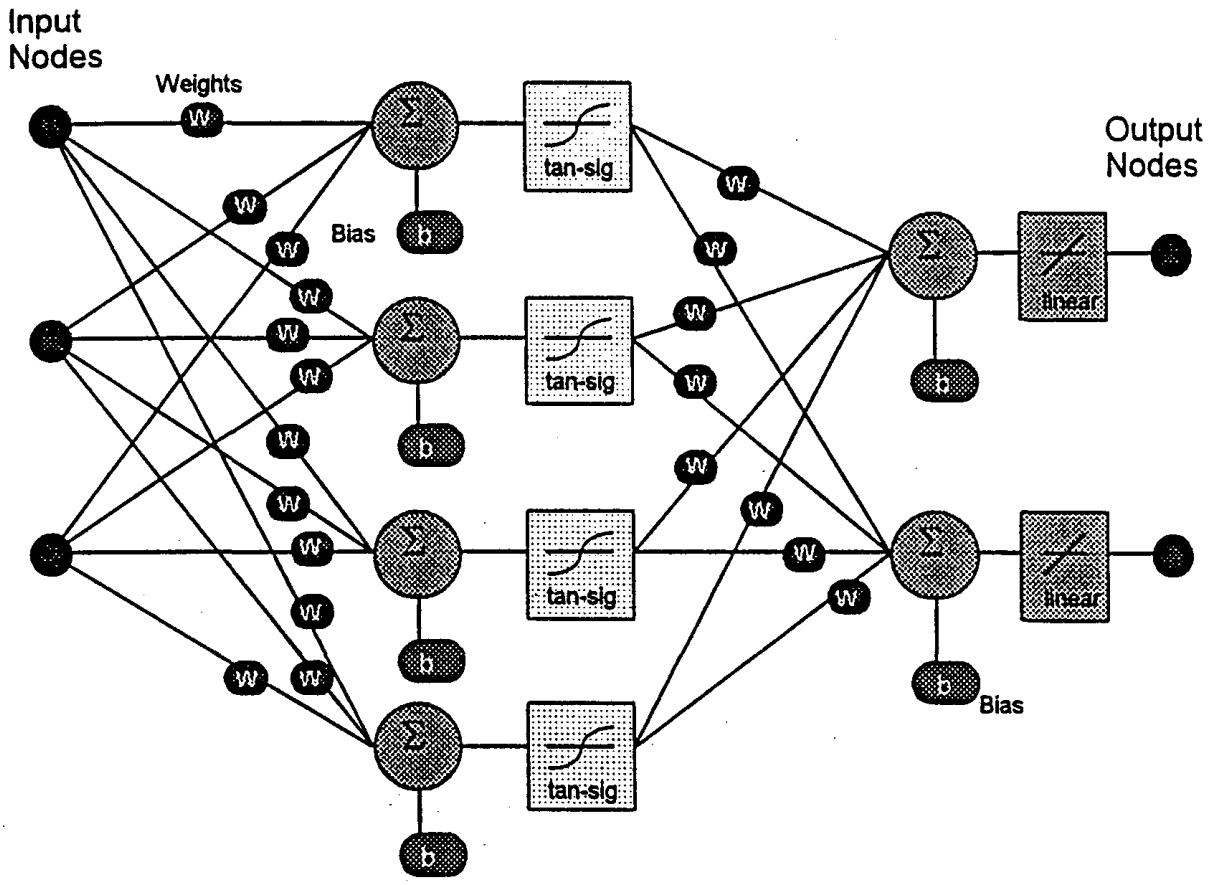


Figure 37: Complete schematics of a Multi Layer Perceptron. The Input Layer, Hidden Layer and the Output Layer are all visible

Figure 4: Architecture of a Multi Layer Perceptron

(Source: https://www.google.com/patents/WO1999053350A1?cl=en*)*

The advantage of Multi Layer Perceptrons is that it is now possible to solve non-linear problems (Figure 37). The same thing happens as with a single perceptron to calculate *h*, but the activation function is different for the hidden layer. Instead of a threshold, a sigmoid or hyperbolic tangent function is used. The end values are different, but they both can be used. Sigmoid gives a value from 0 to 1, but sometimes the output of a hyperbolic tangent function can be of better use. In this case, the sigmoid would be easier because a percentage of how sure the prediction of a number is can be given quite easily with a value from 0 to 1.

The reason a sigmoid or tangent is used as activation function, is that when multiplying the result of the hidden layer when using a threshold, the result could be a zero, while this may be very close to becoming a one. This could give a completely wrong outcome in the output layer when the node is important for the calculations. Therefore a sigmoid or tangent function is used. If a small value is still passes and multiplied with the weight, it can be of significant use to the answer in the output layer. A sigmoid function looks as follows:

$$a = g(h) = \frac{1}{1 + \exp(-h)}$$

The bias *b* in the figure is implemented for the same reason. May the sum after a layer be 0, then the bias will prevent the value from being zero in the rest of the calculations.

Using a MLP

In the case of this project, let say there are some images with 28 by 28 pixels. The objective of the project is to adjust the weights in the function to make a MLP that can predict what numbers is presented in an image.

There are 28*28 input nodes and ten output nodes (zero to nine). The hidden layer can contain as many perceptrons as the user wants. Using more perceptrons does not necessarily mean that you will get a better result. This also should be tested, but this will be dealt with later on.

To make a MLP learn, the variables of the perceptron need to be able to change when having a specific answer *y* that is calculated as output value of the perceptron. These variables that need to be changed are the weights and the biases. When determining how wrong the system is, it is checked how big the error is in the end result, compared to the true target value *t*. This is where **back propagation** comes in. Certain different types of error calculations could be used, but one used in many different MLPs is the sum-of-squares function. The difference between *t* and *y* is calculated, squared and summed:

$$E(t,y) = \sum_{k=1}^{n} (t_k - y_k)^2$$

(source: Stephen Marsland, Machine Learning: An Algorithmic Perspective)

The weights between the output and hidden layer are updated and then the error of the hidden layer can be calculated in exactly the same way. With this error the weights between the input layer can be updated. The updating happens with these formulas:

Updating the output layer: $w_{jk} \leftarrow w_{jk} + \eta\delta_{ok}a_j^{\text{hidden}}$

Updating the hidden layer: $v_{ij} \leftarrow v_{ij} + \eta\delta_{hj}x_i$

(source: Stephen Marsland, Machine Learning: An Algorithmic Perspective)

η is the learning rate. This indicates how fast the MLP should learn and change how big the steps must be in with which the weights adjust themselves. The bigger they are, the bigger the chance that a minimum error will be missed while adjusting. When the learning rate is too small, it may take forever to find a minimum or it will be stuck in the minimum, instead of the absolute minimum (*figure 5*).

Now the MLP is able to predict the number on this image. When the optimal minimum (Figure 38) is reached, the program has learned as good as possible and mostly has a high prediction rate. However, the minimum in *figure 3* is not the biggest minimum. Because starting point of the weights started at the wrong point and the learning rate is too small, the function is stuck in one of the minima. This is always a risk when training a MLP and therefore it is always wise to run the MLP multiple time when training with different random weights to start with. The chance that the optimal minimum will be found is a lot higher then.

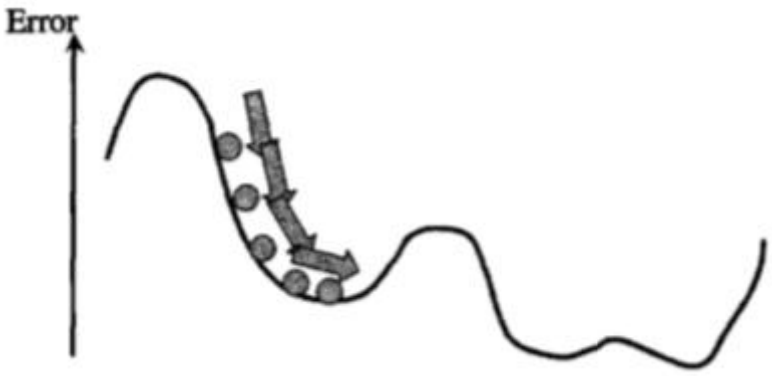


Figure 38: Slope with error reaching a minimum

(source: Stephen Marsland, Machine Learning: An Algorithmic Perspective)

A big problem that can occur with Multi Layer Perceptrons is overfitting. Overfitting means that when the network learn to much, it will eventually also learn that the noise is part of the number on the image, so the user needs to make sure there is not too much training data available for the learning network. Multi Layer Perceptrons, mostly needs thousands of images before they show a good result. The right moment to stop is just before the top of the learning curve.

The code

The code that has been used for the learning software has been largely based on the software from the website http://neuralnetworksanddeeplearning.com/index.html . I have made my own implementation of the code so it was easier for me to make adjustments to it for the number recognition software.

The parameters that need to be set for the software are the:

- Amount of input nodes. This is the amount of pixels there are in the image. For this project that number is a constant value of 785
- Amount of hidden nodes.
- Amount of output nodes. This is the amount of numbers that need to be recognised, so ten numbers.
- The learning rate. This determines how big the steps are with which the learning rate changes. (See the formula for updating the hidden and output layer its weights and biases)
- Amount of epochs. Every epoch is a learning round in which it can improve its weights and biases once.
- Training and test data. This can be changed to whatever the user wants. The current settings for the software however are set on the database for images of 28 by 28 pixels.

Appendix C First plan pre-processing

In a perfect situation, the camera and the display stand opposed to each other with perfect lighting to get a perfect image. In this project, to satisfy the requirements, the software also needs to recognise numbers that are rotated with a maximum of 45 degrees. To make this possible, some pre-processing needs to be done. Pre-processing means that certain methods are used to make the inputs of a system more suitable for the application. In this case, this means that the image is made more appropriate for the learning algorithm.

The complete pre-processing consists of multiple actions

- Thresholding the image so only the numbers are presented
- Blur the image to delete noise from the image
- Rotate the image so that the numbers are presented horizontally
- Separate every number and make an image of it
- Scale the image of the numbers

Take Figure 39 as an example of how images with numbers can be presented in an extreme case. This thresholded image represents a rotation in all direction on the Euclidian-axis. But is directly becomes clear that the numbers are still separated in some way, despite the rotation. The image is greyscale, but the zero value is blue in this image and the maximum value of 255 is red.

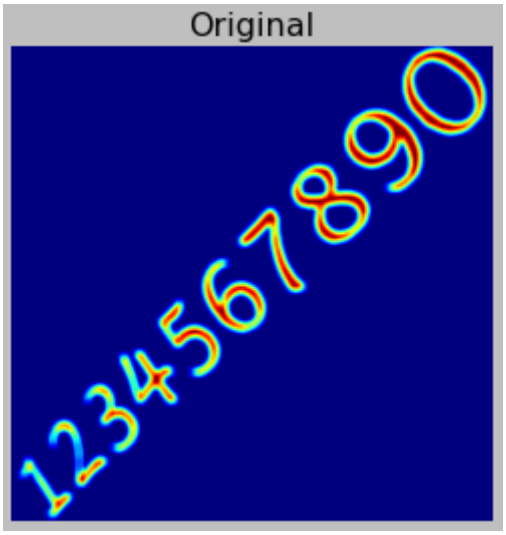


Figure 39: image of numbers thresholded and rotated in x-, y- and z-direction.

