

# Automatic Test Grading Using Image Processing And Machine Learning Techniques



Andries Petrus Smit  
18183085

Report submitted in partial fulfilment of the requirements of the module  
Project (E) 448 for the degree Baccalaureus in Engineering in the  
Department of Electrical and Electronic Engineering at the University of  
Stellenbosch

STUDY LEADER: JA du Preez

DATE: October 2017

## **Acknowledgements**

I would like to acknowledge my skripsie leader, JA du Preez, parents and the engineering class of 2017 for their kind contributing to this thesis.

## Declaration

I, the undersigned, hereby declare that the work contained in this final year project is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

.....

Signature

.....

Date

## Abstract

The aim of this research project is to develop software that can automatically grade tests, written by students. These tests are written on a special template. This template allows the software to extract the students intended answers, after it has been scanned into a digital form. The standard method used in these Optical Mark Recognition (OMR) software, is to allow a learner to specify answers by colouring in bubble grids. To correct a mistake, the previous answer bubbles must first be erased and new ones coloured in. This takes time and increases the probability that a learner might colour in bubbles incorrectly.

This research project tries to solve this problem by implementing additional software that eases the use of such a template. Using computer vision and two machine learning techniques, namely Artificial Neural Networks (ANN) and Probabilistic Graphical Models (PGM), a student can now answer questions using characters and bubbles. In the case of a student number the student does not need to colour in the bubbles at all. The software implemented in this project allows for a decreased time in filling in these templates and thus decreases the number of mistakes made. New methods like these thus allows OMR templates to more easily be incorporated into the traditional educational systems.

## Uittreksel

Die doel van hierdie navorsings projek is om sagteware te ontwikkel wat automaties toetse, wat deur studente geskryf is, na te kan sien. Hierdie toetse word elkeen op 'n spesiale templaar geskryf. Die templaar laat die sagteware toe om die student se antwoord te vind nadat dit digitaal gekopieer is. Die standaard metode wat gebruik word in hierdie Optiesemerk-leser sagteware is om van inkleur rooster borrels gebruik te maak. Om 'n fout wat die student gemaak het regtemaak, moet die ou borrels eers uitgevee word and nuwe borrels ingekleur word. Hierdie proses vat baie tyd en vermeerder die kans dat 'n student die borrels verkeerd in kan vul. Die navorsingsprojek poog om hierdie probleem op te los deur addisionele sagteware te implementeer wat die gebruik van so 'n templaar vergemaklik. Met behulp van rekenaarvisie en twee masjienleertegnieke, naamlik Kunsmatige Neurale Netwerke en Probabilistiese Grafiese Modelle (PGM), kan 'n student nou vrae beantwoord deur karakters en borrels te gebruik. In die geval van 'n studentenommer hoef die student glad nie die borrels eers in te kleur nie. Die sagteware wat in hierdie projek gimplementeer word, veroorsaak 'n verminderde tyd in antwoorde inskryf en verminder dus die aantal foute wat gemaak word. Nuwe metodes soos hierdie kan dus help dat meer OMR sagteware in die tradisionele onderwysstelsels ingebring kan word.

# Contents

<b>Abstract</b>	<b>iii</b>
<b>Uittreksel</b>	<b>iv</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>x</b>
<b>Nomenclature</b>	<b>xii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem background . . . . .	1
1.2 Problem statement . . . . .	2
1.3 Project scope and assumptions . . . . .	2
1.4 Project objectives . . . . .	3
1.5 Research methodology . . . . .	4
1.6 Graphical overview of system . . . . .	5
<b>2 Literature Study</b>	<b>6</b>
2.1 Existing OMR techniques . . . . .	6
2.1.1 Standard OMR systems . . . . .	6
2.1.2 Finding the template . . . . .	8
2.1.3 Processing a bubble . . . . .	8
2.2 Optical character recognition . . . . .	9
2.2.1 Probabilistic approach . . . . .	9
2.3 Conclusion: System requirements . . . . .	10

<b>3</b>	<b>Image Processing</b>	<b>11</b>
3.1	Orientation Detection . . . . .	11
3.1.1	Initial filtering and orientation detection . . . . .	12
3.1.2	Radon Transform . . . . .	13
3.1.3	Finding the template . . . . .	15
3.2	Bubble detection and processing . . . . .	16
3.3	Data processing and grading . . . . .	17
3.4	Conclusion . . . . .	18
<b>4</b>	<b>Machine learning approach</b>	<b>19</b>
4.1	Character recognition using a neural network . . . . .	19
4.1.1	Preprocessing and creating digit images . . . . .	20
4.1.2	Classification of digits . . . . .	23
4.1.2.1	The Neural Network Basics . . . . .	24
4.1.2.2	The Artificial Neuron . . . . .	24
4.1.2.3	Generating an output from the network . . . . .	25
4.1.2.4	Deep Convolutional Neural Network(DCNN) . . . . .	25
4.1.2.5	Training of the neural network . . . . .	26
4.2	Probabilistic Graphical Models . . . . .	26
4.2.1	Overview of the system . . . . .	27
4.2.2	Estimating the intended answer . . . . .	27
4.2.3	Estimating the student number . . . . .	30
4.2.4	Training of a Probabilistic Graphical Model . . . . .	31
4.3	Conclusion . . . . .	31
<b>5</b>	<b>Analysis of results</b>	<b>33</b>
5.1	Results of 25 test cases . . . . .	34
5.1.1	Basic system . . . . .	34
5.1.1.1	Marking statistics . . . . .	34
5.1.1.2	Clashlist . . . . .	35
5.1.1.3	Incorrect automatic graded results . . . . .	35
5.1.2	Complete system . . . . .	36
5.1.2.1	Marking statistics . . . . .	36
5.1.2.2	Clashlist . . . . .	36

5.1.2.3	Incorrect automatic graded results . . . . .	36
5.1.3	Analysis or results . . . . .	37
5.2	Grading of tutorial tests . . . . .	38
5.2.1	Marking statistics . . . . .	38
5.2.2	Clashlist . . . . .	38
5.2.3	Incorrect automatic graded results . . . . .	38
5.2.4	Analysis or results . . . . .	38
<b>6</b>	<b>Summary and conclusions</b>	<b>41</b>
6.1	Project summary . . . . .	41
6.2	How this final year project benefits society . . . . .	41
6.3	What the student learned . . . . .	41
6.4	Future improvements . . . . .	42
6.5	Conclusion: Summary and conclusions . . . . .	42
	<b>References</b>	<b>43</b>
<b>A</b>	<b>Project plan</b>	<b>44</b>
<b>B</b>	<b>Outcome compliance</b>	<b>46</b>
<b>C</b>	<b>Overview of system</b>	<b>47</b>
C.1	System as a whole . . . . .	47
C.2	Deriving the intended student entry . . . . .	48
C.2.1	Student answer . . . . .	48
C.2.2	Student number . . . . .	49
C.3	Deriving the estimated digit . . . . .	51
<b>D</b>	<b>Implementation/Algorithms</b>	<b>53</b>
D.1	Deep Convolutional Neural Network . . . . .	53
<b>E</b>	<b>Validation and results</b>	<b>54</b>



# List of Figures

1.1	Automatic test grading template layout. . . . .	3
1.2	Graphical overview of the system as a whole. . . . .	5
2.1	Standard OMR template with refernce blocks on the left, from <a href="#">VijayaForm (2017)</a> . . . . .	7
2.2	Contours found around corrected answer. . . . .	9
3.1	Four markers found from applying Radon transforms. . . . .	12
3.2	Reduced contours in image. . . . .	14
3.3	Radon transform applied on a 2 dimensional space, from <a href="#">Edoras (2017)</a> .	14
3.4	Result in rotation after applying radon transform. . . . .	15
3.5	Detection of contours in image and estimation of bubble locations. . . . .	16
4.1	Image showing found contours for boxes used for character recognition. .	20
4.2	The box contour found is normalized to form a rectangular shape. . . . .	21
4.3	Box after black lines gets filtered out, found using a Radon transform. . .	21
4.4	Custom segmentation algorithm used to find the main cluster in the remaining image. . . . .	22
4.5	Area block drawn around segment most probable to belong to the digit. .	22
4.6	Image after final translation and normalization is applied. . . . .	22
4.7	Example image used as input to the neural network, from <a href="#">Tensorflow (2017)</a>	23
4.8	Basic structure of a neural network, from <a href="#">Karpathy (2017)</a> . . . . .	24
4.9	Graphical setup for determining the intended digit written by a student.	28
4.10	Column with the evidence that gets considered for the calculation of an intended digit. . . . .	28
4.11	Graphical setup of determining student answer. . . . .	30

## LIST OF FIGURES

---

5.1	Image showing answer with crossed out answers that the system misinterpreted. . . . .	35
5.2	Student filled in answer with only character information. . . . .	36
5.3	Crossed out character that confused the grading system. . . . .	37
5.4	Incorrectly identified answer as 95. . . . .	40
A.1	Project plan for the final year project. . . . .	45
C.1	System overview. . . . .	47
C.2	Graphical setup of determining student answer. . . . .	49
C.3	Graphical setup of determining student number. . . . .	51
C.4	Graphical setup of determining intended digit. . . . .	52

# List of Tables

5.1	Table describing number of clashes in the different catagories. . . . .	<a href="#">39</a>
-----	---	--------------------

# Nomenclature

## Acronyms

<i>ANN</i>	Artificial neural network
<i>DCNN</i>	Deep convolutional neural network
<i>DCNN</i>	Deep convolutional neural network
<i>OCR</i>	Optical character recognition
<i>OMR</i>	Optical mark recognition

<i>PGM</i>	Probabilistic graphical model
------------	-------------------------------

## Symbols

$\sigma(z)$	Normalization function in a neural network.
$b$	Bias variable added to allow a neuron to have an offset in its output.
$c$	Number of inputs to a neuron.
$f(x, y)$	Two dimensional function that a Radon transform is applied over.
$G(r, \theta)$	Radon transform defined over $r$ and $\theta$
$n$	Number of bubbles of template sheet.

$w_i$	Weight value at index i.
$x_i$	Input value at index i.
$z$	Weighted some of a neuron's inputs and internal variables.

# Chapter 1

## Introduction

As modern technology and machine learning techniques advances, it is important for the educational sector to also continuously advance their learning environment. This allows for an ever improving learning experience in and outside the classroom.