Rotate the complete image

The most important rotation is that the numbers are put horizontally. Then it is easier to separate all numbers with a horizontal line and present the position of the numbers in a histogram. With digital displays, the numbers are almost always separable by a vertical line when the image is standing right. Since the testing displays are standard digital displays, this feature was thought to be the best method for extracting numbers from images.

To find the position of all numbers, two histograms are used. One for the x-axis and one for the y-axis. They represent the amount of pixels with a value of one or more in the image per x- or y-coordinate (Figure 40).

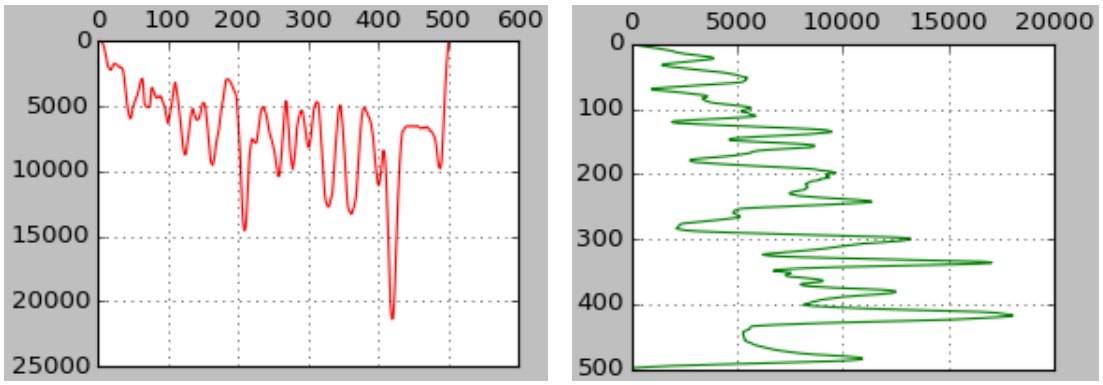


Figure 40: A histogram of x-axis of the image (left) and a Histogram of the y-axis of the image

Separating the numbers from this image is hard, if not almost impossible, but now that the minimum and maximum x and y are known for the position of the numbers, it will take less processing time for the program to search for numbers, because it can look on a specific location in the image.

With every y-coordinate where a pixel with a value of one or more is found, the median value is taken and put in a new graph (Figure 41). With a graph like this it can roughly be said what the direction is the numbers are standing in. By using linear regression a line is drawn in the formula form $y = ax + b$. The rotation on the image then happens with the tangent rule. The result is shown in Figure 42.

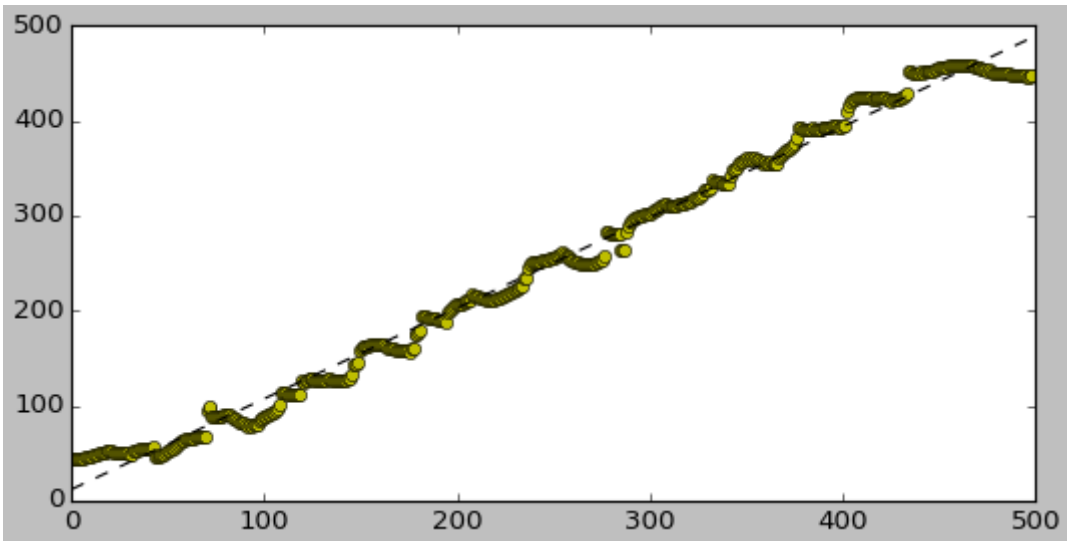


Figure 41: The median values of all black pixels in every column in the thresholded image

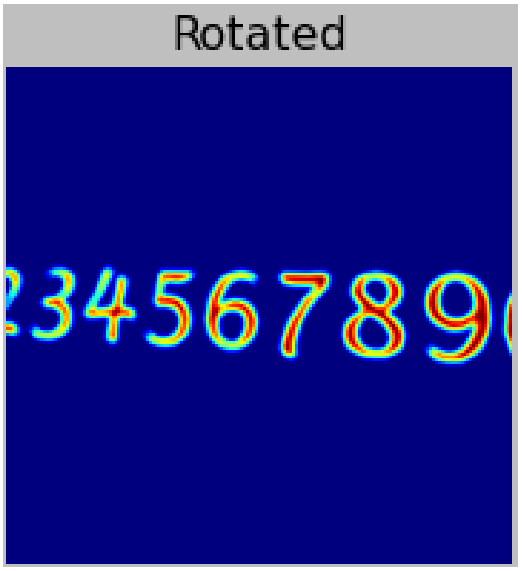


Figure 42: The image is rotated with the angle that the line in figure 9 made with the horizontal x-axis.

Figure 43 and Figure 44 show that by doing the previous rotation, the numbers are separable. By locating the minimum and maximum x and y of every “mountain” in the histogram of the x-axis, the space in which the number lays can be cut out and used for the learning method. To make sure that the software will recognise the numbers, the test set will also contain numbers that are slightly skewed. This will prevent the program from having to skew every number it wants to recognise because it has learned that is can be presented like this.

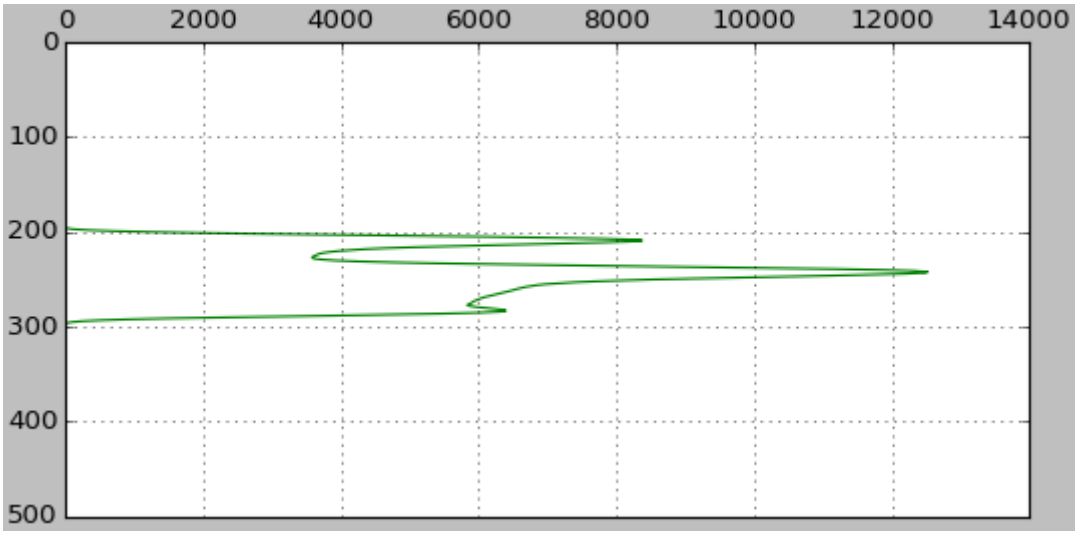


Figure 43: Histogram of the y-axis of the rotated image

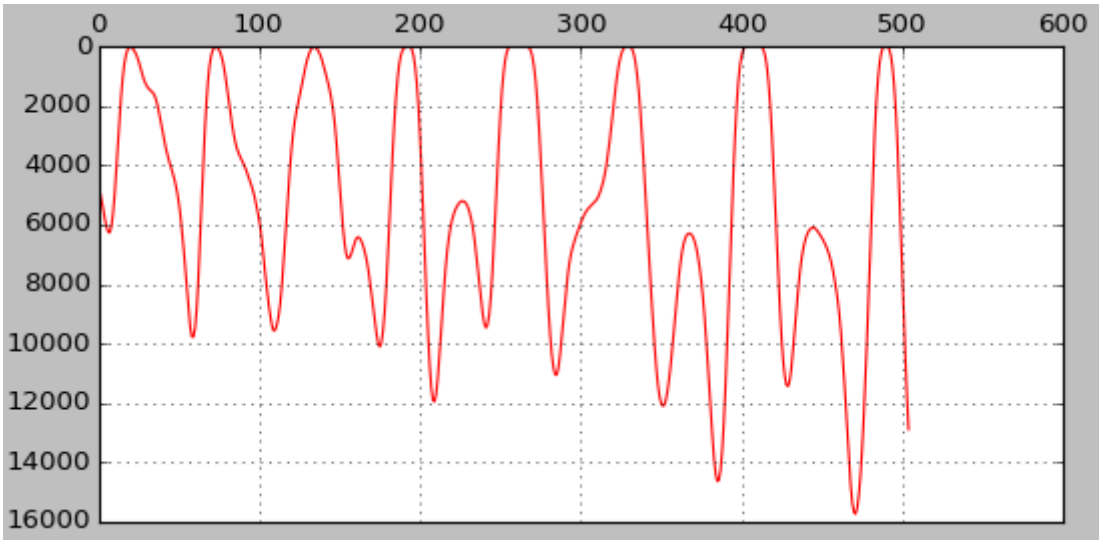


Figure 44: Histogram of the x-axis of the rotated image

In the rotated picture it is visible that the numbers become bigger from left to right and that the numbers are slightly skewed. To make sure the difference in size is not a problem for the recognition software, the size of the image of the numbers is resized.

The skewness of the image will be dealt with by putting slightly skewed numbers in the trainings set. It will then learn to recognise straight and skewed numbers.

Why did it not work?
The main reason this method did not work, was because of two reasons.