### 1.1 Problem background

*In the recent years the Applied Mathematics Department of Stellenbosch Engineering, started observing a decrease of accuracy in grading of tutorial tests, done by a teaching assistants and demies. Students complain on a regular basis about correct answers being marked wrong or even that their answers were totally ignored. Further the assistants also takes a long time to grade these tests. To address this problem the Applied Mathematics department proposes to automate the process of grading these tutorial tests.*

The head of the department wants a system that can analyse and grade tests written on a specific template. These answer sheets are handed out to the students to fill in their respective answers. The answer sheets are then scanned-in to create a digital copy. The system is tasked with automatically grading all these digital copies and transferring the graded results to a database.

The department has tried to use a basic version of this type of system in the past. Such a system only really becomes useful to the department if it can grade decimal values instead of only being able to grade multiple choice answers. Thus a template needs to be designed that allows students to answer with decimal valued answers. Another factor to consider is filling in those values must still be relatively easy. Thus students need to

have the freedom of quickly crossing out an answer instead of erasing it each time. In this research project the department sends weekly scanned in answer sheets, which needs to be graded and the results returned to them. The feedback from these weekly tutorial tests then acts as validation tests for the system. These tests are done in parallel with the development and expansion of the test grading software. For these reasons an agile development methodology is used to develop this software.

## 1.2 Problem statement

Given the problem background and stakeholders discussed in the previous section the problem to be solved can be formulated as

**Develop** and **implement** a *automatic test grading system* that will *increase marking accuracy* and *decrease marking time* on the *grading* of Applied Mathematics tutorial tests, written by students.

## 1.3 Project scope and assumptions

Initial discussions with the department revealed that a specific template can be used. This template allows the image processing software to more accurately determine what the student's intended answer is. The template consists of bubbles that can be filled in as well as blocks for handwritten digits, as seen in Figure 1.1. The focus of this project lies on processing the scanned in answer sheet written on the specific template. To use the template the student must fill in his/her student number and question answers in the designated character blocks. They are also required to fill in the bubble underneath each digit, corresponding to that specific digit. Additionally a bubble next to each question is provided if a negative sign is required. (Gebriuk dalk template wat nie scrubble of het nie?)

Any additional assumptions are specified at the appropriate times throughout this project. In the next section the project objectives is laid out.

[illegible]

Figure 1.1: Automatic test grading template layout.

## 1.4 Project objectives

The problem as stated in section 1.2, is addressed by pursuing the following objectives:

1. Do a *literature study* on the topics of *Image Processing*, *Computer Vision* and *Character Recognition*. Further a *literature study* on *Neural Networks* and *Probabilistic Graphical models* is also done.
2. Develop a *software application* to allow a *user* to grade a large number (approximately 1000) scanned in student tests automatically.
3. The software should providing precise and useful feedback. To do this every graded result will also include feedback on what questions the student answered incorrectly.
4. Upgrade the software to allow students to cross out answers over having to erase them.
5. Do a weekly *validation experiment* with the software, by grading tutorial test for the department. The results are then used as the student's grade for that test.



6. Use an *agile development methodology* to improving the software in parallel with the grading of weekly tests.
7. Add additional software using machine learning to improve the accuracy of the system in grading these test.
8. Add additional software that allows students to specify their student number only using characters, which makes the template easier to use.

The objectives are covered in different chapters in the report. Note that each objective (1–8) build on the objective(s) previously listed.

## 1.5 Research methodology

To complete the objectives listed in Section 1.4, a agile development methodology is used. The methodology consists of six different phases:

1. Identify a new feature or update that needs to be implemented into the software package.
2. Do a study on the existing methods to implement this new software.
3. Implement and integrate the new software with the current knowledge of the solution.
4. Test the software and observe if it is working as planned.
5. If the software is not working as planned, revisit steps 2-4 until the new software is working.
6. Use version control, in this case Git, to save the latest version of the software.

Next the structure and graphical overview of the software is presented.

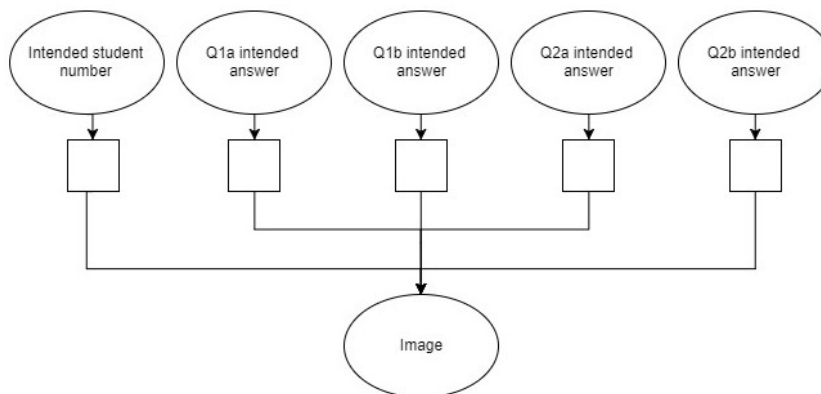


Figure 1.2: Graphical overview of the system as a whole.

## 1.6 Graphical overview of system

When thinking about the template, from a philosophical sense, it can be represented using 6 information nodes. These nodes are illustrated in Figure 1.2. The unnamed blocks indicate that information processing happens in these steps. The student has certain information he/she wants to portray on the paper. This includes the 4 answers and student number he/she wants to write down. Thus those 5 nodes gives rise to the image, representing the last node. These nodes have to be represented in a probabilistic way. The reason for this is, because the process of writing answers down and scanning the test sheet in is going to produce a different image every time a test is written, even though the same answers are intended. Thus the system is fundamentally tasked with inferring the probability of each answer and the student's number given that image as evidence. This is done by processing the image evidence to produce an estimate of the intended answers. These processes are described in later sections. In the next section a literature study on current systems is discussed.

For a more detailed mathematical overview of the system, refer to Appendix C.

# Chapter 2

## Literature Study

In the previous chapter the problem statement and objectives of this project was laid out. Further a brief system overview is also provided. This chapter will provide information about already existing Optical Mark Recognition(OMR) systems and the techniques they use in doing image processing on images. Research on additional machine learning approaches used in character recognition and probabilistic modelling is also described.

### 2.1 Existing OMR techniques

A OMR system is a piece of software that is used to extracted hand written information from a filled in form. Each system normally has a specific template that it can extract information from. An example template is shown in Figure 2.1. These systems are generally used when fast and accurate grading of tests are needed. The biggest drawback of these systems are that information can only be portrayed in a very limited manner, due to the bubbles. On OMR templates there are a grid of bubbles that allows a user to cohose between different options to answer. OMR systems are thus excellent for the grading multiple choice type questions.

#### 2.1.1 Standard OMR systems

As can be seen in Figure 2.1, there is normally specific reference blocks on a OMR template. These blocks are included to allow the computer vision and image processing

Figure 2.1: Standard OMR template with reference blocks on the left, from [VijayaForm \(2017\)](#)

algorithms find the orientation of the image more easily. Other templates include lines that can be used to locate the template.

In an OMR system there are normally two phases in the grading of an test, as stated in [Ivetic & Dragan \(2003\)](#). The first step is to determine the grid within where the answers are located in the image. In this process the system finds the orientation of the template in the image and thus can approximate the location of the use of colour in bubbles. Normally some preprocessing on a blank template is done beforehand too aid in locating the bubbles. Once the bubbles are found their estimated locations gets stored. The second step is then to estimate the value each bubble and use these values as the estimated answers. These steps is described in more detail next.

### 2.1.2 Finding the template

The first process performed by OMR software is to locate a template grid inside the test image. This step is necessary for the software to know which bubble corresponds with each answer. One method of locating the template is looking for lines in the template. This border normally contains long lines that can be extracted using a Hough transform, as stated in [Patel & Prajapati \(2003\)](#). A Hough transform is used to locate instances of an imperfect object within a certain shape range. A specific form of a Hough transform can be implemented to detect lines. This form is called a Radon transform, as described in [MathWorks \(2017\)](#). A Radon transform provides a way of representing a image as a summation of different line integrations, as is discussed in [Section 3.1.2](#).

Once at least two line references on a page has been found the template orientation can be determined. Those two lines are then used to find two reference points on a page. These points allows the system to estimate the locations of every bubble on the template sheet. In the next section a method to process these bubble is described.

### 2.1.3 Processing a bubble

Now that an estimated location for each bubble is known, the next step for an OMR system is to process these bubbles. As stated in [Patel & Prajapati \(2003\)](#), a basic image processing method to classify bubbles is to simple count up the number of coloured in pixels. If this value is above a threshold value the bubble is classified as filled in, otherwise not. To detect if a bubble is crossed out an additional algorithm is needed.

To determine if an answer in a bubble is truly coloured in and not just crossed out, contour detection is used. This means that the contour around the bubble needs to be detected and used in analysis. In python(or C) this can be implemented using the freely available OpenCV library, as described in [Rosebrock \(2016\)](#). OpenCV is an image processing library that has highly optimized techniques to find contours in a given image. An example of this can be seen in [Figure 2.2](#). Once the contour information is known the bubbles can be assessed by the pixels inside it, as well as its shape. Thus by looking at the shape of a contour it can be determined if a bubble is crossed out or not. (Verander die image na een wat 'n contour om het)



that consist of a graph that specifies the probabilistic relationship between a number of random variables. These type of models works well in some aspects of the medical field. If a patient has certain symptoms a PGM can be used to predict what the underlining illness behind the decease is. In this project a PGM can be used to estimate the underling student entries given the evidence presented. The library used in [Ankan & Panda \(2015\)](#) is called pgmPy. This library allows for a PGM to be constructed in python.

## 2.3 Conclusion: System requirements

In conclusion it is seen that a combination of image processing and machine learning techniques is needed to successfully grade a student test paper. A good method in locating a template is using a Radon transform to find reference lines in an image. Once this is done the bubbles can be estimate in the image. Too classified digits it is found that a DCNN can be implemented in the TensorFlow library space. Once all these evidence are acquired a Probabilistic Graphical Model was found to be a perfected method in predicting the estimated entry answer of the student. In Chapter 3 a more detailed overview on the image processing techniques used in this project is given.

# Chapter 3

## Image Processing

The previous chapter focused on existing methods of grading tests automatically. It was found that most systems only use image processing, without a machine learning component, to grade these test. In this chapter the core techniques behind processing these answer sheets, using image processing, is described. By using only these image processing techniques a reasonably accurate system can already be constructed.