1. <u>The numbers are not always separable by a linear line in the image</u>

When the camera stand directly perpendicular to the display, the software will almost always be able to find a space between the numbers with a vertical line through them with zero pixels of a high value. This changes when the camera is rotated (Figure 45)

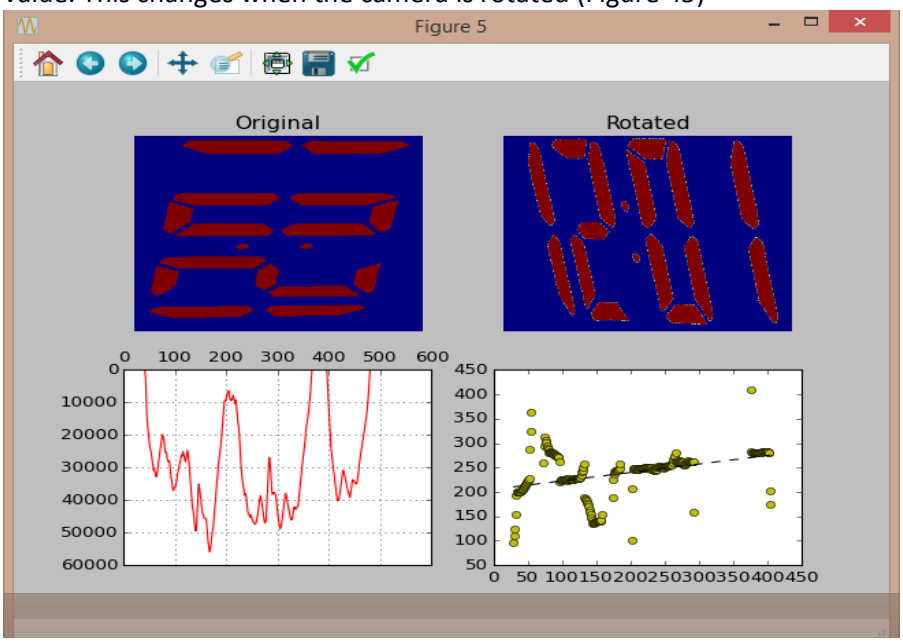


Figure 45: An image of a display, with a rotated camera

The rotation of the image happens perfectly, but due to the rotation of the camera, the image is severely skewed. There is still a clear separation between all numbers, but if this must be found, the software should rotate the vertical line that checks for these separations every time it gets on a pixel. This is very time consuming and it may not even be reliable. It could happen that the line goes directly through the open space between two segment of a numbers for example and this would give major problems with locating the numbers in the image.

2. When there were less than 3 numbers in the image, doing a rotation could form problems.

The software was able to rotate the image, despite the amount of numbers in the image. The accuracy however went down rapidly when the amount of numbers became less (see Figure 46, Figure 47 and Figure 48). Note, the numbers are not completely in the image, but this had no influence on the rotation, because it happened afterwards. The images are just to give an illustration of the situation.

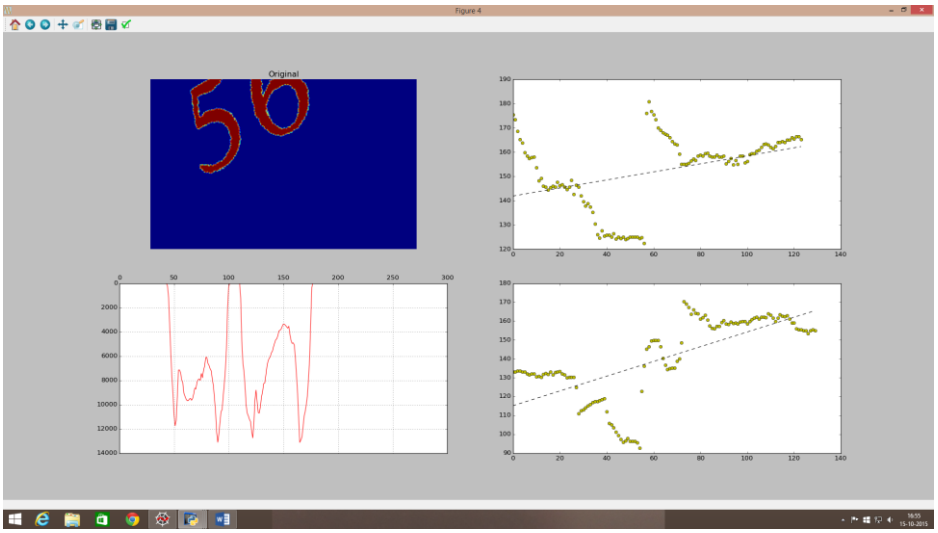


Figure 46: Rotation of two numbers in the image

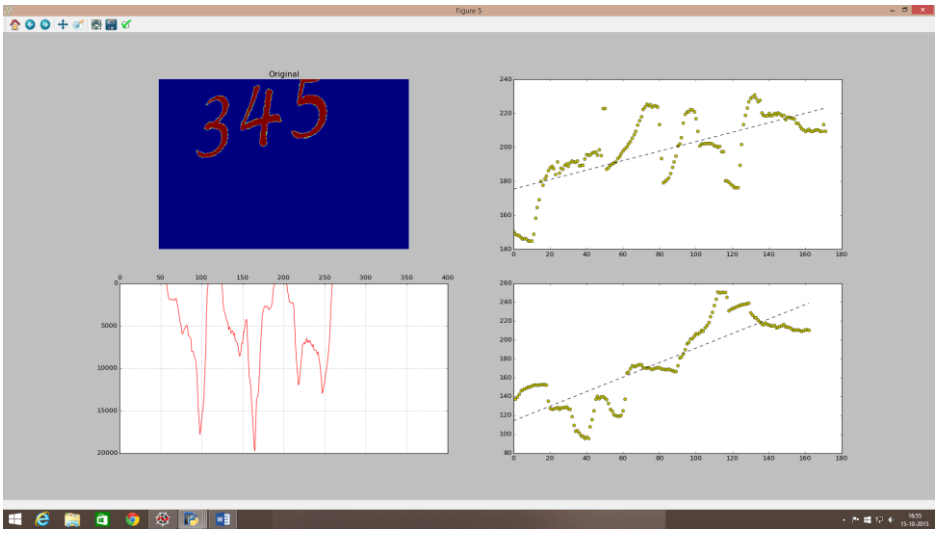


Figure 47: Rotation of three numbers in the image

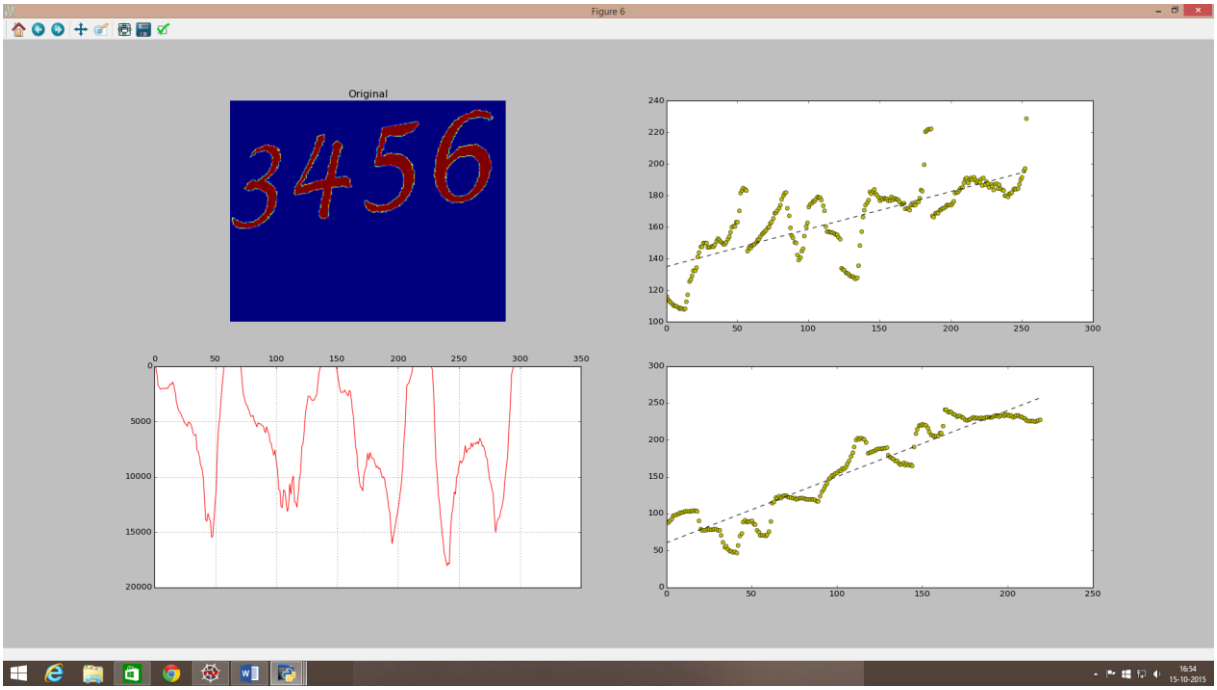


Figure 48: Rotation of four numbers in the image

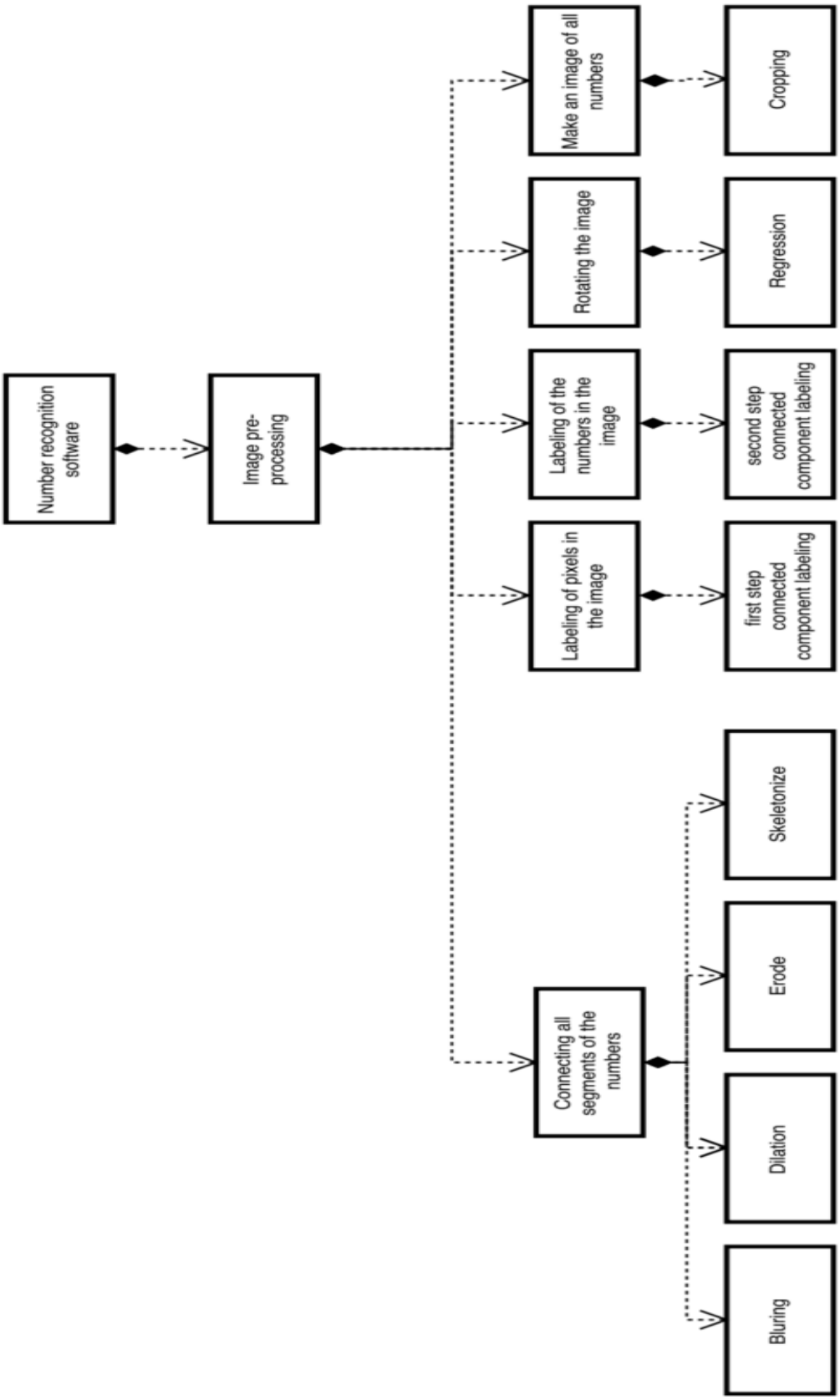
The problem was that depending on the number, the amount of pixels with a high value (red) for every vertical line was measured for the rotation, and then the median value was taken an put in the histogram. This meant that of there were only two numbers, the shape is a really important influence on the rotation. It can be seen that Figure 45 and Figure 48 both have four numbers, but with the Figure 45, the numbers are on top equal to the bottom, unlike the number in Figure 48.