For further improvements in accuracy we and implement two machine learning approaches. These approaches are discussed in Chapter [4](#).

### 3.1 Orientation Detection

As mentioned in Section [2.1.1](#), there are two main steps in OMR grading. The first challenge with grading a scanned in answer sheet, is finding the orientation of the template in the image. To do this 2 or more references points must be found on the page. These reference points then allows for the calculation of the template's rotation, offset and size inside the image. In Chapter [2](#) it was found that the traditional way to find this reference points was to include them on the page, in forms of black blocks or lines. Including black blocks is an effective method of finding the template, but might look a bit less attractive, due to the black blocks on the page.

For this project the markers or reference points that the software uses are already present on the template paper. These are the two longest horizontal lines as well as the two vertical lines on the comment box, as can be seen in Figure [3.1](#). Together these lines have enough information to determine 4 reference points, shown in red in the figure. The



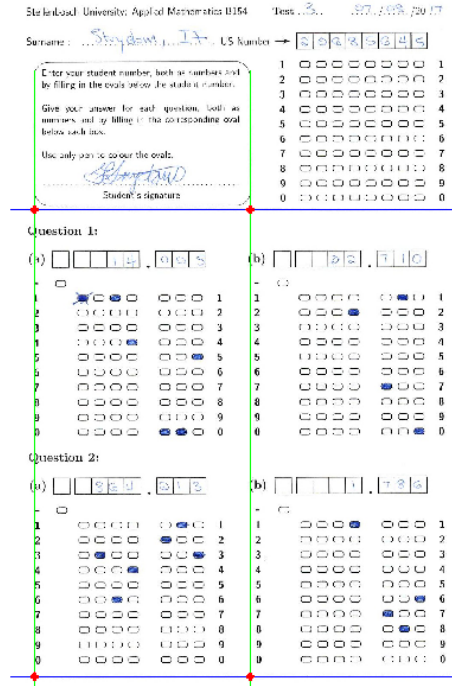


Figure 3.1: Four markers found from applying Radon transforms.

reason these lines are chosen as references, are due to the fact that a Radon transform can easily be applied to find them, as seen in Section 3.1.2. Thus the software can successfully find the template even though one of the 4 lines are identified incorrectly. Using the reference points the rotation, offset and size of the template is calculated. But before the orientation of the image can be determined it is a good idea to quickly check if the image might be upside down. This is done to make it easier to find the orientation afterwards. To do this some initial image filtering is firstly required and is discussed in the next section.

### 3.1.1 Initial filtering and orientation detection

To check if an image is upside down the software first needs to find relevant contours on the page. The contours are then filtered out if it does not loosely match the characteristics of a bubble or character block. This process can be described in 5 steps:

1. Threshold the image by making all the pixel values either white (lower than the mean) or black (higher than the mean).
2. Do contour analysis on the image to find all the contours, using the python library OpenCV.
3. Filter through the contour array to filter out all contours that are not approximately the size and aspect ratios desired.
4. Save these contours for later use.
5. Determine if more contours lie above the middle of the image. This is true if the image is the right way around. Rotate the image by 180° otherwise.

It is important to note that there are still unwanted contours in the list, but for now this reduced list is sufficient. Once the list is found the software counts the number of contour centrepoints below and above the image center. Figure 3.2 shows the resulting contours found in the image. As can be seen in the figure more bubbles should be below the horizontal center line, for the image to be the right side up. The next step will now be to determine the coordinates of the answers the student wrote down. To do this the template is first located in the image. This process is described in Section 3.1.2.



#### 3.1.2 Radon Transform



The Radon transform is an integral transform that can be represented by a series of line integrals over a function  $f(x, y)$ . These transforms are commonly used in CT scans. Mathematically this transform is defined as

$$G(r, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \delta(x \cos(\theta) + y \sin(\theta) - r) dx dy.$$



Where  $r$  represents the perpendicular offset of the line and  $\theta$  the angle of the line.  $x$  and  $y$  represents variables within a 2D space within which the function  $f(x, y)$  is defined. A visual interpretation of this transform can be seen in Figure 3.3.



**Question 1:**

(a)  , 

(b)  , 

**Question 2:**

(a)  , 

(b)  , 

### 3.1 Orientation Detection

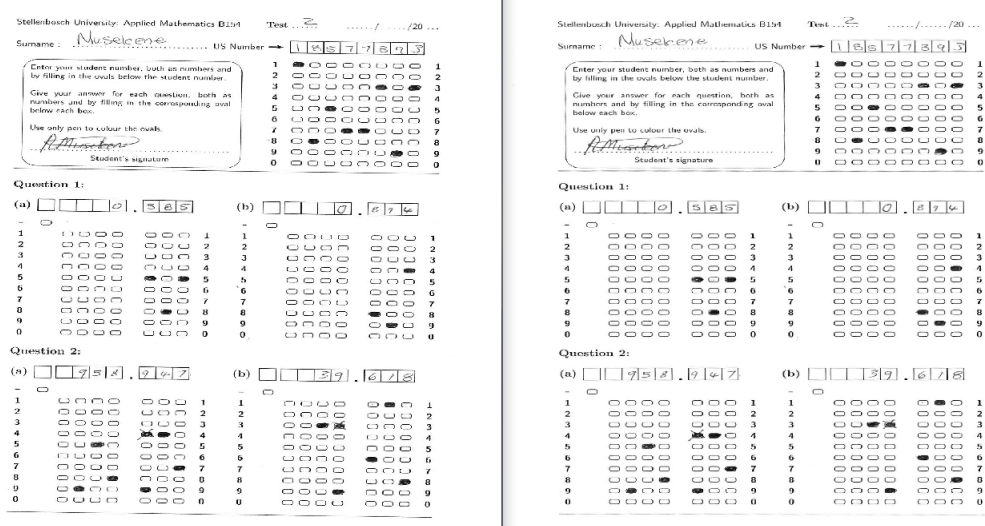


Figure 3.4: Result in rotation after applying radon transform.

#### 3.1.3 Finding the template

In this project the image's Radon transform is calculated for specific intervals of the gradient. Each gradient interval will thus generate a 1 dimensional array of values corresponding with the pixel intensities along the lines that are being summed, as seen in Figure 3.3. This method is used determine where the 2 horizontal and 2 vertical lines are located, as mentioned in Section 3.1.

To find the first two horizontal lines the Radon transform's values are calculated between the angles  $85^\circ$  and  $95^\circ$ . It is amused that the image will not be rotated by more that  $5^\circ$  in either direction. Black lines will cause a spike to appear on the Radon transform for the angle where it is summing parallel with that black line. Thus by finding the transform angle value that present this maximum value, the rotation of the template can be found. The two maximum values that is recorded at this angle is then recorded as the relative locations of the two horizontal lines. After the correct angle is found the two vertical lines can be found by applying a Radon transform at a angle  $90^\circ$  clockwise from the previously calculated angle. The two peak values at that angle then provides the relative locations of the two vertical lines. The four reference points can now be calculated, as seen in Figure 3.1 can be calculated. Once the reference points are found the image is rescaled and orientated to the original template size for further processing. In Figure 3.4 the corrected image is shown.

## 3.2 Bubble detection and processing

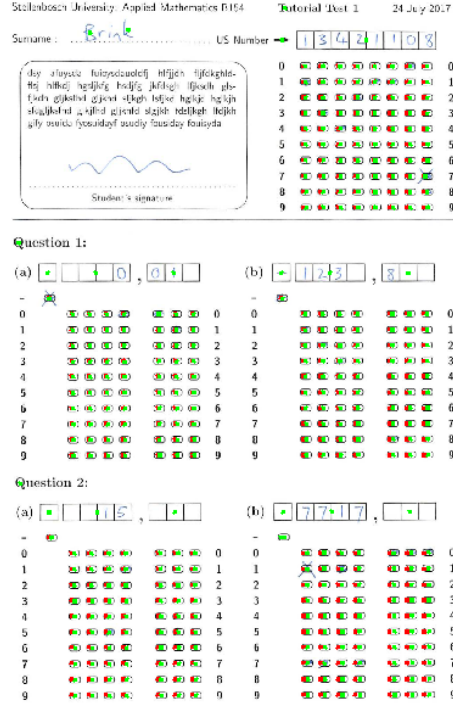


Figure 3.5: Detection of contours in image and estimation of bubble locations.

Once the template is found the bubble values and digit blocks can be determined, using reference location. These reference locations were calculated in preprocessing done on an empty template. Figure 3.5 illustrates the final estimation of all the bubbles in the template. The estimated bubble centres are coloured red, while the green points represent the centres of all the remaining contours.

Now that a list of possible bubble contours are found with the estimate bubble centres, one contour needs to be assigned to each bubble. This is discussed in the next section.

## 3.2 Bubble detection and processing

To find the location of each bubble in the image, the system simply takes the contour closest to the estimated bubble location. This can be done in an efficient manner by sorting the contours by their locations. Searching through the contours now becomes linear and of complexity  $O(n)$ , where  $n$  is the number of bubbles. This means that the processing time to find these bubbles is linearly related to the number of bubbles in the

template image.

Next the data in each contour needs to be processed and stored. The first type of evidence is calculating the average pixel intensity inside the contours. If this value is high the bubble is most likely coloured in or crossed out. The advantage of using the closest contour as the bubble's estimated location, over conventional methods now becomes apparent. In conventional methods an estimated area is calculated where the bubble is most likely to be located. This becomes problematic if the system needs to know if the bubble has been crossed out, as only data about pixel intensities are present.

Using contour analysis, information about the shape of the bubble entry is also provided. Thus by drawing the smallest possible block around the contour, that still covers every value inside the contour, an area can be calculated. This area becomes large when an answer is crossed out, due to the lines stretching outside the initial bubble. By inspecting the area value, the system can successfully determine if the bubble is filled in and crossed out.

### 3.3 Data processing and grading

The previous section now allows each bubble to be classified into 3 categories namely, empty, completely filled in and crossed out. An additional category of partially filled in will also be introduced, as it aids grading of tests where students write lightly. An algorithm to determine what bubble was chosen can now be described as follows:

1. Start with column 0.
2. Count the number of completely filled in answers in this column. Store the position of that entry for later use.
3. If there are no completely filled in answers, count the amount of partially filled in answers and override the previous values.
4. If the previous result is 0, set the output value for that column to 0.
5. If step 2 or 3 presents a value greater than 1, save the answer sheet to a clashlist to be evaluated manually once the automatic grading of the test are completed.
6. Repeat steps 2 to 5 for each column in the bubble grid.

This algorithm thus checks if there are more than one entry in any of the columns. If this is true the results are sent to a clashlist to be marked manually. If one bubble was found to be coloured in the value gets set to the index of that bubble. Finally if no bubbles were coloured in the result for that column gets set to 0.

## 3.4 Conclusion

This chapter provided an overview of a basic automatic test grading system using image processing and computer vision. The system can achieve acceptable results using only these techniques.

The following chapter will focus on applying additional machine learning techniques to further improve the accuracy of grading these tests.

# Chapter 4

## Machine learning approach

The previous chapter briefly described the basic workings of the optical mark recognition (OMR) system inside the automatic test grader. There is still one critical piece of evidence that has not yet been observed in this basic system. This information is the characters that the student writes in the designated boxes.

This next chapter will provide two machine learning approaches to significantly improve the accuracy in identifying digits over the previous standalone basic system, discussed in Chapter 3. An approach to locate and classify hand written characters, provided by the students, using a deep convolutional neural network (DCNN), is described. Next a more accurate method in estimating the student answers and student number, using probabilistic graphical models (PGM), is also discussed.

### 4.1 Character recognition using a neural network

This section focusses on processing the characters that the students write down in the designated boxes, as shown in Figure 4.1. To process these digits a machine learning approach, called a neural network, is implemented. This neural network can take a input of a 28 by 28 dimensional array of floating point numbers. These numbers represent a 28 by 28 greyscaled image that includes the digit being classified. The neural network is then tasked with calculating the probability of each digit being present in that image. Before each digit can be classified using a neural network, the individual 28 by 28 greyscaled image must first be found for each digit inside the test sheet. To do this image processing is required as further described in the next section.



## 4.1 Character recognition using a neural network

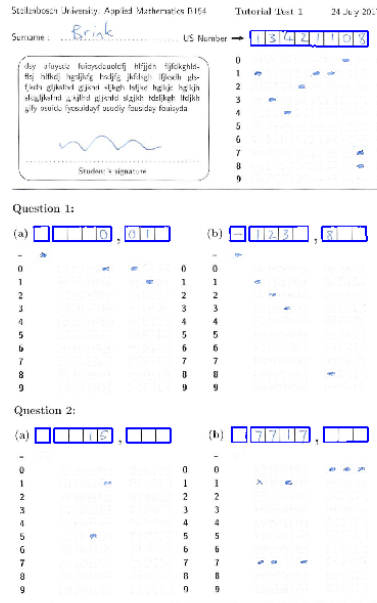


Figure 4.1: Image showing found contours for boxes used for character recognition.

### 4.1.1 Preprocessing and creating digit images

To generate a 28 by 28 pixel image for each digit the system must first find the locations of each digit. Once this is done the digits is processed and a corresponding 28 by 28 image, best representing that digit, can be created. This is done in 6 steps as seen below:

1. Find the contour closest to the expected location of the block, calculated in Section 3.1.3. This is illustrated in Figure 4.1. It can be seen that the bubbles are already processed.
2. Transform the image to become fully rectangular using OpenCv's *four\_point\_transform* method. This method applies a four point perspective transform on the image to reshape it into a rectangular form. An example of the final product can be seen in Figure 4.2.
3. Perform a Radon transform on the image, at an angle of  $0^\circ$  and  $90^\circ$  to find the dark lines in the blocks. These lines are then removed from the image, as seen in Figure 4.3.

## 4.1 Character recognition using a neural network

---

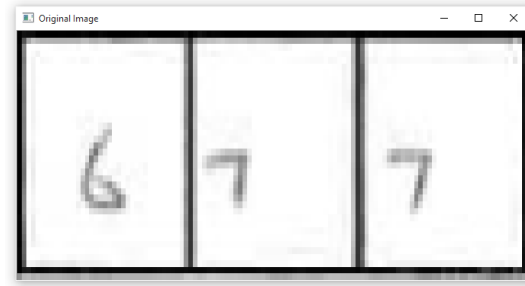


Figure 4.2: The box contour found is normalized to form a rectangular shape.

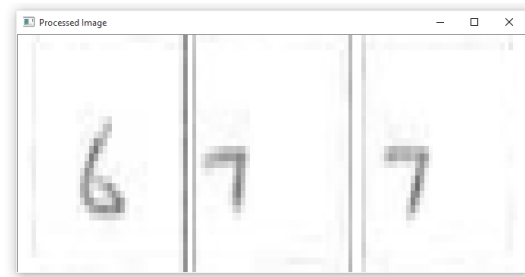


Figure 4.3: Box after black lines gets filtered out, found using a Radon transform.

4. Use the indexes received from the Radon transform, of where the box lines were, to segment the image into the different boxes.
5. Using a custom segmentation algorithm to find the pixels most likely to belong to the digit. This algorithm is developed and implemented using breath first search technique to cluster the image into different segments. The algorithm works by first searching the image for pixels higher than a threshold value. This value specifies if a pixel is background or belongs to an object. Then all the pixels higher than the threshold value get assigned to a segment. A segment is thus classified as a local region that does not connect to any other segment through pixels higher than a threshold. The segment that most likely represents the digit is then extracted, as seen in Figure 4.4.
6. The area that the segment occupies is then calculated, as seen in Figure 4.5
7. Next the image is centred and normalized using the image area as reference. This is seen in Figure 4.6

## 4.1 Character recognition using a neural network

---

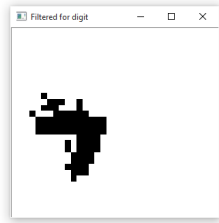


Figure 4.4: Custom segmentation algorithm used to find the main cluster in the remaining image.

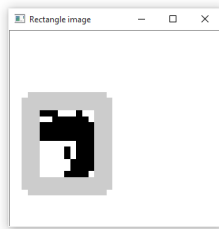


Figure 4.5: Area block drawn around segment most probable to belong to the digit.

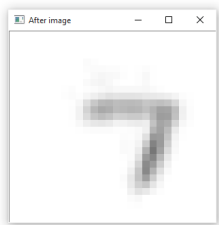


Figure 4.6: Image after final translation and normalization is applied.

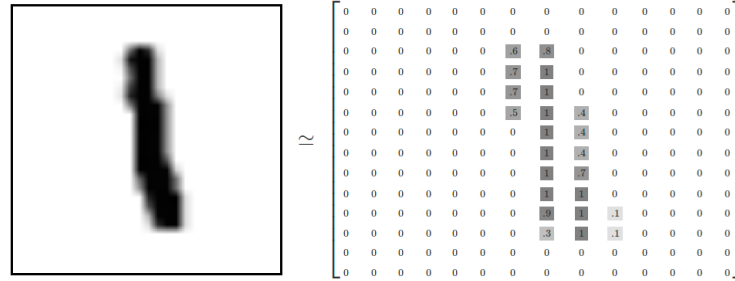


Figure 4.7: Example image used as input to the neural network, from [Tensorflow \(2017\)](#)

8. The image is then reshaped into a 28 by 28 greyscaled image to be processed by the neural network.

For this project a 28 by 28 greyscaled image is used as input. Thus if each pixel represents one value, ranging from 0.0 to 1.0, there is total of 784 input values. Each digit image is thus represented by one of these 28 by 28 greyscaled images. These 784 numbers is now used as input to an neural network to predict the digit. An overview of this neural network will be given next.

### 4.1.2 Classification of digits

A neural network is a powerful machine learning tool for approximating complex functions. The basic architecture of a neural network can be seen in Figure 4.8. These networks are simplified approximations of how neurons in the brain works. Each neuron in the network acts as a small processing unit that can take a input and produces a output. The power of a neural network lies in that fact that these neurons have internal values that can be tweaked depending on the characteristics the user wants the network to approximate. This thus allows complex functions to be trained onto such networks.

For this project, a neural network is trained to estimate the probability of which of the 10 digits, from 0 to 9, are most likely present in the input image. The neural network will take a 2 dimensional array, representing a greyscaled image, as input to the network. Figure 4.7 illustrates an example input of a neural network using and 14 by 14 example image. For this project a 28 by 28 image is used. In the next section the basics of these neural networks is discussed.

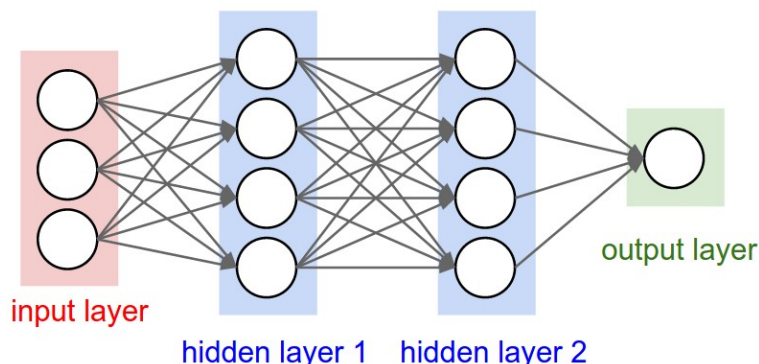


Figure 4.8: Basic structure of a neural network, from [Karpathy \(2017\)](#)

### 4.1.2.1 The Neural Network Basics

The neuron unit is the small processing unit and by including many of these neurons a neural network can be built. Neural networks consist of input, hidden and output layers, as described in [Nielsen \(2015\)](#). This network works by first assigning values to the input layer of the network. In this project those values will be the 784 values received from the digit image. The network then send signals through the network and generates an output. A breakdown of a neural network will be given next.

### 4.1.2.2 The Artificial Neuron

Each neuron in the network takes in a list of input values. This input values are the output values of the neurons in the layer before this neuron, seen in Figure 4.8. This values are that multiplied by internal weights of the neuron and a final weight, named the bias, is then added. These internal weights are thus fixed to a specific value, depending on what function that neuron tries to approximate. A bias variable is also added to expand the range of functions a neuron can approximate. The weighted sum of these inputs are then added with the bias variable to give,

$$z = \sum_{i=0}^c x_i * w_i + b. \quad (4.1)$$

Where  $c$  is the number of inputs.  $x_i$  and  $w_i$  respectively are the input and weight values at index  $i$ .  $b$  is a term that enables and offset in the output( $z$ ) of the neuron. The

summed value then gets normalize using an sigmoid function, seen seen

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \quad (4.2)$$

This artificial neuron thus takes in a weighted input with a bias and produces a normalized output  $\sigma(z)$ . By adjusting the weights and bias variable, each neuron can learn to exhibit certain characteristics. This process thus allows different functions to be approximated by adjusting these variables. If a network of these neurons is used together, as shown in Figure 4.8, complex functions can be trained onto the network. In this project these weights and biases needs to be set to specific values, to allow the network to accurately categorize digits from an image. To do this these weights and biases will be trained form training data, as seen in Section 4.1.2.5.