In the case of these numbers, they were still separable with a linear line as was shown by the histogram. However, the fact that the rotation was not reliable, made this method unsuitable for the project. It was unpredictable if the pre-processing would do its work.

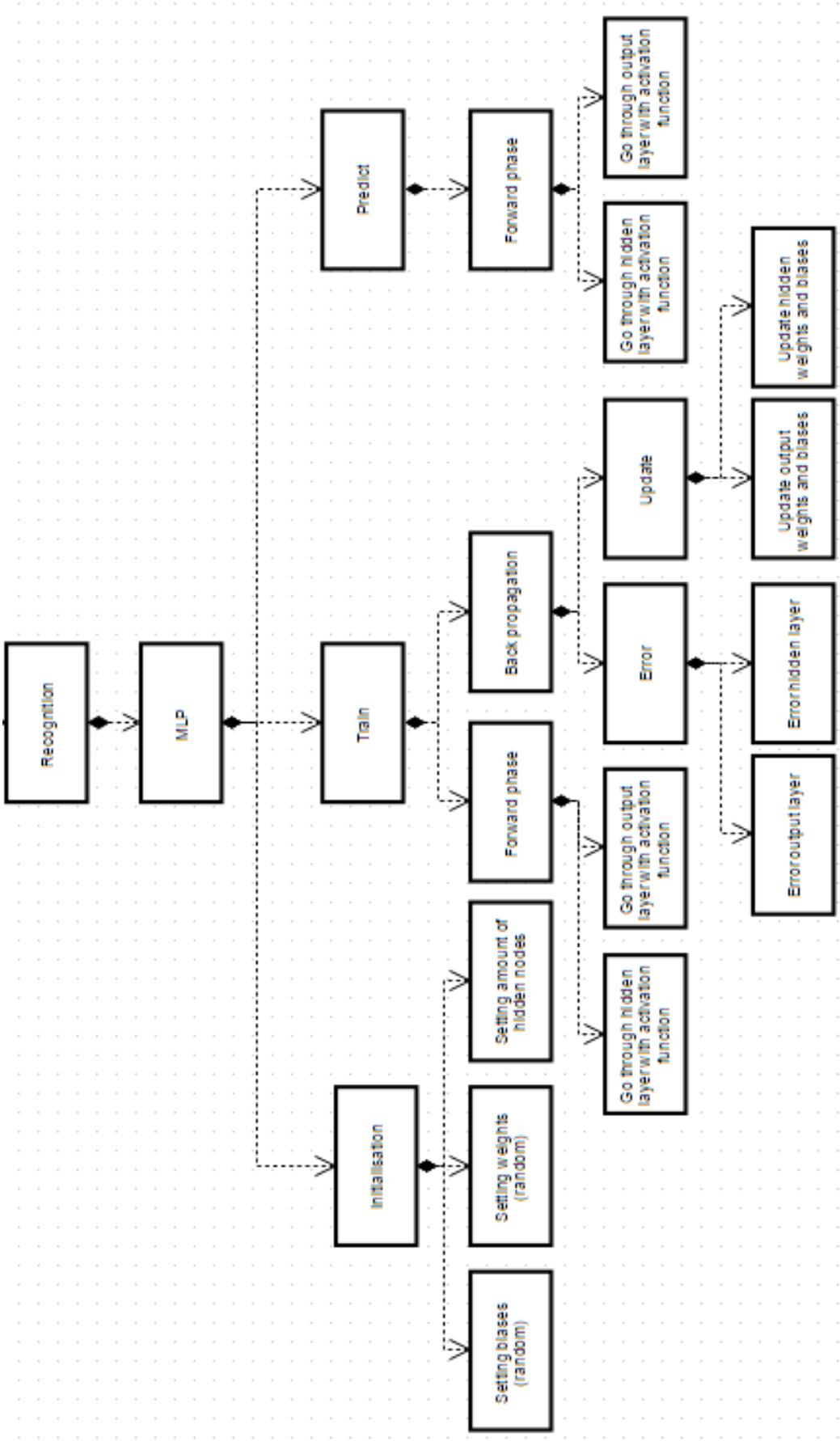
These are the reasons that this form of filtering out the numbers has not been used in the project.

Appendix D SysML diagrams

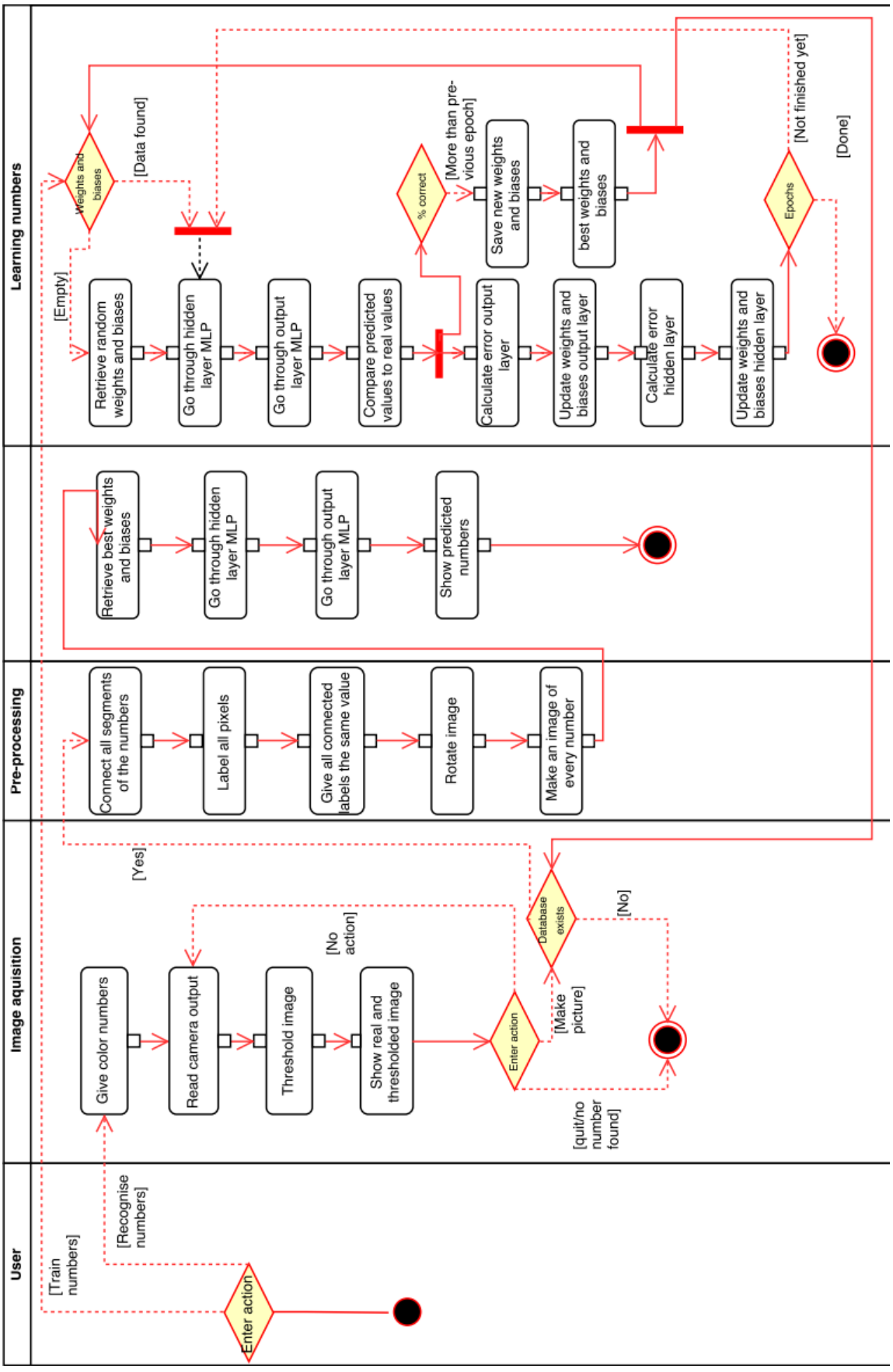
D1: block definition diagram of the Image pre-processing