### 4.1.2.3 Generating an output from the network

After the 748 input values have been assigned to the network inputs, each of the network's layers can be calculated consecutively one after another. This is done using the previously mentioned Equations 4.1 and 4.2 on each layer consecutively of the network starting at the first hidden layer. Once the first layers's outputs are calculated the next layer can be calculated. This process is repeated until all the layers is calculated. After this step the final values can be read of of the output neuron that represents the result. In this project then output neurons is used to represent the likelihood of each of 10 digits, given the input data. To turn the observed on the output neurons into probabilities, the values gets normalized using

$$p(i) = \frac{\sigma(z_i)}{\sum_{k=0}^{10} \sigma(z_k)}. \quad (4.3)$$

Where  $p(i)$  is the probability of digit  $i$  being the character in the image. The value  $\sigma(z_i)$  is the output of the output neuron at index  $i$ .

### 4.1.2.4 Deep Convolutional Neural Network(DCNN)

The previous section gave an overview of the basic a neural network implementation. In recent years much more powerful neural networks such as Deep Convolutional Neural Network(DCNN) has also been introduced. A DCNN uses the basic principles described

## 4.1 Character recognition using a neural network

---

above, but has extra optimized features that allows an neural network with many layers to be constructed. The exact mathematics behind these Deep Neural Networks are beyond the scope of this project report. This neural network, implemented in TensorFlow, allows an neural network to achieve an high accuracy on clasifying digits take from these test. These results are described in 5.

### 4.1.2.5 Training of the neural network

To train a DCNN neural network the MNIST dataset is used. This is a database that has a labelled training set of 60,000 images, and a labelled test set of 10,000 images. For each image in the training set there is an accompanied label that specifies what digit it is. The neural network is then trained to model this training set. This is done by adjust the networks internal weights so that that the network's output better represent the labelled training data, given the input images. The basic idea behind the training method used in a neural network is described in the following steps.

1. Calculate the network's output for each of the training images used in this training round.
2. Get the error function of the network, using a formula that compares the true labels of the training image, with the estimated labels generated by the network.
3. Calculate the value with which each weight should be changed to reduce the error function slightly. One method of doing this is using gradient decent with back propagation.
4. Repeat steps 1-3 until a time or accuracy criteria is met.

Once this process is done the network is now ready to classify handwritten digits. This produces a probabilistic output of each intended digit in the test sheet. The next step now is to incorporate this character evidence with the bubble evidence to produce a accurate estimate of the intended student entries. A method to do this is discussed next.

## 4.2 Probabilistic Graphical Models

The final step in determining the students answers is to probabilistically determining the most likely answers, given the bubble and character evidence presented. To do this 2 probabilistic graphical models (PGM) are implemented.

### 4.2.1 Overview of the system

For this project a graphical approach is followed in representing the software developed. A graphical model in essence allows software to be represented as information (nodes or circles) and relationships (directed arrows). The directions of the arrows represents what information causes other information to be created, thus given a parent to offspring interpretation). An example of such a graph can again be seen in Figure 4.9. These graphs allows for intuitive reasoning about how the system should operate. For this project an observation is made in the form of an image which is used as evidence. The system is then tasked with predicting what the written entries are given this information. Due to the probabilistic nature of the system a probabilistic component is also needed in this graphical models. Thus a probabilistic graphical model (PGM) is used.

A probabilistic graphical model (PGM) is a probabilistic graph containing random variables, where the graph expresses the conditional independence structure between these variables. The type of PGM used for this project is a Bayesian network. A Bayesian network models a set of random variables and their conditional dependencies via a directed acyclic graph (DAG). As seen in Figure 4.9, arrows are used to indicate which variables are conditionally dependent on others.

### 4.2.2 Estimating the intended answer

The figure should be interpreted in the direction which information flows. Originally a student has a certain digit that he/she wants to portray on the page. This is given by the 'Digit user intended bubble'. There are 10 possible digits to consider and thus the bubble has 10 possible states. The intended digit gives rise to the intended bubbles and character that the student wants to write down. The student might sometimes mistakenly think that the first bubble represents 0 and thus even if the intended digit is 0 the intended bubble might be 1. Thus, this must also be done in a probabilistic manner to compensate



## 4.2 Probabilistic Graphical Models

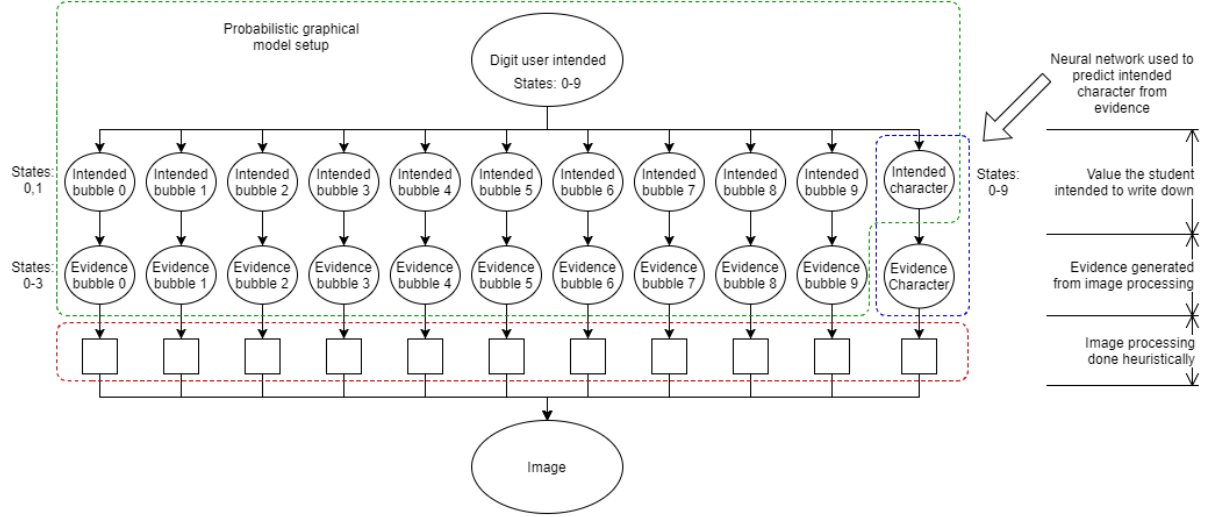


Figure 4.9: Graphical setup for determining the intended digit written by a student.

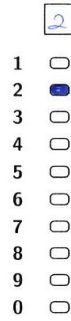


Figure 4.10: Column with the evidence that gets considered for the calculation of an intended digit.

for this uncertainty. The intended bubbles and character then produces evidence, as seen in Figure 4.9.

When looking at Figure 4.10, it is observed that there are 11 evidence areas to consider. They are the the 10 bubbles and the character block. The process of writing down this digit introduces so noise into the system, due to the fact the student is not always going to write down the digit in exactly the same way. Thus the evidence is also probabilistically linked to the 'intended bubble' and 'intended character' parent distribution. This evidence then gets written down on the paper and is what ultimately influences how the image looks. Each of the bubbles can take one of 4 states as evidence. These

states are blank, crossed-out, partially coloured in and fully coloured in. The character block evidence is a 28 by 28 greyscale image. Thus it can have  $28 \times 28 \times 256$  possible states, where 256 is the possible pixel intensities of each pixel. This value with true range 0 to 256, gets converted into a normalized value between 0.0 and 1.0 before it gets assigned as inputs to the neural network.

After the model is constructed the intended digit needs to be estimated, with the image as evidence. This can be done by reasoning from the bottom (image evidence) upwards to the intended digit. The first step is to process the image, using of image processing, to produce more tractable evidence. Producing the bubble evidence from the image is described in Chapter 3. In Section 4.1.1, the process to extract the character evidence from the image is also described. Using the neural network the probability of intended character, given the character box, can be determined. After these steps is completed the intended digit can be estimated using the PGM. This is done by first setting the evidence node to the values calculated when image processing were done on them. The 'intended character' node then gets set to prior calculated through the use of the neural network. After this is done inference is done on the PGM and the resulting probability of each digit is calculated for the 'intended digit node'. The PGM structure can again be seen in Figure 4.9.

There are 8 possible columns to use for an answer. The first column represents the sign of an answer. Thus two signs are possible. For each of the remaining 7 columns a number from 0 - 9 can be represented. Thus there are ten possible values for each column. This gives a possible number of values that a answer can take to be  $2 \times 10^7 = 20000000$ . Each of these states needs to be calculated and becomes computationally intractable. Thus a assumption is needed to reduce the number of possible states. A fair assumption to make, is that all 20000000 possible values are equally likely to be written down. Thus each column digit becomes independent of the another columns values. This means that if a value in one column is known, it does not influence the probabilities of the other columns being a certain value. Thus the number of states to calculate now becomes  $2 + 10 \times 7 = 72$ , because each column's states can now be calculated independently. Thus knowing this independent property the answer model can be described as a combination of 7 digit models with a heuristic calculation of the intended sign. This model is illustrated in Figure 4.11.

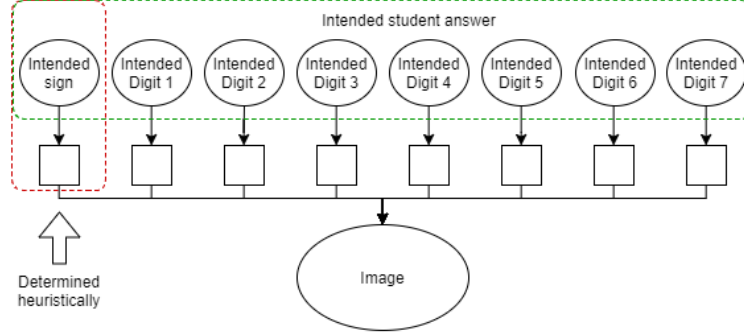


Figure 4.11: Graphical setup of determining student answer.