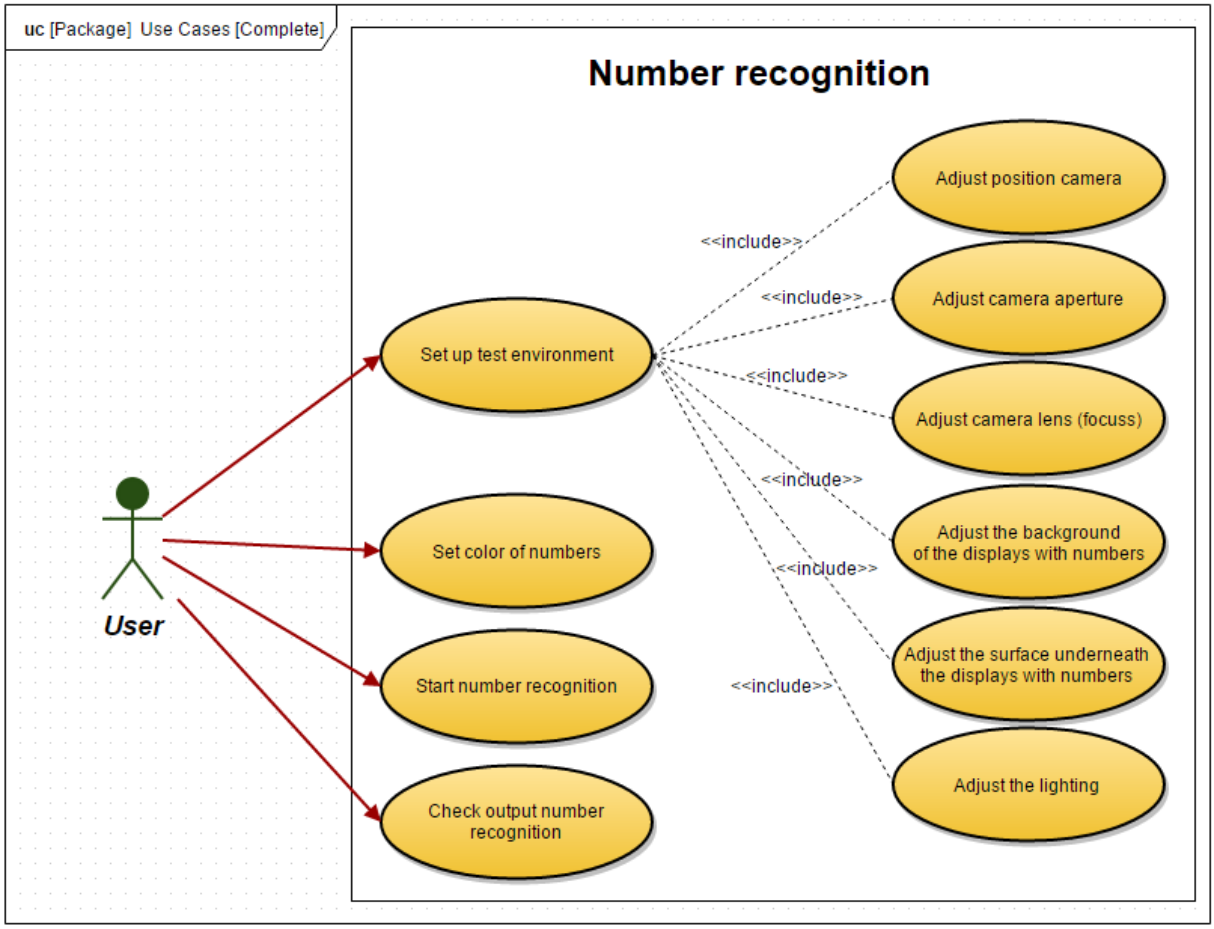
D2: block definition diagram of the Recognition parts of software



D3: Activity diagram of the number recognition software



D4: Use case diagram of the number recognition software



Appendix E Specifications of the Basler A312fc camera

SPECIFICATIONS

PRODUCT SPECIFICATIONS

BASLER A310 SERIES

Features/Benefits

- Superior image quality improves your image processing results
- Super compact size reduces the space needed in your installation
- 100% factory testing ensures consistent product quality
- Area of Interest (AOI) scanning allows higher frame rates
- Choice of resolutions maximizes your system design flexibility
- Electronic exposure time control provides maximum flexibility

BASLER VISION COMPONENTS

Description

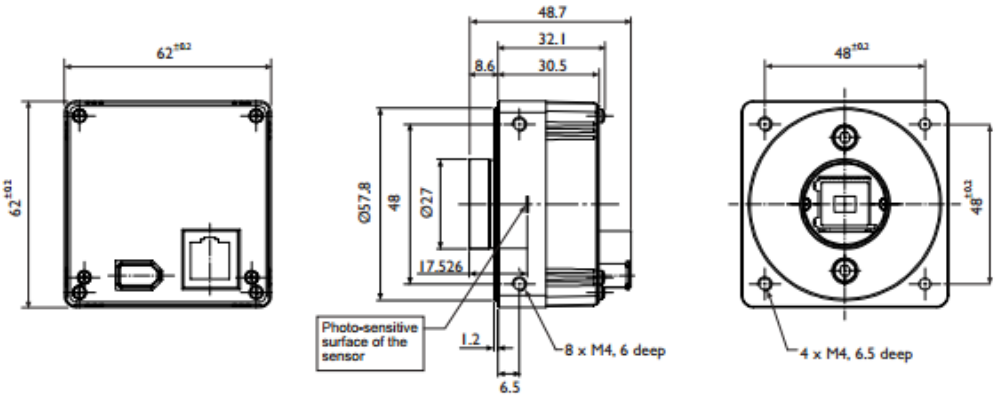
The A310 Series of high-performance, digital cameras is ideal for a variety of industrial applications. The cameras can be triggered via an external sync signal or run in an internally controlled "free-run" mode. A310 cameras operate with a single voltage power supply and have the advantage of remarkably simple cabling requirements. A combination of features such as digital shift, and test images, ensure that these versatile cameras provide an exceptional price/performance ratio.

Applications

- Semiconductor and component inspection
- Manufacturing quality control
- Food and beverage inspection
- Microscopy and medical imaging
- Biometrics
- Many other vision applications

BASLER VISION TECHNOLOGIES

Dimensions (in mm)



Specifications	A311f	A311fc	A312f	A312fc
Sensor Size (H x V Pixels)	659 x 494	658 x 492	782 x 582	780 x 580
Sensor Type	Progressive CCD			
Pixel Size (in µm)	9.9 x 9.9	9.9 x 9.9	8.3 x 8.3	8.3 x 8.3
Max. Frame Rate at Full Resolution	73 fps	73 fps	53 fps	53 fps
Color / Mono	Mono	Color	Mono	Color
Video Output Format	Mono8, Mono16 (12bpp)	Mono8, Mono16 (12bpp), Raw8, Raw16 (12bpp), YUV422	Mono8, Mono16 (12bp	Mono8, Mono16 (12bpp), Raw8, Raw16 (12bpp), YUV422
Synchronization	Via external trigger or the 1394 bus			
Exposure Control	Programmable via 1394 bus			
Power Requirements	12 VDC (8 to 36 VDC), max. 3.0 W			
Lens Mounts	C-mount			
IR-Cut Filter	Optional	Standard	Optional	Standard
Housing Size (L x W x H)	40.8 mm x 62 mm x 62 mm			
Weight	Ca. 240 g			
Conformity	CE, FCC			

Specifications are subject to change without prior notice.

BASLER VISION TECHNOLOGIES

0604

Germany, Headquarters
Phone +49 4102 463 500
Fax +49 4102 463 599
vc.sales.europe@baslerweb.com

USA
Phone +1 610 280 0171
Fax +1 610 280 7608
vc.sales.usa@baslerweb.com

Singapore
Phone +65 6425 0472
Fax +65 6425 0473
vc.sales.asia@baslerweb.com

www.basler-vc.com

Source: http://www.graftek.com/pdf/Brochures/basler/A310.pdf

Appendix F Specifications of the Navitar NMV-5WA camera lens

Back Focal Length	9.960
EFL (mm)	4.5
Field Angle 1/2 (HxV)	79 x 59.4
Field Angle 1/3 (HxV)	59.7 x 45.1
Focus Control	Manual
Focusing range from front of lens (m)	0.2 - inf.
Format	1/2"
F Stop	1.4-16
Iris Control	Manual
Mount	C
Object Area at M.O.D. (HxV) 1/2"	260h x 180v
Object Area at M.O.D. (HxV) 1/3"	195h x 135v
Object Area at M.O.D. (HxV) 1/4"	130h x 90v

Source: https://navitar.com/products/imaging-optics/low-magnification-video/navitar-machine-vision/navitar-machine-vision-12-format/

Appendix G Specifications of the function generator

20 MHz/10MHz/7MHz/4MHz DDS FUNCTION GENERATOR

SFG-2100 Series (20/10/7/4 MHz)

CE

SFG-2000 Series (20/10/7/4 MHz)

CE

SIGNAL SOURCES

Based on the Direct Digital Synthesized (DDS) technology and unique FPGA design, the SFG-2000/2100 Series Function Generators are built with exceptionally high performance far exceeding that of any conventional function generators at a very competitive price. Stable output frequency, low distortion, and fine frequency resolution are the most remarkable characteristics of this product series.

The SFG-2000/2100 Series include four members in each family at 4MHz, 7MHz, 10MHz and 20MHz bandwidth(perivd), The SFG-2100 Series have additional functions of Sweep, AM/FM modulation, and External Counter. As a result of the±20ppm stability level and output waveform accuracy, The SFG-2000/2100 Series well fit into a wide variety of applications, such as signal generator for experiment labs, reference signal for PLL (Phase Locked Loop), and calibration and adjustment source for electronic devices.

FEATURES

* DDS Technology and FPGA Chip Design
* Frequency Range:0.1Hz~4/7/10/20 MHz
* High Frequency Accuracy : ±20ppm
* High Frequency Stability : ±20ppm
* Frequency Resolution : 100mHz
* Low Distortion Sine Wave : -55dBc, 0.1Hz ~ 200kHz
* Front Panel Setting Save/Recall with 10 Groups of Setting Memories
* Built-in 9 Digits, 150MHz/High Resolution Counter (SFG-2100 Series Only)
* INT/EXT AM/FM Modulation (SFG-2100 Series Only)
* LIN/LOG Sweep Mode (SFG-2100 Series Only)