In the previous section it was found that determining the intended sign and digit for each column the intended answer can be found. Thus the intended digit for each column is calculated separately using the digit model, as described in Section 4.2.3. The intended sign is determined heuristically by using the image processing methods described in Chapter 3 to find if the bubble underneath the sign is coloured in. Once this is done the intended answer is determined by combining the estimated sign and digits in the order seen in Figure 4.11.

### 4.2.3 Estimating the student number

For a student number additional information is present. Knowing the digit value of one column changes the probability of each other column taking a certain value. The reason for this is, because there is only a limit number of student numbers to consider. Thus if the first digit is a 2, only student numbers starting with 2 still have to be considered. To account for this fact an additional node is added above the individual digit probabilities. This node represents the probability of each student number being present in the image and is related to the intended digits. The student number thus produces the intended digits. This node will have  $+ - 900$  states, depending on the number of student numbers. The student number graph can be seen in Figure C.3.

Now that the graph has been setup the model can be used to infer the most probable student number. By setting all the bubble evidence and character priors the student number probabilities can be inferred. This model allows for a accurate result, because most student numbers are quite different to one another. It is found that if the student

provides only character information and no bubbles are coloured in, the student number can still be estimated reliably. These results are stated in Appendix D.

### 4.2.4 Training of a Probabilistic Graphical Model

For this project the conditional probabilities for the intended digit model was found by using training data. From these distributions the answer model can also be constructed due to it only consisting of independent digit models. The probability of a student number given the intended digits was estimated manually.

As seen in Figure 4.9, the digit PGM will include the bubble evidence as well as a prior probability which the neural network provides. Once these values are assigned, the PGM model will infer the intended digit using the conditional probability distributions specified. To do this the conditional probabilities of the PGM, must first be determined from data.

The process of training a PGM is a simple process once enough training set is present. This was done by allowing the initial software, without the PGM model, classify the digits. Thus the training set contains the bubble and neural network outputs as evidence. The training set then also includes the intended bubbles and intended digit, as estimated by the image processing unit. Once all these test data are gathered, the conditional probabilities can be calculated. These probabilities then get stored and used as prior probabilities in the PGM.

## 4.3 Conclusion

This chapter looked at two machine learning techniques to improve the accuracy with which the system infers the answers written on each scanned test sheet. A method was shown, using a neural network, to estimate the probability of each digit given only the character box as input. Additionally a approach was discussed, using a PGM, to allow the system to make a final prediction of what the student intended to write down given the bubble and character boxes as evidence. This PGM method is found to be accurate enough to determine the student number by only using the character recognition information provided.

The following chapter will cover the validation and results of the system from weekly grading done for the Applied mathematics department.

# Chapter 5

## Analysis of results

In the following chapter a study is described on the accuracy of the test grading system. In the first two sections of this chapter the basic and complete system are compared using the same datasets. The first test grading system includes only image processing techniques, described in Chapter 3. The second system contains the complete project software with all the machine learning techniques described in Chapter 4. In the last section a description of 3 additional validation tests is given. These tests are done on tutorials written by Applied Mathematics students as is described in the agreement with the department.

Each systems is assessed in the next section under 3 categories. Firstly some marking statistics will be given on the tests grade. The second category describes all the tests, the system appointed to be marked manually, because it was not sure what the correct answer is. In this catagory the tests gets sent to a clashlist. After all the test is graded the systems displays a interface with values it thinks the student wrote down. An additional image is also displayed. The user is then tasked with scanning through each of the clashed tests, using the interface, to see if any tests is graded incorrectly by the system. This process is normally fast, because the system has a high accuracy in guessing the answers correctly. Lastly, all the answers that the system did not assign for manual marking, but identified incorrectly will be described.

## **5.1 Results of 25 test cases**

In this section the results of grading 25 hand picked tests is compared between the two systems. Among the 25 tests there are different categories that tests different aspects and limitations of the systems. The categories and the number of case included can be seen in the list below:

1. Test with crossed out answers(7).
2. Test with lightly coloured entries or partially coloured in entries(4).
3. Test with negative signed answers(1, but also included under other categories as well).
4. Test with no bubbles and only characters filled in for a specific entry field(5)
5. Test with no characters filled in and only bubbles for a specific entry field(2)
6. Test with data filled in correctly(2).
7. Random page with no template on them(2).
8. Page with tilted template inside image(2).

These test is specifically chosen, because in combination it approximates all the types of tests the system has to asses. Most of the tests are slightly extreme cases of what students has filled in on tutorial tests.

### **5.1.1 Basic system**

#### **5.1.1.1 Marking statistics**

Using this basic system an average time for each test is calculated to be 0.305 seconds.





(b)      2 .  7  1  0

-

1	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	1
2	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	2
3	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	3
4	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	4
5	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	5
6	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	6
7	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	7
8	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	8
9	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	9
0	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	0

Figure 5.2: Student filled in answer with only character information.

## 5.1.2 Complete system

### 5.1.2.1 Marking statistics

Using this complete system an average time for each test is calculated to be 2.011 seconds.

### 5.1.2.2 Clashlist

A total of 6 out of the 25 test where reported as clashes. The two random images were both reported in the clash list. One test was reported due to it only having characters written in with no bubbles. Thus even though the system identified every character correctly, it had to low of a percentage faith in its answer and reported it to the clash list. The final two cases that was reported to the clash list was due to the character recognition determining a crossed out character as the intended character. An example of this is shown in Figure 5.3.

### 5.1.2.3 Incorrect automatic graded results

There were no automatically graded answers that was done incorrectly.

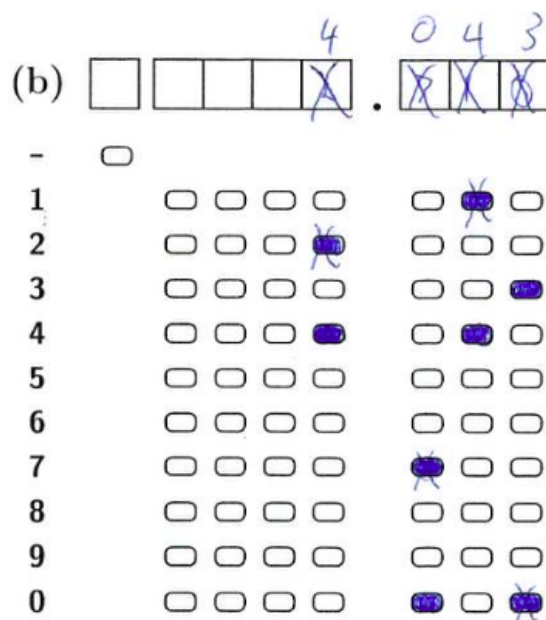


Figure 5.3: Crossed out character that confused the grading system.

### 5.1.3 Analysis or results

As is seen in the section below, the complete systems does not reduce the number of clashes for these 25 images. The complete system sometimes classifies a crossed out character is the intended character in that block. But because it is crossed out the system assigns a low certainty to that value, as the neural network is not to sure which digit it is. A drawback of the completed system is that it takes on average 2.011 seconds to grade a tests. This is due to the student number PGM taking on average 1.5 seconds to infer the correct student number. The complete system had no incorrectly automatically graded answers in contras to the 4 graded wrongly using the basic system. A reason to this is attributed to more evidence that gets considered in the complete system. Thus only if that evidence match up will the system be certain enough to accept its answer as the correct one. In the next section the complete system use used to grade a tutorial written by students.

## **5.2 Grading of tutorial tests**

In this section the complete system is tested on the final tutorial test written by the student class. Thus the final version of the complete system was used in grading the students' tests. To analyse the results student feedback was recorded to locate tests that were graded incorrectly.

### **5.2.1 Marking statistics**

For these tutorial test an average marking time per test of 2.3 seconds was recorded.

### **5.2.2 Clashlist**

In total there were 67 clashes in the 888 tests. These 67 clashes are categorised in [Table 5.1](#):

### **5.2.3 Incorrect automatic graded results**

For this tutorial 888 test were written. To check through every test manually to find mistakes becomes impractical. Thus the students were asked to report any results that were marked wrong by the system. This is all the results that the system decided to automatically grade and in doing so graded the test wrong.

Only 1 result was reported where the software automatically marked the wrong answer to a test. The correct answer was -95.0 and the system marked the answer as 95.0, as seen in [Figure 5.4](#). There might still be test that were marked wrong, but these test(s) were not reported.

For a more detailed description of the results from grading all 4 tutorial tests, refer to [Appendix E](#).

### **5.2.4 Analysis of results**

In conclusion it is noted that the complete system, with its machine learning capabilities, slightly reduces the number of tests that the user must mark manually from the previous basic system. About 7.5% of the tests in the tutorial had to be marked manually, due

Number of tests in category	Category description
41	In these tests the system guessed the right values, but with a certainty value below 90%. Some of the cases was when the student number was only filled in the character box. The software always identified the correct student number, but was below 90% sure of its overall test answer.
15	In these tests the system could not distinguish between a crossed out answer and correct answer. This is due to the crossed out answer being interpreted as a filled in answer.
8	These tests have an answer with only character information in them. The system tried to identify each answer, but made a mistake in atleast one of the digits.
2	These images contained blank papers that did not include test templates.
1	In this test the grid of the test paper can not be found. Thus the test could not be marked.

Table 5.1: Table describing number of clashes in the different catagories.

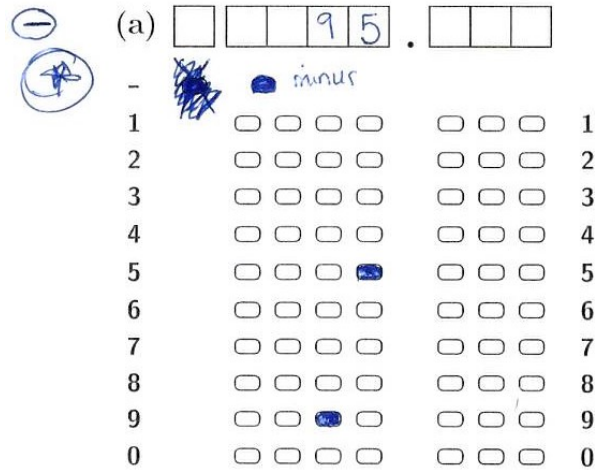


Figure 5.4: Incorrectly identified answer as 95.

to the system being unsure of its answers. The system does however have a high probability of grading tests correctly if they are done automatically. A reason for this can be attributed to the fact that the system correlates two pieces of evidence to predict the correct answer. Thus both the character and bubble information has to be interpreted incorrectly for the system to automatically grade a answer incorrectly.