SPECIFICATIONS

	SFG-2000 Series				SFG-2100 Series			
MAIN								
	SFG-2004	SFG-2007	SFG-2010	SFG-2020	SFG-2104	SFG-2107	SFG-2110	SFG-2120
Frequency Range(For Sine, Square)	0.1Hz~ 4MHz	0.1Hz~ 7MHz	0.1Hz~ 10MHz	1Hz~ 20MHz	0.1Hz~ 4MHz	0.1Hz~ 7MHz	0.1Hz~ 10MHz	1Hz~ 20MHz
Range(For Triangle)	0.1Hz~1MHz (1Hz ~ 1MHz for SFG-2020/2120)							
Resolution	0.1Hz (1Hz for SFG-2020/2120)							
Stability	± 20 ppm							
Accuracy	± 20 ppm							
Aging	± 5 ppm / year							
Output Function	Sine, Square, Triangle							
Amplitude Range	2mV ~ 10Vpp (into 50Ω load)							
Impedance	50Ω±10%							
Attenuator	-20dB±1dBx2							
DC Offset	<-5V ~ >+5V (into 50Ω load)							
Duty Control	20% to 80% , 2Hz ~1MHz (Square wave only)							
Range Resolution	1%							
Display	9 digits LED display							
SINE WAVE								
Harmonics Distortion	-55dBc,0.1Hz~200kHz; -40dBc,0.2MHz~4MHz; -30dBc,4MHz~10MHz (Specification applied to both TTL/CMOS OFF and from MAX. to 1/10 level)							
Flatness(Relative to 1kHz)	<±0.3dB, 0.1Hz~1MHz; <±0.5dB, 1MHz~4MHz; <±2dB, 4MHz~10MHz							
TRIANGLE WAVE								
Linearity	≥98%,0.1Hz~100kHz;≥95%,100kHz~1MHz							
SQUARE WAVE								
Symmetry	±1% of period +4ns, 0.1Hz~100kHz							
Rise or Fall Time	≤25ns at maximum output.(into 50Ω load)							
CMOS OUTPUT								
Level	4Vpp±1Vpp~15Vpp±1Vpp adjustable; Rise or Fall Time≤120ns							
TTL OUTPUT								
Level	≥3Vpp							
Fan Out	20 TTL load							
Rise and Fall Time	≤25ns							
SWEEP OPERATION								
Rate	–				100:1 ratio max. and adjustable(*)			
Time					1 Sec~30 Sec adjustable(**)			
Mode					Lin./Log. switch selector			
AMPLITUDE MODULATION								
Depth & Modulation	–				0~100%; 400Hz(INT),			
Frequency					DC~1MHz(EXT)			
Carrier BW					100Hz~5MHz(-3dB)			
EXT Modulation Sensitivity					≤10Vpp for 100%modulation			
FREQUENCY MODULATION								
Deviation & Modulation	–				≥±50kHz,center at 1MHz,			
Frequency					400Hz fixed(INT),1kHz fixed(EXT)			
EXT Modulation Sensitivity					≤10Vpp for 10% modulation(center at 1kHz)			
FREQUENCY COUNTER								
Range	–				5Hz~150MHz			
Accuracy					Time base accuracy±1count			
Time base					±20ppm(23℃±5℃)after 30 minutes warm up			
Resolution					100nHz			
					for 1Hz; 0.1Hz for 100MHz			
Input Impedance					1MΩ/150pf			
Sensitivity					≤35mVrms (5Hz~100MHz)			
					≤45mVrms (100MHz~150MHz)			

NOTE : 1.(*) In order to get the maximum sweep span, the sweep time needs to be tuned on when adjusting the sweep span.
2.(**) When the sweep time is too long, the stop frequency will reach and stay at the maximum frequency of the instrument until the end of the sweep cycle.

C3 GW INSTEK

SFG-2100 Series

Rear Panel

SIGNAL SOURCES

SPECIFICATIONS

	SFG-2000 Series				SFG-2100 Series			
	SFG-2004	SFG-2007	SFG-2010	SFG-2020	SFG-2104	SFG-2107	SFG-2110	SFG-2120
STORE/RECALL FUNCTION								
	10 groups of panel settings							
POWER SOURCE								
	AC115V ±10%, AC230V+10%/-15%, 50/60Hz							
DIMENSION & WEIGHT								
	266(W)×107(H)×293(D) mm; Approx. 3.1kg				266(W)×107(H)×293(D) mm; Approx. 3.2kg			

ORDERING INFORMATION

SFG-2004 4MHz DDS Function Generator
SFG-2007 7MHz DDS Function Generator
SFG-2010 10MHz DDS Function Generator
SFG-2020 20MHz DDS Function Generator
SFG-2104 4MHz DDS Function Generator with Counter, Sweep & AM, FM Modulation
SFG-2107 7MHz DDS Function Generator with Counter, Sweep & AM, FM Modulation
SFG-2110 10MHz DDS Function Generator with Counter, Sweep & AM, FM Modulation
SFG-2120 20MHz DDS Function Generator with Counter, Sweep & AM, FM Modulation

ACCESSORIES :
User manual×1, Power Cord×1
GTL-101 test lead×1 (SFG-2000 Series)
GTL-101 test lead×2 (SFG-2100 Series)

SELECTION GUIDE

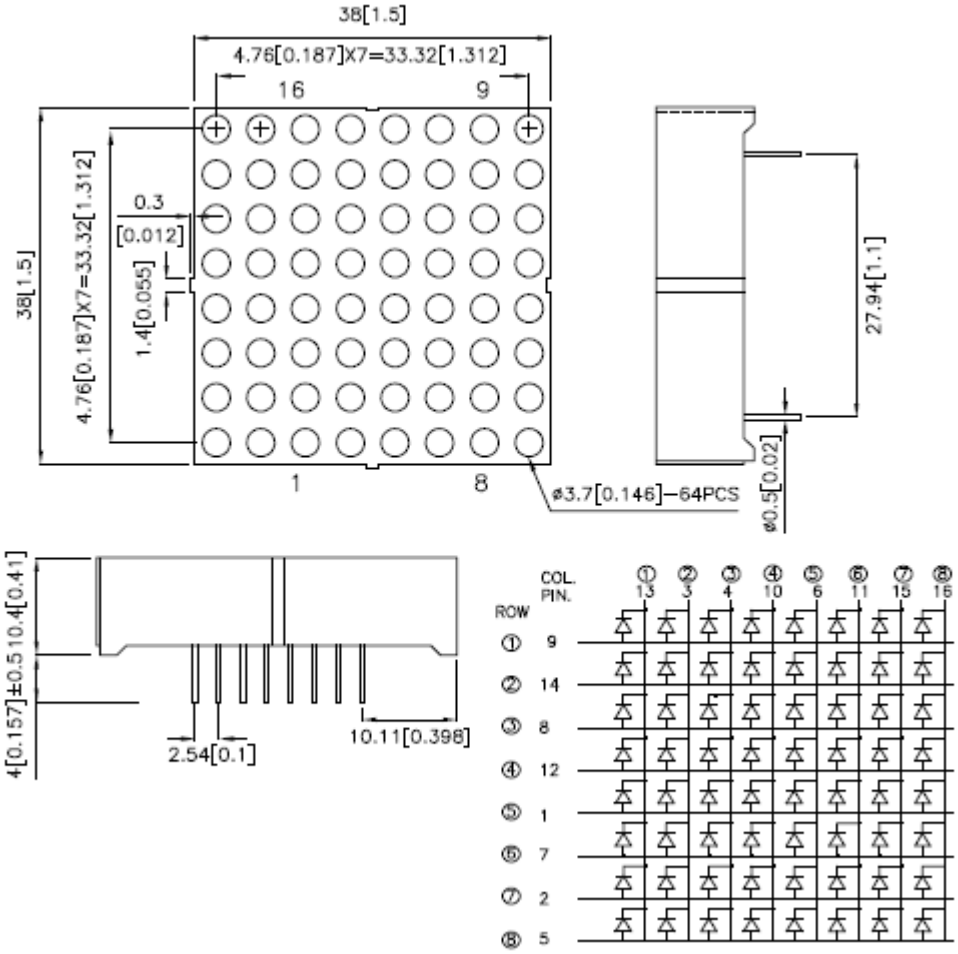
FREQUENCY RANGE	4MHz		7MHz		10MHz		20MHz	
MODEL	SFG-2004	SFG-2104	SFG-2007	SFG-2107	SFG-2010	SFG-2110	SFG-2020	SFG-2120
DUTY	✓	✓	✓	✓	✓	✓	✓	✓
TTL/CMOS	✓	✓	✓	✓	✓	✓	✓	✓
DC OFFSET	✓	✓	✓	✓	✓	✓	✓	✓
LIN/LOG SWEEP		✓		✓		✓		✓
AM/FM MODULATION		✓		✓		✓		✓
EXT COUNTER		✓		✓		✓		✓

Made to Measure. | Good Will Instrument Co., Ltd. C4

http://www.gwinstek.com/en-global/products/Signal_Sources/DDS_Function_Generators/SFG-2000

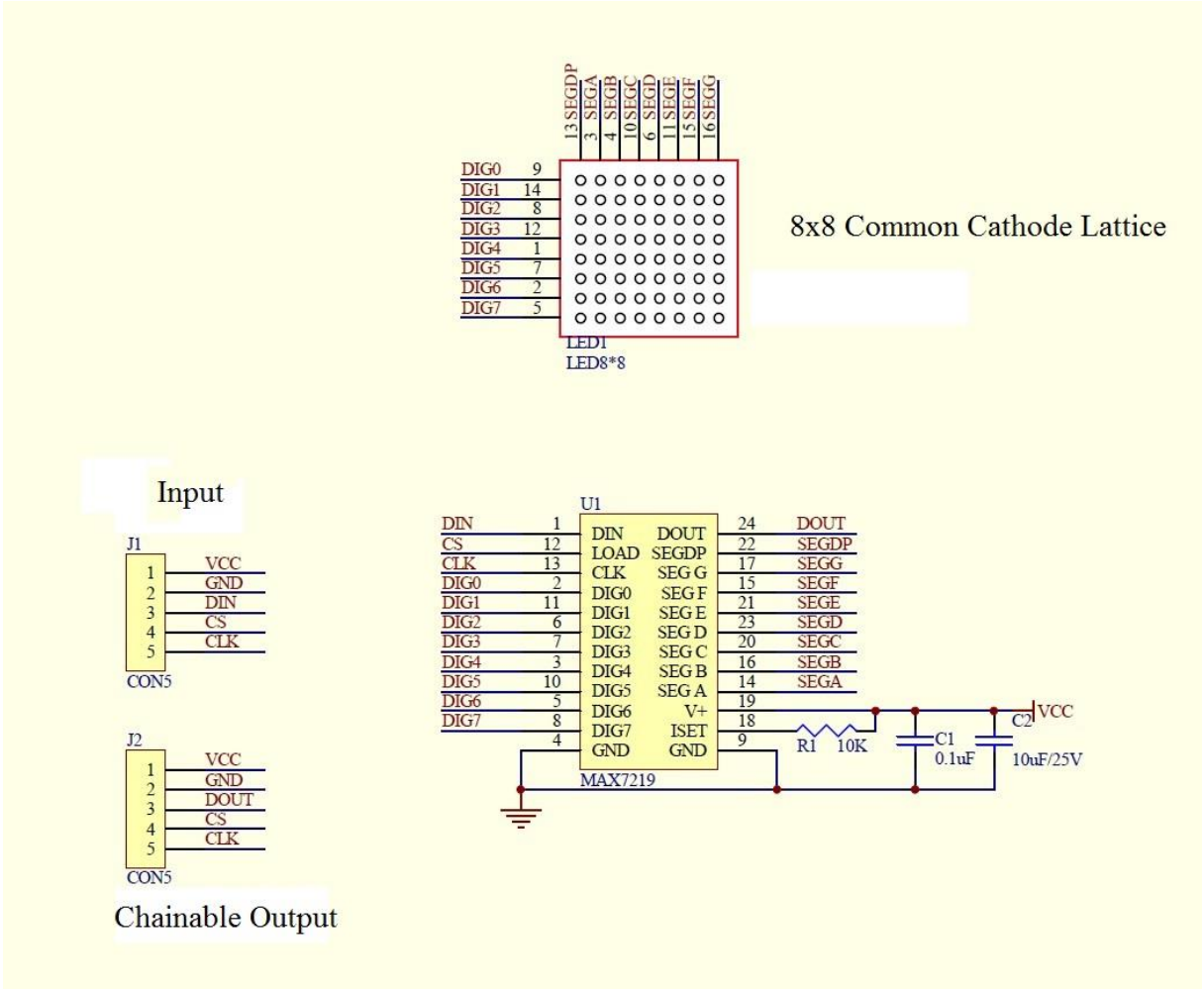
Appendix H 8x8 dot matrix display

Appendix H1 specifications of the 8x8 dot matrix LED display



http://www.best-microcontroller-projects.com/image-files/led-dot-matrix-display-kingbright-tc15-11srwa_490.png

Appendix H2 Dot matrix LED display (8x8) connected to MAX7219 controller



Appendix I Best test results of the number recognition software

results = [(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),
(0, 0), (0, 0), (0, 0), (7, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1),
(1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1),
(1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1),
(1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1), (1, 1),
(1, 1), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2),
(2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2),
(2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (2, 2),
(2, 2), (2, 2), (2, 2), (2, 2), (2, 2), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3),
(3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3),
(3, 3), (3, 3), (3, 3), (2, 3), (3, 3), (7, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3),
(3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (4, 4), (4, 4), (4, 4), (4, 4),
(4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4),
(4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4),
(4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4),
(4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4),
(4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (4, 4), (5, 5), (5, 5), (5, 5), (5, 5),
(5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5),
(5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5),
(5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5),
(5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5),
(5, 5), (5, 5), (5, 5), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (0, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6),
(6, 6), (6, 6), (5, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6),
(6, 6), (6, 6), (6, 6), (6, 6), (5, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6),
(6, 6), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7),
(7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7),
(7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7),
(7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8),
(8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (6, 8), (9, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8),
(8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8),
(8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9),
(9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9),
(9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (3, 9), (9, 9), (9, 9), (9, 9), (9, 9),
(9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (8, 9), (9, 9), (9, 9), (9, 9), (3, 9)]

Epoch 88 : 563 out of 574 correct (98.0836236934 %)

Appendix J Best test results of the single (linear) perceptron

results = [(0, 0), (1, 1), (2, 2), (3, 3), (7, 3), (3, 3), (9, 3), (3, 3), (4, 4), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (9, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (6, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (9, 5), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5), (9, 5), (9, 5), (8, 5), (9, 5), (9, 5), (9, 5), (8, 5), (9, 5), (8, 5), (2, 5), (9, 5), (9, 5), (9, 5), (9, 5), (9, 5), (9, 5), (6, 5), (9, 5), (6, 5), (8, 5), (9, 5), (6, 5), (9, 5), (9, 5), (6, 5), (6, 5), (6, 5), (6, 5), (9, 5), (8, 5), (6, 5), (9, 5), (5, 5), (9, 5), (5, 5), (6, 6), (5, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (6, 6), (7, 7), (0, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (7, 7), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (9, 8), (0, 8), (9, 8), (8, 8), (8, 8), (8, 8), (3, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (6, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (9, 9), (8, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (3, 9), (9, 9), (9, 9), (9, 9), (9, 9), (9, 9), (3, 9)]

Epoch 117 : 526 out of 574 correct (91.637630662 %)

Appendix K Number recognition software

Appendix K1 Image retrieval code

```
"""
################################################################################
Image pre-processing for number recognition (ImageRetrieval.py)
Created on Thu Dec 17 03:45:25 2015

@Company: Carya Automatisering
@Author: Mark Schoneveld

@Study: Mechatronics
@School: The Hague University of Applied Sciences (Delft)
################################################################################
"""

import cv2
import numpy as np

class imgRetrieval:
    def get(self):
        cap = 0
        cv2.destroyAllWindows()
        path = "C:/Users/Carya/Desktop/testenmaar/images/"
        filetype = ".jpg"
        cap = cv2.VideoCapture(0)
        i=1
        counter = 0
        try:
            while True:
                # Capture frame-by-frame
                ret, frame = cap.read()
                hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
                print hsv

                #red, orange & yellow
                lower = np.array([0,180,200])
                upper = np.array([60,255,255])

#               lower = np.array([35,180,0])
#               upper = np.array([100,255,255])
#
                # Threshold the HSV image to get only blue colors

                mask = cv2.inRange(hsv, lower, upper)
                res = cv2.bitwise_and(frame, frame, mask = mask)
                mask = cv2.blur(mask,(i,i))

                cv2.imshow('frame',frame)
                cv2.imshow('mask',mask)
                cv2.imshow('res',res)
```

```
        if cv2.waitKey(1) & 0xFF == ord('p'):
            filename = str(counter) + filetype
            cv2.imwrite(path + filename, mask)
            results = mask
            break
            counter+=1

        if cv2.waitKey(1) & 0xFF == ord('q'):
            results = 0

            break
except KeyboardInterrupt:
    pass
# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()

return(results, path, filename)
```

Appendix K2 Image pre-processing code

```
# -*- coding: utf-8 -*-
"""
################################################################################
Image pre-processing for number recognition (IPP.py)
Created on Thu Nov 12 13:59:49 2015

@Company: Carya Automatisering
@Author: Mark Schoneveld

@Study: Mechatronics
@School: The Hague University of Applied Sciences (Delft)
################################################################################

Description
This program consists of the following classes and functions:

class IPP
x   -def imgageRetr
x   -def noiseRemoval
x   -def connectPartsNrs
x   -def regression
x   -def rotate (2x. The whole number region and for every number)
      *input:image, angle
   -def findNrs
   -def makeImagesNrs
   -def deItalicNrs


   -def histogram
   -def showImage

################################################################################
"""

from skimage.morphology import skeletonize
import math
import cv2
import numpy as np
import os

class imagePreProcessing(object):
   def __init__(self, path, file_name):
      self.img = cv2.imread(path+file_name, 0)
      self.kernel_d_and_r = (3,3)
      self.amount_dilate = 2
      self.amount_erode = 1


#       #Threshold values
```

```
        self.thres_l = 50
        self.thres_h = 255
        self.thres_type = cv2.THRESH_BINARY

    def noiseRemoval(self):
        #A bilateral filter removes noise, but maintains edges in images better than gausian- or median
blur.
        img_orig = self.img
        self.img = cv2.blur(self.img,(5,5))
#         cv2.imshow("1bilateralfilter", self.img)
        img_blur = self.img
        return(img_orig, img_blur)

    def resize(self):
        #Resize image (array) to 500 by 500 pixels, sot he image is always a square
        print len(self.img)
        print len(self.img[0])
        print "bla"

        if len(self.img) > len(self.img[0]):
            length = len(self.img)
            empty_column = np.zeros((length-len(self.img[0])/2))
            for adding_c in xrange(length):
                self.img = np.insert(self.img, 0, empty_column, axis=1)
                self.img = np.insert(self.img, len(self.img[0]), empty_column, axis=1)
            if len(self.img[0]) % 2 != 0:
                self.img = np.delete(self.img[0], (0),axis=1)

        if len(self.img) < len(self.img[0]):
            length = len(self.img[0])
            empty_row = np.zeros(length)

            for adding_r in xrange((length-len(self.img))/2):

                self.img = np.insert(self.img, 0, empty_row, axis=0)
                self.img = np.insert(self.img, len(self.img), empty_row, axis=0)
            if (len(self.img[0])+(len(self.img))) % 2 != 0:
                self.img = np.insert(self.img, 0, empty_row, axis=0)

        img_resize = self.img
        return(img_resize)

    def threshold(self):
        #Convert to numpy array and gray-values
        """Threshold image
        #img = cv2.threshold(img, low, high, type_of_thresholding)
        """
        _,self.img = cv2.threshold(self.img, self.thres_l, self.thres_h,self.thres_type)

        img_thres = self.img
        return(img_thres)
```

```
    def connectPartsNrs(self):
        #Give kernel for dilation and erosion
        kernel = np.ones(self.kernel_d_and_r,np.uint8)

        #First dilate the image and then erode. With this, the dots can be connected.
        #When the dots are not connected yet, the user can adjust the kernel
        self.img = cv2.blur(self.img,(3,3))
        self.threshold()
        self.img = cv2.dilate(self.img, kernel, iterations = 2)
        self.img = cv2.erode(self.img, kernel, iterations = 2)
        self.img = ((skeletonize(self.img/255)).astype(np.uint8))*255

        img_connected = self.img

        return(img_connected)


    def regression(self):
        y_white = []
        x_white = []

        #For every row
        for i in xrange(len(self.img[0])):
        #Create empty list
            y_help = []
            #For every column
            for j in xrange(len(self.img)):
                #if a[i][j] is above 0
                if self.img[i][j]>0:
                    #add x coordinate j to list c
                    y_help.append(float(j+1))

#           Give y_white the values of y_help (2D-matrix)
#           Give x_white the value of i (1D-matrix)
#           If y_help is empty, don't do anything
            if y_help:
                y_white.append((sum(y_help))/float(len(y_help)))
                x_white.append(i)

#       Make regression line with pointcloud of position of numbers
        x_white, y_white = zip(*sorted(zip(x_white, y_white)))
        fit = np.polyfit(x_white,y_white,1)
        fit_fn = np.poly1d(fit)

        return (fit, fit_fn)


    def rotate(self, fit):
        #If "foto" is entered, i = 1 and the foto is a seperate number
        foto = self.img
```

```
        #Get amount ofrows and columns of foto
        rows,cols = foto.shape
        #Get rotation matrix according to the "fit"
        if fit[0]<=0:
            rot_ofset = 90
        elif fit[0]>0:
            rot_ofset = 90

        r_matrix = cv2.getRotationMatrix2D((cols/2,rows/2),(-
(math.degrees(math.atan2(1,(fit[0])))))+rot_ofset,1)
        #Rotate foto
        foto_rotated = cv2.warpAffine(foto,r_matrix,(cols,rows))

        self.img = foto_rotated

        return(foto_rotated)

    def labelPixelsOne(self):

        for j in xrange(len(self.img)):
            self.img[0,j] = 0

        for i in xrange(len(self.img)):
            self.img[i,0]

        #Create an empty matrix with the size of self.img for the labels
        foto_labels = np.zeros([(len(self.img)),len(self.img[0])])
        #Create empty list to store connected label numbers
        connected = []
        #label to give to number
        label = 1

        #Make sure the first row and column (0) are not being processed
        i=1
        j=1

        #From row 1 to the single last row
        for i in xrange(len(self.img)-1):
            #From column 1 to the single last column
            for j in xrange(len(self.img[0])-1):
                #If img pixel is not zero...
                if self.img[i,j] != 0:
                    """If the value (middle one) is 1, store the labbeled
                    values left of it and the values above it:

                            x x x
                            x 1 o
                            o o o

                    """
```

```
            #Create empty list to store values of surrounding pixels
            around = []
            around.append(foto_labels[i, j-1  ])
            around.append(foto_labels[i-1, j-1])
            around.append(foto_labels[i-1, j])
            around.append(foto_labels[i-1, j+1])


            #If a labelled pixel is connected to the current pixel...
            if (np.sum(around)!=0):
               around.sort()

               around.reverse()
               #Remove all zero's in the labelled list
               around[:] = [cntr for cntr in around if cntr != 0]


               foto_labels[i,j] = (np.array(around)).min()
               #For every label number in the surrounding pixels...
               around = list(set(((np.array(around)).astype(int))))

             #Put the connected numbers in the row with the label number

               all_labels = []
               for k in around:
                  all_labels = list(set(all_labels + connected[k-1] + around))

               for l in all_labels:
                  connected[l-1] = list(set(connected[l-1] + all_labels))

            #If there is no connection to a labelled pixel...
            else:
               #The pixel gets a new label value
               foto_labels[i,j] = label
               #The label value goes up by one
               label+=1
               #An extra row is created at the connection matrix
               (connected).append([])
      return(foto_labels, connected)

   def labelPixelsTwo(self, foto_labels, connected):

      #Make sure the first row and column (0) are not being processed
      i=1
      j=1
      labelnrs = []
      coordinates_i = []
      coordinates_j = []
      #From row 1 to the single last row
      for i in xrange(len(foto_labels)-1):
         #From column 1 to the single last column
```

```
            for j in xrange(len(foto_labels[0])-1):
                if foto_labels[i,j] != 0:
                    label = [label for label in connected if foto_labels[i,j] in label][0]
                    smallest_label = connected.index(label) + 1
                    foto_labels[i,j] = smallest_label+50

                    if smallest_label in labelnrs:
                        coordinates_i[labelnrs.index(smallest_label)].append(i)
                        coordinates_j[labelnrs.index(smallest_label)].append(j)

                    else:
                        coordinates_i.append([])
                        coordinates_j.append([])
                        labelnrs.append(smallest_label)
                        coordinates_i[labelnrs.index(smallest_label)].append(i)
                        coordinates_j[labelnrs.index(smallest_label)].append(j)

        values = []
        for length in xrange(len(coordinates_i)):
            values.append(foto_labels[ (coordinates_i[length][0]) , (coordinates_j[length][0]) ])

        self.img = foto_labels

        return(foto_labels.astype(np.uint8), coordinates_i, coordinates_j,values)


    def makeImageNrs(self, values, coord_i, coord_j):
        counter = 0
        cropping_all = self.img
        kernel = np.ones(self.kernel_d_and_r,np.uint8)

        test_pixels = []
        test_results = []
        test_own = []
        for length in xrange(len(coord_i)):

            cropping = cropping_all[min(coord_i[length])-5:max(coord_i[length])+5 , min(coord_j[length])-
5:max(coord_j[length])+5]

            cropping = cv2.dilate(cropping, kernel, iterations = self.amount_dilate)

            width = len(cropping[0])*20/len(cropping)
            if width %2 != 0:
                width -= 1
            cropping = cv2.resize(cropping, (width, 20))
            empty_column = np.zeros(len(cropping))

            print "width = ",width
        #Making individual images of all numbers
            #Add black lines at the sides of the number, but keep the proportions of the number
            for adding_c in xrange((28-width)/2):
```

```
            cropping = np.insert(cropping, 0, empty_column, axis=1)
            cropping = np.insert(cropping, len(cropping[0]), empty_column, axis=1)

        #Add 4 black lines above and underneath the number
        empty_row = np.zeros(len(cropping[0]))
        for adding_r in xrange(4):
            cropping = np.insert(cropping, 0, empty_row, axis=0)
            cropping = np.insert(cropping, len(cropping), empty_row, axis=0)

        #Make all numbers in the individual image white
        for x in xrange(len(cropping)):
            for y in xrange(len(cropping[0])):
                if cropping[x,y] == values[length]:
                    cropping[x,y] = 255
                else:
                    cropping[x,y] = 0
        print len(cropping)

        #Make compressed database of all numbers in the image
        test_own+=(zip([test_pixels], [test_results]))


        cntr_str = str(counter)
         #Write away an image per number
        path, dirs, files =
os.walk("C:/Users/Carya/Desktop/testenmaar/images/created/new_training_7").next()
        file_count = len(files)
        cntr_str = str(file_count)

        cv2.imwrite('C:/Users/Carya/Desktop/testenmaar/images/created/new_training_7/' +
cntr_str + '.jpg', cropping)
        counter += 1



    return(cropping, test_own)


  def fitNumbers(self, coord_i, coord_j):
    #If "foto" is entered, i = 1 and the foto is a seperate number
    middle_squares_i = []
    middle_squares_j = []
    for length in xrange(len(coord_i)):
        middle_squares_i.append( max(coord_i[length]) - ((max(coord_i[length])-
min(coord_i[length])) / 2))
        middle_squares_j.append( max(coord_j[length]) - ((max(coord_j[length])-
min(coord_j[length])) / 2))

    #      Make regression line with pointcloud of position of numbers
    middle_squares_i, middle_squares_j = zip(*sorted(zip(middle_squares_i, middle_squares_j)))
    print "Middle i = ", middle_squares_i
```

```
print "Middle j = ", middle_squares_j
fit = np.polyfit(middle_squares_j, middle_squares_i,1)

return(fit)
```

Appendix K3 Multi Layer Perceptron code

```
"""
This code is based on the code retrieved from
http://neuralnetworksanddeeplearning.com/index.html, from Michael Nielsen.
An own implementation of the code has been made to accomodate it to my own needs.

################################################################################
Number recognition software (MLP.py)
Created on Tue Sep 22 11:43:03 2015

@Company: Carya Automatisering
@author: Mark Schoneveld

@Study: Mechatronics
@School: The Hague University of Applied Sciences (Delft)
################################################################################


Description
This program consists of the following classes and functions

class ImageLoadAndProcess
   -imExtract
   -imPreprocess

class features
   -featureExtraction
   -featureRecognition

class MLPTrain ()
   -__init__
   -train (main function to call for training the network)
   -forward (feedforward through the MLP)
   -sigmoid (sigmoid calculation)
   -deltaOut (differencebetween target and calculated value)
   -deltaHidden (difference between target(hidden node)and calculated value)
   -update (recalculate all weights and replace the old ones)
   -backProp (calculations of the "should be" weights from output to the weight, hence backwards)
"""
import random
import numpy as np
print "Hello"

class MLPTrain(object):
   """ Input: variableName = filename.classname(nrInputNodes, nrHiddenNodes, nrOutputNodes)
      Input example: network = MLP.MLPTrain(700,4,10)
      Means 700 input nodesin input layer, 4 hidden nodes in output layer
      and 10 output nodes in output layer.
   """
```

```
    def __init__(self, MLP):
        """Length of the network, so number of layers."""
        self.MLP = MLP
        """Create 3D matrix with weights belonging to connected nodes."""
        self.weights = [np.random.randn(y,x) for y,x in zip(MLP[1:], MLP[:-1])]
        """Create 1D array with random values for the bias."""
        self.biases = [np.random.randn(y,1) for y in MLP[1:]]

    def train(self, traindata, testdata, epochs, eta, nr_mini_batch):
        random.shuffle(traindata) #Shuffle the trainingdata every time all epochs are done.
        """When i is smaller than the nr of epochs, a whole itteration (epoch)
        will be performed to update all weights and biases"""
        eta = eta/nr_mini_batch #Make sure the learning rate is equally devided over every image in a
batch.
        for i in xrange(epochs):
            batches = self.makeBatches(traindata, nr_mini_batch) #Create batches
            for batch in batches:
                self.update(batch, eta)
            self.outcome(i, testdata)

    def makeBatches(self, traindata, nr_mini_batch):
        random.shuffle(traindata) #shuffle all traindata
        batches = [traindata[j:j+nr_mini_batch] for j in xrange(0,len(traindata),nr_mini_batch)]
        return (batches)

    def update(self, batch, eta):
        d_weights = [np.zeros((y,x)) for y,x in zip(self.MLP[1:], self.MLP[:-1])]
        d_biases = [np.zeros((y,1)) for y in self.MLP[1:]]
        for x, target in (batch):
            delta_weights, delta_biases = self.backProp(x, target)
            d_weights = [d_w_old - d_w_new for d_w_old, d_w_new in zip(d_weights, delta_weights)]
            d_biases = [d_b_old - d_b_new for d_b_old, d_b_new in zip(d_biases, delta_biases)]
        self.weights = [weights - eta * d_w_old for weights, d_w_old in zip(self.weights, d_weights)]
        self.biases = [biases - eta * d_b_old for biases, d_b_old in zip(self.biases, d_biases)]

    def backProp(self, inputs, target):
        d_weights = [np.zeros((y,x)) for y,x in zip(self.MLP[1:], self.MLP[:-1])]
        d_biases = [np.zeros((y,1)) for y in self.MLP[1:]]
        inputs_copy = inputs
        """Forward movement through network till output is reached."""
        sigm = self.forwardTrain(inputs_copy)
        """Backward calculations"""

        d_out, d_weights, d_biases = self.deltaOut(d_weights, d_biases, target, sigm)

        d_weights, d_biases = self.deltaHidden(d_weights, d_biases, d_out, sigm)


        return (d_weights, d_biases)
```

```
  def outcome(self, i, testdata):
    results = [(np.argmax(self.forwardTest(x)), y) for x,y in testdata]
    grade = sum(int(estimate == y) for estimate, y in results)
    print "results = ", results

    print "Epoch ", i, ":   ", grade ,"out of", len(testdata) ,"correct (", float(grade)/len(testdata)*100,
"%)"

######################## FORWARD ##############################################
  def forwardTrain(self, x):
    """
    Input
    -x(old) (all values of every pixel in the batch)

    Output
    -x(new)
     """
    sigm = [x]

    for weight, bias in zip (self.weights, self.biases): #doorloop elk gewicht en elke bias.
      c = np.dot(weight, x) + bias
      x = self.sigmoid(c) # y(j) = (x(1)(j)*w(2)(j)+.....x(i)(j)*w(i)(j)) + b(j).
      sigm.append(x) #Put value x at the end of the array sigm.
    return (sigm)

  def forwardTest(self, x):
    for weight, bias in zip (self.weights, self.biases): #doorloop elk gewicht en elke bias.
      #c = np.dot(weight, x) + bias
      x = self.sigmoid(np.dot(weight, x) + bias) # y(j) = (x(1)(j)*w(2)(j)+.....x(i)(j)*w(i)(j)) + b(j).
    return (x)

  def sigmoid(self, y):
    """Called in forward"""
    return (1.0 / (1.0+np.exp(-y)))

 ################## Delta ##########################
  def deltaOut(self, d_weights, d_biases, target, sigm):
    d_out = (target-sigm[-1])*sigm[-1]*(1-sigm[-1])
    d_biases[-1] = d_out
    d_weights[-1] = np.dot(d_out, sigm[-2].transpose())
    return (d_out, d_weights, d_biases)

  def deltaHidden(self, d_weights, d_biases, d_out, sigm):
    for i in xrange(2, len(self.MLP)):
      d_hidden = np.dot(self.weights[-i+1].transpose(), d_out) \
        * ((sigm[-i]*(1-sigm[-i])))
      d_biases[-i] = d_hidden

      d_weights[-i] = np.dot(d_hidden, sigm[-i-1].transpose())
    return (d_weights, d_biases)
```