Thus overall the system could graded 92.5% of all the test correctly and automatically. From the remaining 7.5% only 1 test or 0.1% of the tests was found to be graded wrongly. For the remaining 7.4% of the tests the system was not certain enough to grade these test automatically. The system did classify most of the clashlist test correctly. This means the system could possible have graded 97.1% of the tests correctly. This can be done by changing a threshold value, but allows the other 2.9% of the tests to be classified incorrectly.

# Chapter 6

## Summary and conclusions

### 6.1 Project summary

In this project an automatic test grading system is developed with the aim of grading student test using a special template. Firstly research was done into excising methods of grading test automatically. It was found that these software packages normally uses only image processing methods to graded bubbles on a paper. For this project additional machine learning capabilities was also built into the system. This allows for a better estimation of what the student wanted to write down on the paper.

### 6.2 How this final year project benefits society

In the African society there is a great number of individuals who does not have access to quality educational opportunities. The educational systems these individuals do have are normally of pour grade with limited teaching assistance. Educators who are available are not accessible to learners to provide quality education. Automatic Mark Recognition software like the one developed in this project allows for a great number of tests to be assessed automatically in a short time span. This gives educators the power to handle bigger classes and thus provide more learners the opportunity for a better education.

### 6.3 What the student learned

During the execution of this project, the student learned that time management is important to complete an project in due time. Time management also allows an individual

to continuously assess how he/she is doing with respect to a schedule. This not only increases performance, but also self confidence in the final product. Finally the student learned how to develop a software package under a deadline. This project also allowed the student to gain a basic knowledge a a broad range of fields including image processing, neural networks and probabilistic graphical models.

## 6.4 Future improvements

To increase the speed of grading tests it should be considered to use a faster PGM library to inference the intended student number. This can be done by using Stellenbosch's emdw library. Next a improvement in estimating the bubble locations can be made by updating the orientation of the template as more bubble contours gets classified. This allows for example the final student number bubbles to be estimate extremely accurately. Further increases in test grading speed can be achieved by only doing image processing on the expected locations of the bubbles. This will bring some extra technical hurdles, but if solved can significantly increase the software's speed. Further the accuracy of the character recognition neural network can be increased by making use of Generative Adversarial Networks(GAN) to train the network on actually classified test results.

## 6.5 Conclusion: Summary and conclusions

For a tutorial setting with around 890 tests the system takes  $\pm 30$  minutes to grade these tests automatically. This time is longer than other OCR software, but allows for less limited answers to be given by a student. An example of these answers are the system's capability to identify a student number by only referring to the characters written in the student number box.

In conclusion a test grading system was build that can automatically grade tests with a 97.1% accuracy. The reason for this accuracy not being at 100% is mainly due to some students crossing out answers only partially and thus confuses the system. An additional feature is implemented that transfers tests, the system uncertain about, to a user to manually grade using the software. In combination with this manually checked tests, the system achieves an overall accuracy of 99.8%. Thus on average only 1 test in every tutorial session of around 890 tests gets graded incorrectly.

# References

- ANKAN, A. & PANDA, A. (2015). pgmpy: Probabilistic graphical models using python. PROC. OF THE 14th PYTHON IN SCIENCE CONF. [9](#), [10](#)
- EDORAS (2017). The inverse radon transform. Image Processing Toolbox. [viii](#), [14](#)
- GOOGLE (2017). Deep mnist for experts. [9](#)
- IVETIC, D. & DRAGAN, D. (2003). Projections based omr algorithm. [7](#)
- KARPATHY (2017). Cs231n convolutional neural networks for visual recognition. [viii](#), [24](#)
- MATHWORKS (2017). Detect lines using the radon transform. [8](#)
- NIELSEN, M.A. (2015). *Neural Networks and Deep Learning*, vol. 1. Determination Press. [24](#)
- PATEL, N.V. & PRAJAPATI, G.I. (2003). Various techniques for assessment of omr sheets through ordinary 2d scanner: A survey. *International Journal of Engineering Research & Technology (IJERT)*, **4**. [8](#)
- ROSEBROCK, A. (2016). Bubble sheet multiple choice scanner and test grader using omr, python and opencv. [8](#)
- TENSORFLOW (2017). Mnist for ml beginners. [viii](#), [23](#)
- VIJAYAFORM (2017). Omr sheet. [viii](#), [7](#)



# Appendix A

## Project plan

The final year project plan is shown in Figure [A.1](#). This idea of this Gantt chart plan was to give the student an indication of the progress of the final year project. It was last updated as a final adjustment to the project report.

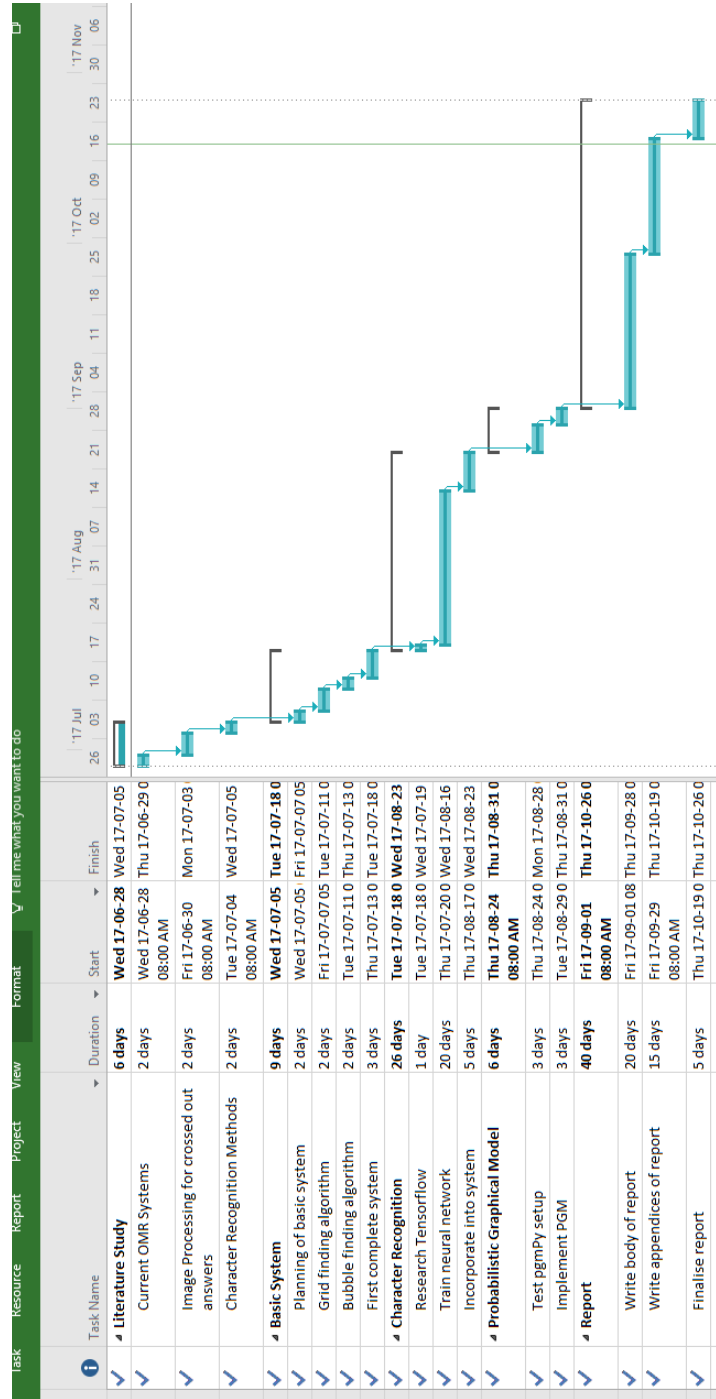


Figure A.1: Project plan for the final year project.

# Appendix B

## Outcome compliance

Maak die kolomme outcomes, chapters, description

Outcome	Reference	
	Sections	Pages
1. Problem solving: Demonstrate competence to identify, assess, formulate and solve convergent and divergent engineering problems creatively and innovatively.	<i>All</i>	<i>All</i>
5. Engineering methods, skills and tools, including information technology: Demonstrate competence to use appropriate engineering methods, skills and tools, including those based on information technology.	<i>2, 3, 4, 5, 6 &amp; 7</i>	<i>10 – 62</i>
6. Professional and technical communication: Demonstrate competence to communicate effectively, both orally and in writing, with engineering audiences and the community at large.	<i>All</i>	<i>All</i>
9. Independent learning ability: Demonstrate competence to engage in independent learning through well developed learning skills.	<i>2, 3, 4 &amp; 5</i>	<i>10 – 37</i>
10. Engineering professionalism: Demonstrate critical awareness of the need to act professionally and ethically and to exercise judgment and take responsibility within own limits of competence.	<i>8.3 &amp; 8.4</i>	<i>66 – 67</i>

# Appendix C

## Overview of system

In this chapter a graphical and mathematical derivation of the entire system is given.

A.b onder aan j- vir sum over all derivation sit die all teken onder aan die sumation

### C.1 System as a whole

As described in Chapter 1, the system can fundamentally be represented with 6 information nodes. These nodes are shown in Figure C.1. The student has certain information he/she wants to portray on the paper. This includes the 4 answers and student number the student wants to write down. Thus those 5 nodes gives rise to the image, representing the last node. Thus fundamentally the system is tasked with working out these 5 conditional probabilities:  $P(StudentNumber/Image)$ ,  $P(Answer1/Image)$ ,

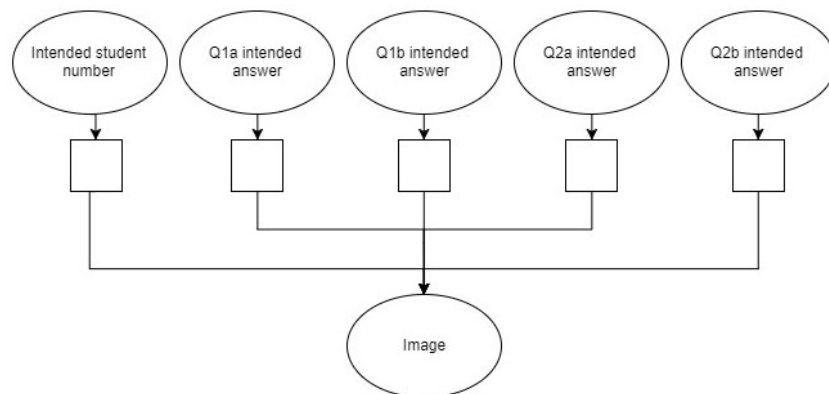


Figure C.1: System overview.

$P(\text{Answer2}/\text{Image})$ ,  $P(\text{Answer3}/\text{Image})$  and  $P(\text{Answer4}/\text{Image})$ . The random variables StudentNumber and Answer(1 to 3) thus represent all the possible values that the student number and answers possibly can be. The image is a random variable representing the total number of possible states the image can represent. This value is in the order of  $1240 \times 1754 \times 256$  possible states. To practically represent this further assumptions are made in the following sections.

The aim of the software is to maximize the likelihood of those probability distributions indicating the correct answers. The reason the problem is represented in a probabilistic way is due to the fact that different images are going to be generated every time a test is written. This is true even when the same information is going to be portrayed. Everytime a student writes a test he/she is going to write in different ways. A probabilistic graphical model (PGM) is thus used to describe this system and its inner operations. For a more detailed explanation on PGM's, refer to Section 4.2. The blocks in Figure C.1 represent additional processing that is described in the following sections.

To calculate those 5 probabilities, some more detailed derivations are necessary. These derivations are described in the next section.

## C.2 Deriving the intended student entry

### C.2.1 Student answer

In Section 4.2.2 it was determined that the student answer can be calculated by combining the intended sign and digits of each column to form an answer. The reason for this could be attributed to the fact that these digits were independent of one another. This means that the intended digit in a certain column is not influenced by what the values in other columns are. In Equation C.1 this independence property can now be seen. To find the most probable answer only  $P(\text{sign}/\text{Image})$  and  $P(\text{digit}(1 - 7)/\text{Image})$  still needs to be calculated. Using the image processing techniques described in Section 3 the  $P(\text{sign}/\text{Image})$  can simply be determined heuristically by determining the probability of the bubble being coloured in, underneath the sign. Thus the only values yet to calculate is  $P(\text{digit}/\text{Image})$ . This is described in Section C.3.

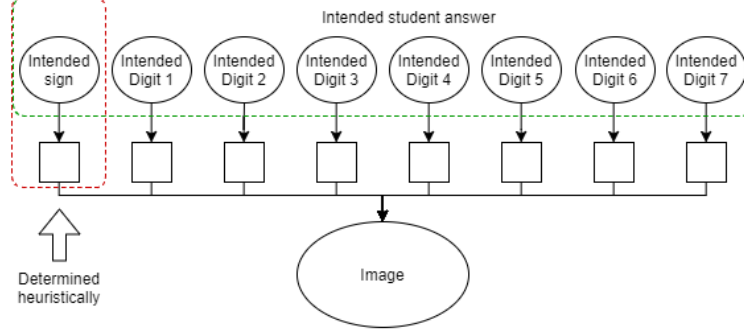


Figure C.2: Graphical setup of determining student answer.

$$P(\text{Answer}/\text{Image}) = P(\text{sign}/\text{Image}) * P(\text{digit1}/\text{Image}) * ... * P(\text{digit7}/\text{Image}) \quad (\text{C.1})$$

### C.2.2 Student number

As state in Section 4.2.2 the assumption of independence does not hold in the case of a student number. The reason for this is, because every 8 digit number is not equally likely to be a student number. Only student numbers that are valid will be considered as a possible state that the student number node can take on. This node will have  $+ - 900$  states, depending on the number of student numbers. The student number graph can be seen in Figure C.3. Next a derivation for  $P(\text{studentNumber}/\text{Image})$  is needed.

In the below equations, 'Digits' represents the 8 digit random variables. The first step is to apply Bayes rule as seen in Equation C.2.

$$P(\text{StudentNumber}, \text{Digits}/\text{Image}) = \frac{P(\text{Image}/\text{StudentNumber}, \text{Digits}) * P(\text{StudentNumber}, \text{Digits})}{P(\text{Image})} \quad (\text{C.2})$$

When all the digits are known, the image becomes independent of the student number. This can be seen in Equation C.2. Also the chain rule states that Equation C.4 must be true.

$$P(\text{Image}/\text{StudentNumber}, \text{Digits}) = P(\text{Image}/\text{Digits}) \quad (\text{C.3})$$

---

## C.2 Deriving the intended student entry

$$P(StudentNumber, Digit) = P(Digit/StudentNumber) * P(StudentNumber) \quad (C.4)$$

Next the Digit random variable is summed out to produce  $P(studentNumber/Image)$ , as seen in Equation C.5.

$$P(StudentNumber/Image) = \sum_D \frac{P(Image/Digits) * P(Digit/StudentNumber) * P(StudentNumber)}{P(Image)} \quad (C.5)$$

Applying Bayes rule again the following equality can be presented, as seen in Equation C.6.

$$P(StudentNumber/Image) = P(Image/Digits) = \frac{P(Digits/Image) * P(Image)}{P(Digits)} \quad (C.6)$$

Thus the result can be seen in Equation C.7.  $P(StudentNumber)$  can be taken out of the sum, due to it being independant of  $Digits$ .

$$P(StudentNumber/Image) = P(StudentNumber) * \sum_D \frac{P(Digits/Image) * P(Digits/StudentNumber)}{P(Digits)} \quad (C.7)$$

$P(StudentNumber)$  can be initialized as a equal distribution, because every student number is equal as likely to be in a given test.  $P(Digits/StudentNumber)$  is a value that is trained from data. This value symbolizes the probability that the user intended to write down a given digit given that student's student number. This number is thus strongly correlated with the student number. If the first digit of the student number is 1, Digit1 will have a high probability of being 1. The only values that still needs to be calculated are thus  $P(Digits/Image) = P(Digit1/Image) * ...*$ . This derivation is covered in the next section.

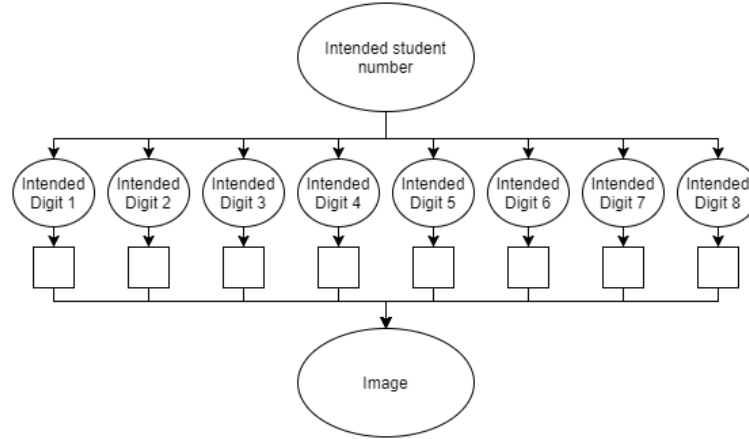


Figure C.3: Graphical setup of determining student number.

## C.3 Deriving the estimated digit



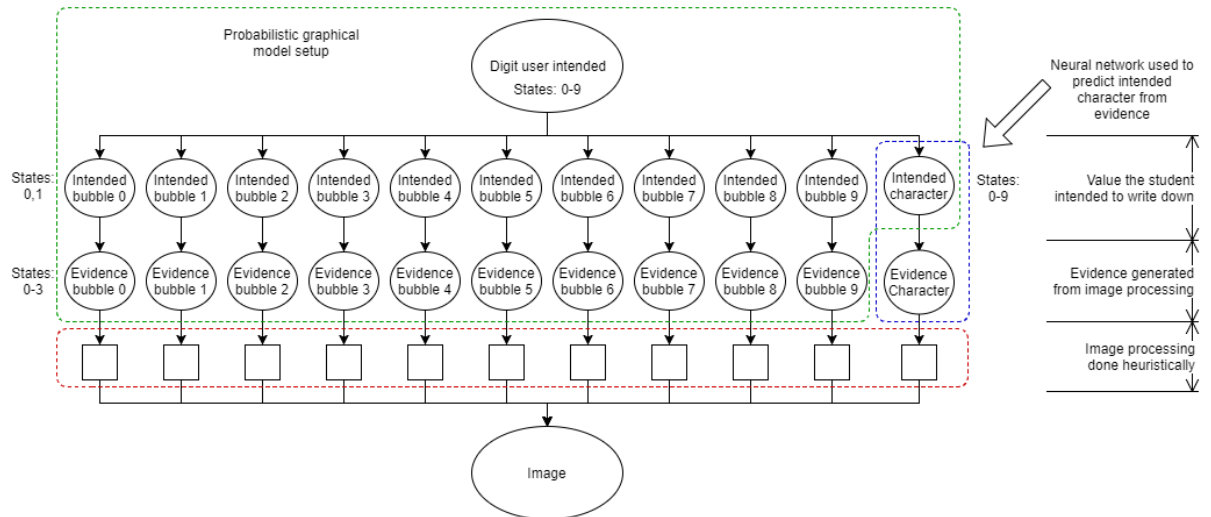


Figure C.4: Graphical setup of determining intended digit.

# Appendix D

## Implementation/Algorithms

Algorithms used in this project.

# Appendix E

## Validation and results

Validation and results of this system.

### E.1 Deep Convolutional Neural Network test results

Accuracy for character recognition:

Using default deep neural network. Up to V22

Test accuracy for network trained on MNIST dataset: On test data for MNIST: 0.9935

On test data for generated digits: 0.666667

Main reason for error. The MNIST dataset has images that are more concentrated to being binary. Either black or white.

Test accuracy for network trained on generated digits: On test data for MNIST: 0.9462 On test data for generated digits: 0.921569

Reasons The algorithm above does not work that well on the characters:

Borders are not cut out properly.

Neural network focusses on color as well which decreases its accuracy for shape.

Attempt 2:

Accuracy for character recognition:

Using default deep neural network. V23

Test accuracy for network trained on MNIST dataset: On test data for MNIST: 0.9935

On test data for generated digits: 0.666667

Main reason for error. The MNIST dataset has images that are more concentrated to being binary. Either black or white.

## **E.1 Deep Convolutional Neural Network test results**

---

Test accuracy for network trained on generated digits: On test data for MNIST:  
0.9462 On test data for generated digits: 0.921569

Reasons The algorithm above does not work that well on the characters: