

Erik Dale
Håvard Østli Fjørkenstad
Yeshi Jampel Pursley

Image Content and Hand Writing Analysis of the Dead Sea Scrolls

Bachelor's thesis in Computer Science

Bachelor's thesis in Computer Science

Supervisor: Aditya Suneel Sole

Co-supervisor: Marius Pedersen

May 2022



NTNU

Kunnskap for en bedre verden

Erik Dale
Håvard Østli Fjørkenstad
Yeshi Jampel Pursley

Image Content and Hand Writing Analysis of the Dead Sea Scrolls

Bachelor's thesis in Computer Science

Bachelor's thesis in Computer Science
Supervisor: Aditya Suneel Sole
Co-supervisor: Marius Pedersen
May 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science



Kunnskap for en bedre verden

Abstract

The Department of Information Security and Communication Technology (IIK) at the Norwegian University of Science and Technology (NTNU) in Gjøvik wanted a methodology that can help automatically extract and classify information, such as letters, from the Dead Sea Scrolls (DSS), later for them to use this information to identify authorship and date of the scrolls. This project investigates possible methodologies and designs and implements a solution for letter extraction from the DSS. Images of the DSS are first enhanced using various computer vision enhancement methods such as contrast improvement and noise removal. This is done to increase the accuracy of the segmentation and classification of the letters respectively. After enhancement, the letters in those images are segmented with Pytesseract. Our solution contains a method for splitting segments, such as words or a few connected letters in a word. A dataset of the 22 ancient Hebrew letters has been generated to train and test a Convolutional Neural Network (CNN). This model then classifies the segmented letters. In this thesis work we have shown that the overall methodology we have designed and used on the dataset we have generated performs better than the results from literature. A user interface has also been made to implement the image enhancement, the image segmentation, and the deep network classification in sequence and allows other functionalities such as opening, closing, cropping and saving of images.

Sammendrag

Institutt for informasjonssikkerhet og kommunikasjonsteknologi (IIK) ved Norges teknisk-naturvitenskapelige universitet (NTNU) i Gjøvik ønsket en måte å automatisk ta ut informasjon, for eksempel bokstaver, fra Dødehavsrullene, for så senere å bruke denne informasjonen for å identifisere forfatteren og perioden til Dødehavsrullen. Dette prosjektet undersøker mulig metodikk og design for å implementere en løsning som finner og tar ut bokstavene fra Dødehavsrullene. Bilder av Dødehavsrullene blir først bildebehandlet ved hjelp av ulike maskinsyn metoder som kontrastforbedring og støyfjerning. Dette gjøres for å øke nøyaktigheten av henholdsvis segmenteringen og klassifiseringen av bokstavene. Etter forbedring blir bokstavene i disse bildene segmentert med Pytesseract. Vår løsning inneholder en metode for å dele segmenter, som for eksempel ord eller sammenhengende bokstaver inne i et ord. Et datasett med de 22 gamle hebraiske bokstavene er generert for å trene og teste et konvolusjonelt nevralt nettverk. Denne modellen klassifiserer deretter de segmenterte bokstavene. I denne oppgaven har vi vist at den overordnede metodikken vi har designet og brukt på datasettet vi har generert, gjør det bedre enn resultatene fra andres tidligere verk. Et brukergrensesnitt har også blitt laget for å implementere bildeforbedringen, bilde-segmenteringen og den dype nettverksklassifiseringen i denne rekkefølgen. Den tillater også andre funksjoner som åpning, lukking, beskjæring og lagring av bilder.

Contents

Abstract	iii
Sammendrag	v
Contents	vii
Figures	xi
Tables	xv
Code Listings	xvii
Acronyms	xix
Glossary	xxi
1 Introduction	1
1.1 The Dead Sea Scrolls	2
1.2 Project Objective	3
1.3 Goals and Frames	3
1.3.1 Frames	3
1.3.2 Result Goals	3
1.3.3 Effect Goals	3
1.3.4 Problem Delimitation	4
1.4 Group Background	4
1.5 Organization	4
1.5.1 Roles and Responsibilities	4
1.6 Structure of the Report	6
2 Development Process	7
2.1 Development Model	7
2.1.1 How We Divided Our Work	8
2.1.2 Gantt-Diagram	9
2.2 Meetings	9
2.2.1 Meetings With Employers	9
2.2.2 Meetings With Supervisors	10
2.2.3 Internal Meetings	10
2.2.4 Meetings With Dead Sea Scroll Expert	10
3 Background	11
3.1 Dead Sea Scrolls	11
3.1.1 Scrolls and Columns	11
3.1.2 Letters	12
3.2 Theory	12

3.2.1	Computer Vision	12
3.2.2	Image Enhancement	12
3.2.3	Testing of Image Enhancement	15
3.2.4	Image Segmentation	16
3.2.5	Testing of image segmentation	17
3.2.6	Machine Learning	18
3.3	Technologies	22
3.3.1	Python	22
3.3.2	OpenCV	23
3.3.3	LabelImg	24
3.3.4	Tesseract	25
3.3.5	QT Box Editor	25
3.3.6	PyTorch	26
3.4	State of The Art	27
3.4.1	Image Enhancement	27
3.4.2	Image Segmentation	28
3.4.3	Machine Learning	29
3.4.4	Summary	29
4	Methodology	31
4.1	Dataset	31
4.1.1	The Size of the Dataset	32
4.1.2	Acquiring the Dataset	32
4.1.3	Variance in The Dataset	32
4.1.4	Usage of the Dataset	33
4.2	System Pipeline	34
4.3	Image Enhancement	34
4.3.1	Contrast Improvement	34
4.3.2	Noise Removal	36
4.4	Image Segmentation	39
4.4.1	Binarization	39
4.4.2	Using Pytesseract to Segment Letters	39
4.4.3	Word Splitter	42
4.4.4	Custom TRAINEDDATA File	44
4.4.5	Calculating the Intersection over Union Score	45
4.5	Machine Learning	46
4.5.1	Model	46
4.5.2	Model Improvements	48
5	User Interface	51
5.1	Functional Requirements	51
5.1.1	User Patterns	51
5.2	Sketch	53
5.3	Method	54
5.3.1	PyQt5	55

5.3.2	Implementation of Image Enhancement, Image Segmentation and Machine Learning	56
5.3.3	Design	56
5.4	Results and Discussion	58
5.4.1	Flow Chart	58
5.4.2	Final Solution	60
5.4.3	Classification Times	62
5.4.4	Future Improvements to the User Interface	62
6	Quality Assurance	65
6.1	Code Quality	65
6.1.1	Tools	65
6.1.2	Documentation	66
7	Results	67
7.1	Image Enhancement	67
7.1.1	Adaptive Histogram Equalization	67
7.1.2	Morphological Transformations	70
7.1.3	Blur and Denoising Methods	71
7.1.4	Image Enhancement Process	73
7.2	Image Segmentation	74
7.2.1	Binarization Results	74
7.2.2	Pytesseract Results	76
7.2.3	Word Splitter Results	77
7.2.4	Custom TRAINEDDATA Results	82
7.2.5	IoU Results	83
7.3	Machine Learning	84
7.3.1	Prediction Results	87
7.3.2	Comparison between epochs	90
8	Discussion	91
8.1	Project Process	91
8.1.1	Working Environment	91
8.1.2	Planning of the Project	91
8.1.3	Meetings	92
8.1.4	Development Process	92
8.2	Technical Results	92
8.2.1	Modifiable and Expandable Code	92
8.2.2	Image Enhancement	93
8.2.3	Image Segmentation	93
8.2.4	Machine Learning	95
9	Conclusion	97
9.1	Goals Achieved	97
9.2	Future Work	97
	Bibliography	99
A	Standard Agreement	105
B	Confidentiality Agreement	117

C	Problem Statement	121
D	Project Plan	123
E	How We Created a Custom TRAINEDDATA File	153
F	User Interface Repository Link and README	155
	F.1 Link	155
	F.2 README	155
G	Repository Link and README	159
	G.1 Link	159
	G.2 README	159
H	YOLO to Images Code	163
I	Time Tracking	165

Figures

1.1	Figure of a Dead Sea Scroll Image	1
1.2	Figure of The Great Isaiah Scroll	2
2.1	Our Kanban board on GitHub	7
2.2	Our Gantt-Diagram	9
3.1	Figure of the old Hebrew alphabet	11
3.2	Figure of histograms before and after adaptive histogram equalization	13
3.3	Figure of a skeletonized image	17
3.4	Figure that shows an example of a poor, good and excellent IoU score. Source: [23].	18
3.5	Figure of a neural network	19
3.6	Figure of a neuron	19
3.7	Visual example of a confusion matrix. Green are correctly labeled, while red are incorrectly labeled. In this case the Dog precision would be true positives divided by the dog column, while recall would be the true positives divided by the dog row	20
3.8	Figure of a convolution process	21
3.9	Example of how max pooling works.	21
3.10	Example of an Alef that has been eroded	22
3.11	Example of LabelImg's user interface	24
3.12	Example of QT Box Editor's user interface	26
3.13	Binarization with BiNet	28
3.14	Skeletonized image and histogram of the vertical image pixels quantity	29
3.15	Jain's Pipeline	30
4.1	Figure of Our Overall System Pipeline.	34
4.2	Figure of The Great Isaiah Scroll with and without noise removal	38
4.3	Describes the segmentation process.	39
4.4	A single larger letter and two narrow letters that are connected.	41
4.5	Image of scroll, gotten from [4], with rectangles around letters. Includes also the ID of the letter, which does not come with the <code>cv2.rectangle</code> method.	41

4.6	Gives an overview of the word splitter's process.	42
4.7	Image of the letters "Mem" and "He" connected.	42
4.8	Two letters connected to each other. Each letter begins with a straight line, shown with a red line, when reading from right to left.	43
4.9	A letter from the DSS and the same letter from modern Hebrew. . .	45
4.10	Figure of the LeNet architecture	46
5.1	Use Case Diagram	52
5.2	Sketch of User Interface	54
5.3	Example of PyQt5 window	56
5.4	Error Messages and Loading Animation	57
5.5	Flowchart For Our User Interface	59
5.6	Our User Interface	60
5.7	Our User Interface When an Image is Displayed	60
5.8	Cropping in Our User Interface	61
5.9	Cropped Image in Our User Interface	61
5.10	Classified Image in Our User Interface	62
7.1	Figure of Contrast Improvement Resultsl	67
7.2	Figure of The Great Isaiah Scroll Column 1	69
7.3	Figure of Morphology Results	70
7.4	Figure of Opening Results	70
7.5	Figure of Image Enhancement Resultsl	71
7.6	Flowchart of Image Enhancement Process	73
7.7	The segmenter with the word splitter performed on a paragraph from the Great Isiah Scroll column 35 using our custom TRAINED-DATA file. Blue rectangles are successful segments while the red rectangles are segments that are either too large or too small. Only the letters surrounded by blue rectangles are saved.	74
7.8	Binarization Results	75
7.9	More Binarization Results	76
7.10	The segmenter without the word splitter performed on a paragraph from the Great Isiah Scroll column 35 using our custom TRAINED-DATA file. Blue rectangles are successful segments while the red rectangles are segments that are either too large or too small.	77
7.11	Examples of problems with the segmenter.	77
7.12	The letters "He" and "Mem", number 53 and 54, found in Figure 7.7, were successfully split.	78
7.13	The letters "He" and "Nun", number 68 and 69, found in Figure 7.7, were successfully split.	78
7.14	The letters "Yod" and "Mem", number 142 and 143, found Figure 7.7, were successfully split.	78
7.15	These Figures show how the segment was split, and where the initial segmentation points were in the image.	79
7.16	Resulting images from splitting this image: 7.15a	80

7.17 These Figures show how the segment was split, and where the initial segmentation points were in the image. 80

7.18 Resulting images from splitting this image: 7.17a 81

7.19 This segment contains three letters, but it did not get sent to the word splitter. This can be found in Figure 7.7, by looking for segment number 132. 81

7.20 The letter "Shin" has been incorrectly split into two. This can be found in Figure 7.7, by looking for the segments between number 101 and 104. 82

7.21 The results when we draw rectangles over the segmented letters. Dark blue = successful segment and not overlapping any other segments, light blue = successful but overlapping, red means the segment was either too large or too small. 82

7.22 The letters we have manually segmented have a blue rectangle around them. 83

7.23 The letters we have manually segmented have a dark green rectangle around them while the automatically segmented letters have a blue rectangle around them. 83

7.24 Graph showing average loss (bottom graph) and accuracy (top graph) for both training (orange) and validation (blue) through 40 epochs, trained on the DSS dataset 85

7.25 Graph showing average loss (bottom graph) and accuracy (top graph) for both training (orange) and validation (blue) through 40 epochs, trained on the Modified National Institute of Standards and Technology (MNIST) dataset 85

7.26 Shows the confusion matrix for validation in this models last epoch. Rows show the true label, while columns show the predicted label. 87

7.27 Shows the predictions with a confidence score of less than 80%, with a bar graph of the other confidence values to visualize what characters they might have been confused with. 88

7.28 Displays the two characters that the classifier didn't classify correctly, with a bar graph of all other confidence values. 88

7.29 Shows all the predictions for characters with a confidence score about 80% 89

7.30 Comparison for loss and accuracy between different epoch lengths 90

8.1 A figure showing two letters. The green rectangles show how we manually segmented the letters, and the blue rectangles show how the segmenter segmented the letters. DSS image gotten from: [4] . 94

8.2 Figure of calligrapher 95

8.3 Figure of lamed and alef 95

I.1 Time tracking table. These numbers represent how many hours we have worked. 165

I.2 Time tracking graph. These numbers represent how many hours
we have worked. 166

Tables

4.1	Distribution of characters in our dataset	32
5.1	User Patterns	52
7.1	Segmentation results on The Great Isaiah Scroll column 35 using different types of noise removal methods.	72
7.2	Peak Signal-to-noise Ratio (PSNR) results on The Great Isaiah Scroll column 35 using different types of noise removal methods.	72
7.3	Training results our dataset at the final epoch	84
7.4	Training results on the MNIST dataset at 21 epochs	84
7.5	Results as described in this paper [47]	84
7.6	Shows all the precision and recall metrics for all letters	86
7.7	Training results with transfer learning while using the model in Table 7.4 as a base	87

Code Listings

3.1	Example of how to perform median blur.	14
3.2	Example of how to perform bilateral blur.	15
3.3	Example of how to perform non-local means denoising.	15
3.4	Example of Bilateral Blur using OpenCV. In this code example <i>import cv2 as cv</i> has been used to import OpenCV	23
3.5	Example of how a crop is saved in a .txt file using the YOLO format	24
3.6	Example of use of Pytesseract. After reading an image, you can use the Pytesseract method <i>image_to_boxes</i> that reads each letter and saves its coordinates. You then iterate through each letter and draw red rectangles over each of them. To preview the results, you can save the image.	25
3.7	Example of use of PyTorch and the creation of a linear feed-forward network. The initialization function defines the network layers, while the forward function is used in the training algorithm.	26
4.1	How we use OpenCV to perform adaptive histogram equalization on an image.	35
4.2	How we use OpenCV to perform closing on an image.	36
4.3	How we use non-local means denoising.	37
4.4	The way we iterate through the segments provided by the <i>image_to_boxes</i> method, extract each segments coordinates, and crop the image based on those coordinates.	40
4.5	A method for drawing rectangles on an image.	41
4.6	Saves the letters image. If the letter image was saved correctly, it will print the ID of the image that was saved and "True".	42
4.7	Implementation of the feature extraction layers in PyTorch	46
4.8	Implementation of the classification layers in PyTorch	47
4.9	Implementation of the convolutional neural network structure in PyTorch	48
4.10	Method used for freezing weights in the fully connected layers for our model	49
5.1	Simple example of how to create a simple window with PyQt5	55

7.1	Median blur.	71
7.2	Bilateral blur.	72
7.3	Non-local means denoising.	72
H.1	How we convert the labels and coordinates from the YOLO format to images.	163

Acronyms

CNN Convolutional Neural Network. iii, 21, 29, 32, 95

DSS Dead Sea Scrolls. iii, xiii, 1–6, 8–11, 24, 27–29, 31, 34, 36–40, 44, 45, 51, 53, 56, 60–62, 67–71, 73, 74, 85, 93–95, 97, 98

IoU Intersection over Union. 17, 45, 46, 74, 83, 93, 97

MNIST Modified National Institute of Standards and Technology. xiii, xv, 48, 84, 85, 95

OCR Optical Character Recognition. 25, 27, 29, 39, 42, 96

PSNR Peak Signal-to-noise Ratio. xv, 3, 15, 28, 38, 72, 73, 97

RNN Recurrent Neural Network. 29, 96, 98

Glossary

C++ A very popular high level, object oriented programming language. 55

PASCAL VOC Pascal VOC is an XML file. It is a normal format to use when dealing with data for machine learning. 24

XML Stands for eXtensible Markup Language. It is a markup language for storing and transporting data. 24

YOLO YOLO is a data format for storing labeled data. xvii, 24

Chapter 1

Introduction

The Department of Information Security and Communication Technology (IIK) ¹ at NTNU Gjøvik are working on understanding the authorship and date\period of Dead Sea Scrolls (DSS), within the scope of the Lying Pen project² in cooperation with University of Agder³. Ordinarily, researchers working with DSS manually extract letters and words from the DSS, a task that takes a lot of time. Therefore, NTNU researchers and the rest of the Lying Pen of Scribes consortium are interested in finding out if image processing and machine learning techniques can help extract and learn features from the DSS images. Machine learning techniques will help to classify the letters. The use of pre-processing on the images might be required to make content visible. In this project, we will focus on using machine learning to extract the different letters of the DSS, which later on can be used for date and authorship identification. The DSS that we will be working on are the ones written in Hebrew. Figure 1.1 shows an example of a DSS written in Hebrew.



Figure 1.1: The Figure shows an example of one of the DSS written in Hebrew, This one is fragmented and a little bit damaged. Image gotten from: [1]

¹<https://www.ntnu.edu/iik>, visited 03.03.2022

²<https://lyingpen.uia.no/>, visited 02.05.2022

³<https://uia.no/>, visited 03.03.2022

1.1 The Dead Sea Scrolls

The DSS are massive collections of biblical and non-biblical manuscripts that were first found in the Judean desert in 1947. These were mainly written on parchment (made out of animal skin) and papyrus (made from the pith of the papyrus plant). Hebrew, Aramaic, Greek and Latin are some of the languages used in the DSS, while other languages remain unidentified. The DSS consist of nearly intact scrolls and thousands of fragments.[2] The Qumran Cave Scrolls are the most well-known texts among the DSS, but there have also been found other documents and letters, especially papyri that had been hidden in caves by refugees from wars.[3] Figure 1.2 shows an example of The Great Isaiah Scroll column 35.

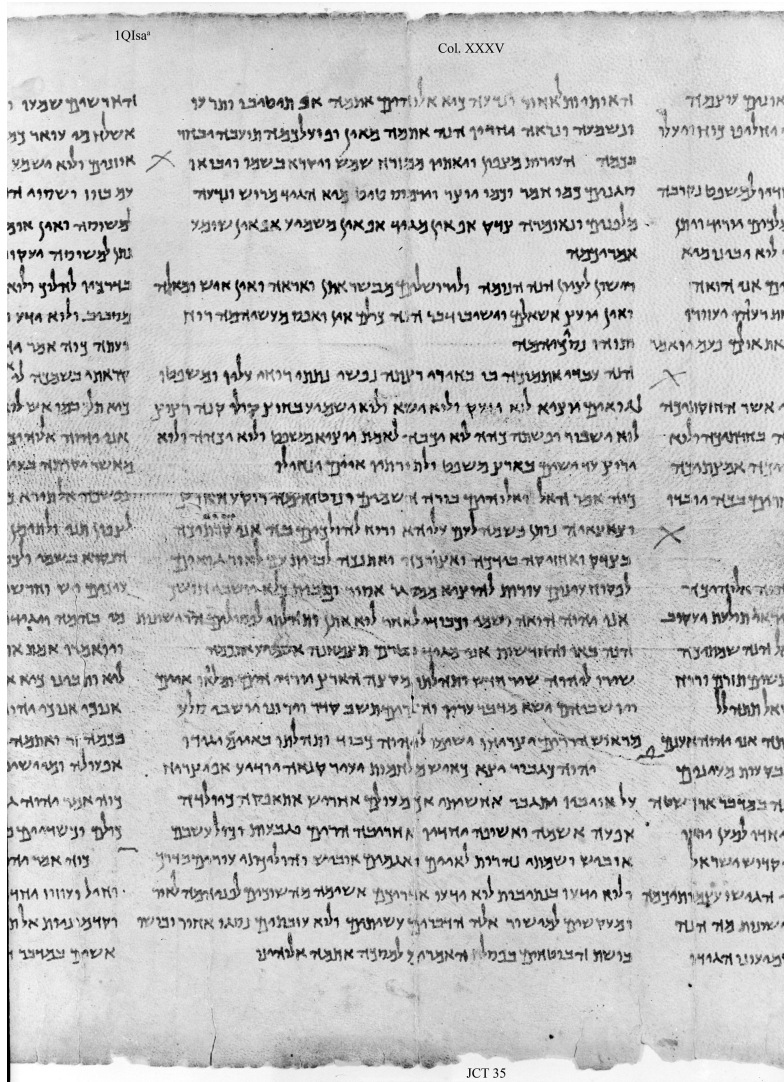


Figure 1.2: The Great Isaiah Scroll column 35. Image gotten from: [4]

1.2 Project Objective

Our task will be to build a system for recognizing handwritten text using machine learning on intact DSS. Our task will be to first do image pre-processing (if that is needed), then segment the letters in the images and finally use machine learning to recognize the letters.[2]

1.3 Goals and Frames

1.3.1 Frames

Time frame

The time frame we have to complete this project is the 10th of January to the 20th of May 2022. Our solution and report need to be finished by this deadline.

Technology frames

- Our code should be able to run on Windows 10 and Windows 11.
- The code we create should be easily modifiable and/or reusable by others.

1.3.2 Result Goals

- We are to create a solution for extracting and learning the features from the DSS images, by classifying the different letters on the DSS images.
- The classification of the letters needs a high accuracy. It should on average have an accuracy of over 0.9 (90%).
- When it comes to the letter segmentation we intend to have an average "Intersection over Union" score of over 0.5 ($IoU > 0.5$).[5]
- We assume that the time it takes to segment and classify letters on a single column of the DSS should not take more than a couple of minutes, preferably not more than 2 minutes. This number is based on early testing with Pytesseract (see Section 3.3.4 for more information). The DSS image we used to test Pytesseract was the Great Isaiah Scroll column 35, which has a size of 1999x2760 pixels.
- When it comes to the pre-processing part we intend our denoising methods to have a Peak Signal-to-noise Ratio (PSNR) score of over 30 dB.[6]
- The users should be able to extract the features on the DSS through an easy and intuitive user interface.

1.3.3 Effect Goals

- Our employers are currently extracting letters manually. Our solution should automatically segment the letters which would greatly reduce the time it takes for our employers to extract letters from the DSS.

- Our work should help free up work capacity for our employers, by making it easier for them to extract information from the DSS.
- Our solution should be modifiable and expandable by others outside our group. Meaning that others outside our group should be able to expand upon our solution.

1.3.4 Problem Delimitation

We as developers are just responsible for working on the extraction of letters and the classification of them, not responsible for actually understanding the meaning of what is written there. The solution we provide is only going to be used by the Department of Information Security and Communication Technology at NTNU Gjøvik and/or other partners involved in the Lying Pen project.

1.4 Group Background

The group consists of three students from the Computer Science programme at the Norwegian University of Technology and Science (NTNU) in Gjøvik⁴. All the students in the group have taken the same mandatory courses in the study plan. The most relevant of all of these is the Computer Vision (IDATG2206)⁵ course, where we among other things went through image processing and machine learning techniques. Another relevant course is Introduction to User-Centered Design (IDG1262)⁶, where we learnt how to design a user friendly interface. The only difference in our competence comes from the different optional courses we have taken.

1.5 Organization

The organizing of roles was a part of our project planning. In the project plan, we specified roles and main responsibility areas. For more information about that see Appendix D.

1.5.1 Roles and Responsibilities

Our Roles

A. Leadership – Project leader (Erik Dale)

Responsibilities: Responsible for making sure that group members follow the rules and that each member is occupied with a task. The leader is the chairman in meetings. If all the group members disagree, the leader has the final word.

⁴<https://www.ntnu.no/gjovik>, visited 03.03.2022

⁵<https://www.ntnu.edu/studies/courses/IDATG2206#tab=omEmnet>, visited: 09.03.2022

⁶<https://www.ntnu.edu/studies/courses/IDG1362#tab=omEmnet>

B. Communication responsible (Yeshi Jampel Pursley)

Responsibilities: Responsible for communication in and outside of the group such as scheduling meetings with our supervisors and employers.

C. Archive/ Document responsible (Håvard Fjørkenstad)

Responsibilities: Responsible for making sure that the meetings are documented. He has a general overview of all of our documentation.

Main Responsibilities**A. Image Pre-Processing (Erik Dale)**

Responsible for coding the necessary image enhancement techniques to ensure a more effective image segmentation.

B. Image Segmentation (Yeshi Jampel Pursley)

Responsible for the code and techniques for segmenting the DSS images into letters to be used as input for the neural network.

C. Machine Learning (Håvard Fjørkenstad)

Responsible for the code behind the neural network model and training, which are to be used for classifying letters.

Other Roles**A. Employers**

Our employers are Sule Yildirim Yayilgan and Tabita Anggraini Meilita Lumban Tobing who works for the Department of Information Security and Communication Technology (IIK)⁷ at NTNU in Gjøvik. Their job is to state the requirements of the project and also to help and give us guidance when it is needed.

B. Supervisors

Our supervisors are Marius Pedersen and Aditya Suneel Sole whom both work at the Department of Computer Science⁸ at NTNU in Gjøvik. Their job is to give us counseling throughout our work on the project.

⁷<https://www.ntnu.edu/iik>, visited 03.03.2022

⁸<https://www.ntnu.edu/idi>, visited 10.05.2022

1.6 Structure of the Report

The report is divided into 9 chapters. They should be read sequentially from chapter to chapter. Following is a short description of the content in each chapter.

Chapter 1 - Introduction

Includes an introduction of the project and the project statement. Also presents the group's goals, organization and background.

Chapter 2 - Development Process

Walkthrough of development plan and how that plan was followed by the group throughout the project.

Chapter 3 - Background

Breakdown of the DSS, theories and technologies this project is based upon. Contains descriptions of DSS theories and technologies written about in the methodology chapter. It also contains state of the art mostly within the field of handwriting recognition that is relevant for our project.

Chapter 4 - Methodology

This chapter explains the methods we used in our research and development and why we chose those methods.

Chapter 5 - User Interface

In this chapter, we go through how the user interface was planned and made. We also show the final solution and discuss possible future improvements to it.

Chapter 6 - Quality Assurance

Walkthrough of the different tools used to take care of code quality, and how the group used those tools.

Chapter 7 - Results

Breakdown of all results gotten from our methods described in the methodology chapter.

Chapter 8 - Discussion

Discussion of the results of our research, development and project process.

Chapter 9 - Conclusion

Concludes if we through our research and development have reached our stated goals.

Chapter 2

Development Process

In the project plan (see Appendix D) we went through what kind of software development model we will use in this project. This chapter further builds upon this plan and explains how it was followed throughout the project process.

2.1 Development Model

The software development model we chose was Agile. Within the agile framework, we chose to use Kanban. You can find our arguments for why we chose this development model in our project plan in Appendix D. A Kanban board visually depicts tasks at different stages in the development process. Each card contains a description of the task. Each card must be placed in a column that describes the status of the task. On GitHub, we created cards that are connected to their own GitHub Issue where more detailed information can be found. The Issue on GitHub is a tracking feature where we can track our work, bugs, and ideas. This is done by commenting in the Issues.

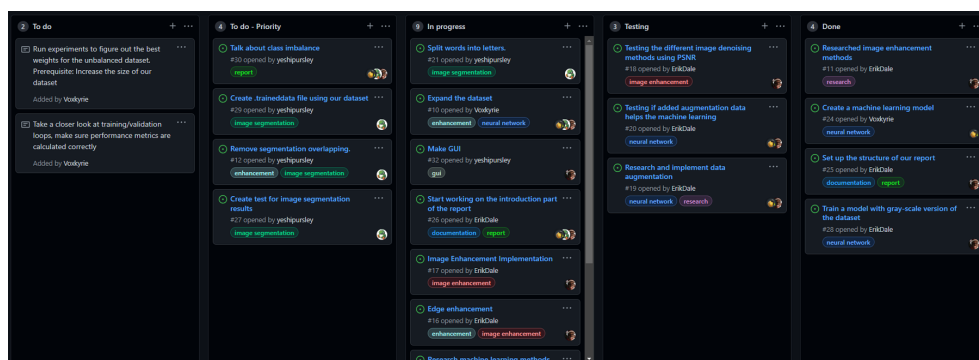


Figure 2.1: Our Kanban board on GitHub.

2.1.1 How We Divided Our Work

The project is divided into five phases. The deadline for each phase can be viewed on our Gantt-diagram (see Figure 2.2). The phases cover tasks that are obligatory to our bachelor thesis and the tasks the employers want us to work on. The phases provided us and our employers a simple overview of our goals for the project.

- **First phase**

The goal of our first phase was to finish the project plan, confidentiality agreement, standard agreement and collaboration agreement.

- **Second Phase**

The goal of our second phase was to finish writing the code for image enhancement and image segmentation, develop a classifier, and write code that is able to classify test case letters. These tasks would provide the core functionality that we would need to work on the DSS. To develop our classifier we helped Tabita Tobing prepare a dataset. We worked more on Image Enhancement and Image Segmentation later to improve results, but when we finished this phase those functionalities would provide good enough results for the next phase.

- **Third Phase**

In our third phase, we expanded on our work from the previous phase. Our goals were to write code that can classify letters provided by our image segmentation and to write code that works for DSS that need image enhancement.

- **Fourth Phase**

In our fourth phase, we started working on the GUI. We also tried implementing transfer learning and experimented with augmentation to expand our dataset.

- **Fifth Phase**

In our last phase, we finished the report. We worked on the report during the entire project, but after we finished our development our main focus was on the report to get it finished before the deadline 20th of May.

2.1.2 Gantt-Diagram

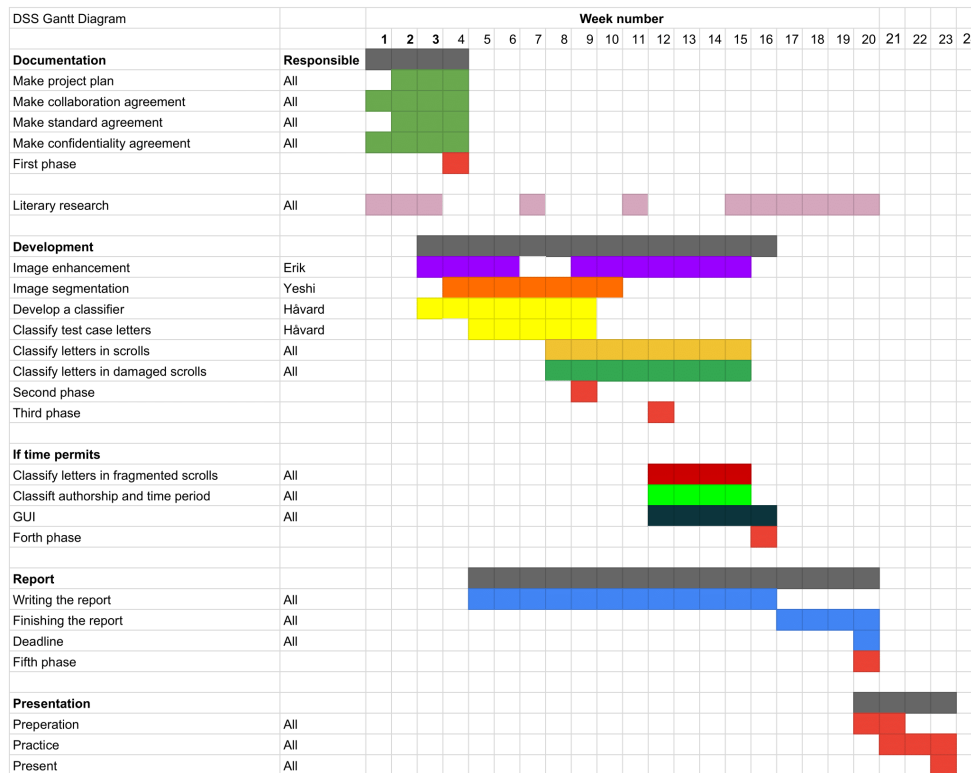


Figure 2.2: Our Gantt-Diagram

2.2 Meetings

We had different types of meetings throughout the project. These are internal meetings within the group, meetings with the supervisors, the employers, and with experts in the DSS. This ensured that everyone in the group was on the same page and that the supervisors and the employers could keep track of the status of our work and development. They could also give us feedback on our work. Our internal meetings were held on Discord¹, while the other meetings were arranged on Teams².

2.2.1 Meetings With Employers

Every Thursday at 12:00 we had a meeting with our employers Sule Yildirim Yayilgan and Tabita Anggraini Meilita Lumban Tobing. In these meetings we would describe our progress, the issues we were facing that our employers could help with, and ask questions related to the DSS.

¹<https://discord.com/>, visited 22.04.2022

²<https://www.microsoft.com/nb-no/microsoft-teams/log-in>, visited 22.04.2022

2.2.2 Meetings With Supervisors

Every Thursday at 14:00 we had a meeting with our supervisors Marius Pedersen and Aditya Suneel Sole. We would describe our progress, talk about the issues we were facing that our supervisors could help with, and ask questions related to Computer Engineering. We would often ask questions about Computer Vision and our bachelor thesis report.

2.2.3 Internal Meetings

Every Thursday at 11:00 we had an internal group meeting where we went through the questions we were going to ask the employers and supervisors in our meetings with them. After every meeting with them, we would have an internal meeting where we would discuss the takeaways and what we needed to work on.

Every Monday at 12:00 our group would have an internal meeting. We would discuss our progress since the last meeting with our employers and supervisors. We would also discuss what we could have ready for the next meeting on Thursday with our employers.

2.2.4 Meetings With Dead Sea Scroll Expert

We had the opportunity to have several meetings with Torleif Elgvin who is an expert on the DSS. In these meetings, we learned about the history of the DSS, the letters and how they are written differently in various DSS.

Chapter 3

Background

This chapter is divided into four sections. The first Section (Section 3.1) is about how the DSS are structured. After that, there is a theory Section (Section 3.2) that we have written to give a theoretical background for readers to understand our methodology used throughout the project. The theory Section contains definitions of key terms related to what we have done. The third section of the chapter (Section 3.3) called technologies, goes through the key technologies we have used throughout the project work. State of the art (Section 3.4) is the last section of the chapter where we go through similar work done before by various research groups.

3.1 Dead Sea Scrolls

In this section we will go through the structure of the DSS, especially the Great Isaiah Scroll. Section 3.1.1 contains information about the structure of the DSS and Section 3.1.2 goes through the letters used in the DSS. See Section 1.1 for more on the history of the DSS, and see Figure 1.2 for an example of a column from the Great Isaiah Scroll.

3.1.1 Scrolls and Columns

A DSS is divided into multiple columns. *1QIsa^a*, also called The Great Isaiah Scroll, contains 54 columns. There exists images of each column in the scroll. [4] The columns are read from right to left. *1QIsa^a* is written by two different scribes. The first scribe wrote

א	א	alef
ב	ב	bet
ג	ג	gimel
ד	ד	dalet
ה	ה	he
ו	ו	vav
ז	ז	zayin
ח	ח	het
ט	ט	tet
י	י	yod
כ	כ	kaf
ל	ל	lamed
מ	מ	mem
נ	נ	nun
ס	ס	samekh
ע	ע	ayin
פ	פ	pe
צ	צ	tsadi
ק	ק	qof
ר	ר	resh
ש	ש	shin
ת	ת	tav

Figure 3.1: The Figure shows all the modern Hebrew letters in the first column (from the left), the equivalent Hebrew letters from the DSS period in the second column and the romanization in the third [2]

columns 1-27 and the second scribe wrote columns 28-54. You can read more about scribes here: [7].

3.1.2 Letters

The Hebrew alphabet has 22 letters. Five of these letters have two different forms. When they are at the end of a word they are written differently compared to when they are at the beginning or in the middle of a word. When these five letters are at the end of a word they are called final letters, while they are called non-final letters when appearing at the beginning or in the middle of a word. The letters with two different forms are "Mem", "Nun", "Tsadi", "Pe" and "Kaf". Ligatures are two letters that are bound together, called bases and companions. Bases are on the right side of the ligature and companions are on the left side. In Hebrew, there are five letters that can be bases and six that can be companions.[2]

3.2 Theory

3.2.1 Computer Vision

Computer vision is a field where the goal is to emulate human vision in computers. This includes learning and being able to make inferences and take actions based on visual inputs.[8] Computer vision can be used for a lot of different tasks. It is used in newer cars where it has an important role when it comes to improving the safety system of a car. It can assist drivers by recognizing dangerous situations and act to prevent them from such situations. Another example where computer vision can be used is in industry to find faulty products in an assembly line.[9] Using digital images or videos and sometimes deep learning models, computers can accurately identify and classify objects. Common methods to achieve computer vision are image enhancement, image segmentation and machine learning, which all are subjects we will cover later in this chapter of the report in Sections 3.2.2, 3.2.4 and 3.2.6 respectively.

3.2.2 Image Enhancement

Image enhancement is one of the methods that help achieve computer vision. It is about improving the quality and the information content of images before further processing.[10] Often it includes methods that improve contrast, remove noise, restore blurred images, etc. It can also be used to solve problems dealing with machine perception and is among other things used to help computers extract features from images. Other areas where computer vision is used can be automatic character recognition, automatic processing of fingerprints, screening of X-rays and blood samples etc.[8]

Contrast Improvement

A very common method when it comes to image enhancement, as mentioned above, is contrast improvement. One method that deals with this is adaptive histogram equalization. To understand adaptive histogram equalization we need to understand histogram equalization. Some images have their pixel values confined to some specific range of values. A dark image will for example have a lot of low pixel values. The goal of histogram equalization is to spread the pixels as equally as possible across all the different values. By doing this, it improves the contrast of the image. Adaptive histogram equalization works almost the same way, but the image is divided into small blocks. In each of these small blocks, often called "tiles", histogram equalization is performed, hereby the name "adaptive". The histogram equalization adapts to the different "tiles" of the image. [11]

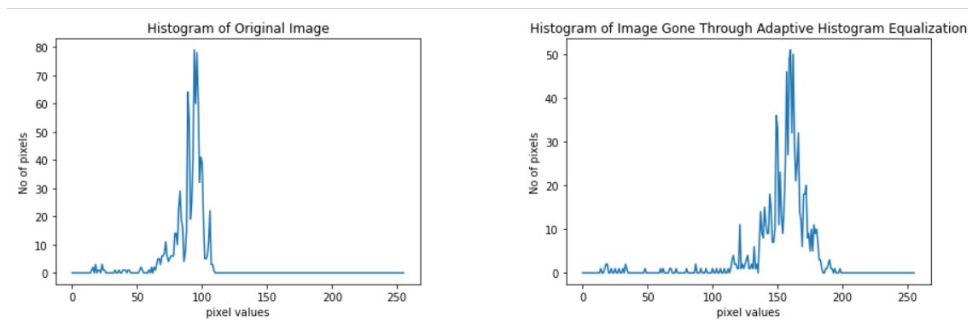


Figure 3.2: Histogram of original image (left) and histogram of that same image after adaptive histogram equalization.

Noise Removal

Noise is very common in images and can occur when the image is captured, during transmission, when the image goes through other image processing methods such as binarization, etc. Noise can be caused by reasons such as insufficient light levels, interference in the transmission channel and imaging sensors being affected by environmental conditions.[12] There are also different kinds of noise that can appear in an image. Some of the different kinds of noise include Gaussian noise, Salt-and-pepper noise and exponential noise. One of the most common ways to remove noise from images is to use filters. Different filters are good at removing different types of noise. The Gaussian filter works well on Gaussian noise for example. If the noise in the image changes throughout it, an adaptive filter like the Wiener filter can be used. This filter adapts to the local image variance.[12] The next four sections describe four different ways of removing noise, Section 3.2.2(Morphological transformations), Section 3.2.2(Median blur), Section 3.2.2(Bilateral blur) and Section 3.2.2(Non-local means denoising) respectively.

Morphological Transformations

Morphological transformations are different kinds of simple operations normally performed on binary images. The method takes as input a structuring element or kernel which decides the nature of the operation. [13] Four of these morphological transformations will be explained further here.

1. Erosion

Erosion shrinks the boundaries of foreground objects of binary images.

$$g = f \ominus s \quad (3.1)$$

As it can be seen from Formula 3.1, a new binary image g is produced by using kernel s on input image f . Ones will be put in all locations (x,y) of the kernel's origin at which s fits the input image f . This means that $g(x,y)=1$ if s hits f and 0 otherwise.[14]

2. Dilation

Dilation is the opposite of erosion. It thickens foreground objects in a binary image. The extent of the thickening is determined by the kernel s .

$$g = f \oplus s \quad (3.2)$$

A new binary image g is formed by the dilation where $g(x,y)=1$ if s hits f and 0 otherwise.[14]

3. Closing

Closing is when dilation followed by erosion is performed on an image. It is useful for closing small holes inside the foreground objects.[14]

$$f \bullet s = (f \oplus s) \ominus s \quad (3.3)$$

4. Opening

Opening is when erosion of an image is followed by dilation. This transformation tends to open up a gap between objects connected by a thin bridge of pixels. It can also be used to remove noise in the background of the image.[14]

$$f \circ s = (f \ominus s) \oplus s \quad (3.4)$$

Median Blur

Median blur is not a very complicated method. It takes as parameters the image and the kernel size. Using for example `cv.medianBlur(img, 5)`, as seen in Code listing 3.1, the kernel size will be 5x5 pixels. The kernel size needs to be an odd number as the kernel needs to have a single central pixel. The method takes the median of all the pixels under the kernel and replaces the central element with this median value. Median blur is highly effective against Salt-and-pepper noise [15].

```
1 median = cv.medianBlur(img,5)
```

Code listing 3.1: Example of how to perform median blur.

Bilateral Blur

Not only does bilateral blur remove noise, but it also keeps the edges sharp. The operation is slower compared to other filters like median blur. Bilateral filtering takes a Gaussian filter in image space and one more Gaussian filter which is a function of pixel intensity difference between the central pixel and the surrounding ones. The job of the first Gaussian filter is to make sure that only nearby pixels are considered for blurring. The second one is responsible for making sure that only those pixels with similar intensities to the central pixel are considered for blurring.[15] Code listing 3.2 shows an example of how to perform bilateral blur using OpenCV.

```
1 bilateral_blur = cv.bilateralFilter(img,9,150,150)
```

Code listing 3.2: Example of how to perform bilateral blur.

Non-local Means Denoising

What non-local means denoising does is that it replaces the color of a pixel with an average of the colors of similar pixels. It is not certain that the most similar pixels are close at all, so the method scans a vast portion of the image and tries to find all the pixels that resemble the pixel you want to denoise.[16] OpenCV has a method called *fastNlMeansDenoising* as shown in Code listing 3.3.

```
1 means_denoised = cv.fastNlMeansDenoising(src=img,h=60.0, templateWindowSize=7,
    searchWindowSize=21)
```

Code listing 3.3: Example of how to perform non-local means denoising.

3.2.3 Testing of Image Enhancement

Pixel to Noise Ratio (PSNR)

How well image enhancement methods work can often be subjective. One person might state that median blur is best at removing noise from an image, while another might state that bilateral blur is best. PSNR makes it possible to establish quantitative measures to compare different image enhancement methods. PSNR expresses the ratio between the maximum power (maximum possible value) of a signal and the power of distorting noise that affects the quality of its representation. This ratio is often used as a measurement of quality between the original image and the output image of an image enhancement method.

$$PSNR = 20 * \log_{10} \left(\frac{MAX_f}{\sqrt{MSE}} \right) \quad (3.5)$$

$$MSE = \frac{1}{mn} \sum_0^{m-1} \sum_0^{n-1} \|f(i, j) - g(i, j)\|^2 \quad (3.6)$$

Equation 3.5 represents the peak signal-to-noise equation and equation 3.6 represents the mean squared error equation [6] where MAX represents maximum power, MSE represents minimum square error and m and n are the image row and column sizes respectively.

3.2.4 Image Segmentation

According to this source, [17], "Image segmentation is a commonly used technique in digital image processing and analysis to partition an image into multiple parts or regions, often based on the characteristics of the pixels, such as pixel values. Image segmentation could involve separating foreground from background, or clustering regions of pixels based on similarities in color or shape." Different image segmentation algorithms and techniques have been developed like clustering, semantic segmentation, and thresholding. These can be used for application areas like medical imaging, automated driving, video surveillance, and computer vision. [17]

Thresholding

Thresholding is a method of segmenting objects based on a threshold value. Global thresholding is when the same threshold value is used on the entire image. If the threshold value changes during the thresholding of an image it's called adaptive thresholding. More information about adaptive thresholding can be read here [8]. With thresholding, you can convert an image into a binary image where the foreground is one value and the background is another value. This process is called binarization.

Otsu's method automatically calculates the threshold value. The method can be summarized by the equation 3.7. The purpose of the Otsu method is to separate the image into two clusters with a threshold, which is found by minimizing the weighted variance of the two classes.

$$\sigma^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t) \quad (3.7)$$

where $w_0(t)$ and $w_1(t)$ are the probabilities of the two classes which is divided by the threshold t . [18]

OpenCV has a method for adaptive thresholding:

```
adaptiveThreshold(src, dst, maxValue, adaptiveMethod, thresholdType, blockSize, C)
```

The *adaptiveMethod* parameter can either be *ADAPTIVE_THRESH_MEAN_C*, which, according to this source [19], means the "threshold value is the mean of neighborhood area", or *ADAPTIVE_THRESH_GAUSSIAN_C*, which means the "threshold value is the weighted sum of neighborhood values where weights are a Gaussian window." [19]

Skeletonization

According to this source, [20], "Skeletonization is a process for reducing foreground regions in a binary image to a skeletal remnant that largely preserves the extent and connectivity of the original region while throwing away most of the original foreground pixels." This process uses the morphological process "thinning" to remove certain foreground pixels to create a line that is one pixel wide or high. The Figure 3.3 shows an example of a skeletonized image.



Figure 3.3: An example of a skeletonized image.

Canny edge detector

Canny edge detection is an edge detection algorithm. It starts off by removing noise, finding the intensity gradient of the image, applying a non-maximum suppression which results in a binary image with "thin edges", and lastly applying hysteresis thresholding which returns only the strong edges in the image. [21]

3.2.5 Testing of image segmentation

Intersection Over Union

According to this source, [22], Intersection over Union (IoU) "is a term used to describe the extent of overlap of two boxes. The greater the region of overlap, the greater the IOU." IoU is used in image segmentation to for example compare a box that represents the ground truth and another box which represents a prediction of where the box should be. The prediction should be as close to the ground truth as possible which means we are aiming for the highest IoU score.



Figure 3.4: Figure that shows an example of a poor, good and excellent IoU score. Source: [23].

3.2.6 Machine Learning

Machine learning is a branch within Computer Science and Artificial Intelligence (AI) that focuses on imitating how humans learn by using data and algorithms. [24] Neural networks are based on the structure of the human brain and mimic how biological neurons signal each other, which allows the computer to recognize features and patterns. [25]

Neural Networks

A neural network is a set of machine learning algorithms that takes an input, passes it along multiple layers of neurons and outputs a prediction based on the final sum. It is composed of multiple layers of neurons, which in its most basic form consists of an input layer, one or more hidden layers, then an output layer, as seen in Figure 3.5. The neurons in each layer are composed of a group of weighted inputs into those neurons, a bias and an activation function to introduce non-linearity into the network. [25]

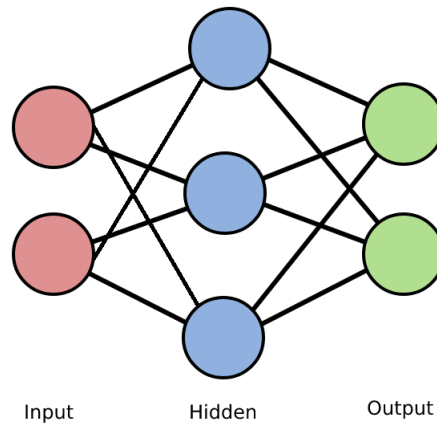


Figure 3.5: Visual example of a neural network structure. Here, each circular node represents an artificial neuron and the lines represent a connection from the output of one artificial neuron to the input of another.

The output of a neuron, like the one in Figure 3.6, can be formulated as

$$\hat{y} = \sigma(x \cdot w + bias) \tag{3.8}$$

where x is an array of the node inputs $x=[x_1,x_2,x_3,...]$ and w is an array of the node weights $w=[w_1,w_2,w_3,...]$. σ is the activation function 3.9, and therefore the node output varies based on which activation function is used. For instance, Rectified Linear Units (ReLU) is a popular activation function used in the hidden layers because of its good performance. For classification problems, Sigmoid or Softmax activation functions must be used in the final layer.

$$\sigma(x) = \begin{cases} z & z > 0 \\ 1 & z \leq 0 \end{cases} \quad \sigma(x) = \frac{1}{1 + e^{-z}} \tag{3.9}$$

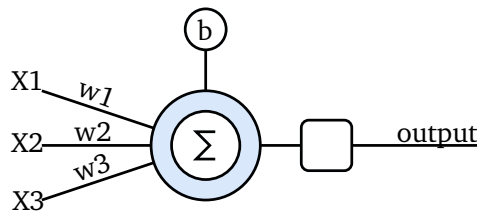


Figure 3.6: Visual example of how a neuron in a network is structured. X are the outputs from previous neurons, while w are the weights. b represents the bias that gets added to the weighted sum of inputs, and sigma is the activation function before the neuron outputs a result.

Neural Networks learn and improve their performance by adjusting these weights using back-propagation, where an error is calculated based on the difference

between what the models predict and the inputs' ground truth with the help of a loss (also known as cost) function. [26] The neural network model is trained over a set number of epochs, where the entire dataset is passed over each iteration.

Machine Learning Metrics

To evaluate a machine learning model, a set of different metrics can be used to get an indication of how well the model is doing. These can be divided into two groups. Group one includes accuracy and loss, while group two includes precision and recall. A neural network's accuracy is the number of correct predictions over the batch size, while the loss value depends on the loss function used, but is generally an indication of how bad a single prediction was. Precision and recall is a measure of quality and quantity, where precision is how many of the predicted characters are relevant and recall is how many relevant characters are predicted. Related to these metrics is the confusion matrix as seen in Figure 3.7, which in a multi-label classification problem is a table between true and predicted classes, with the prediction distributed in the correct rows and columns.

		Predicted		
		Dog	Cat	Horse
True	Dog	10	7	4
	Cat	1	5	6
	Horse	5	3	9

Figure 3.7: Visual example of a confusion matrix. Green are correctly labeled, while red are incorrectly labeled. In this case the Dog precision would be true positives divided by the dog column, while recall would be the true positives divided by the dog row

Convolution

Convolution is an operation that uses a kernel to transform the input image into a set of feature maps, depending on what type of kernels are used. The feature maps are representations of what the kernel has learned about the image. The

operation involves sliding a kernel across the image, where the sum of the kernel at each position is a pixel in the resulting feature map. An example of this can be seen in Figure 3.8. Without padding on the image, the feature map will be smaller than the original, because the kernel cannot perform its operation outside valid elements e.g. pixel values. [27]

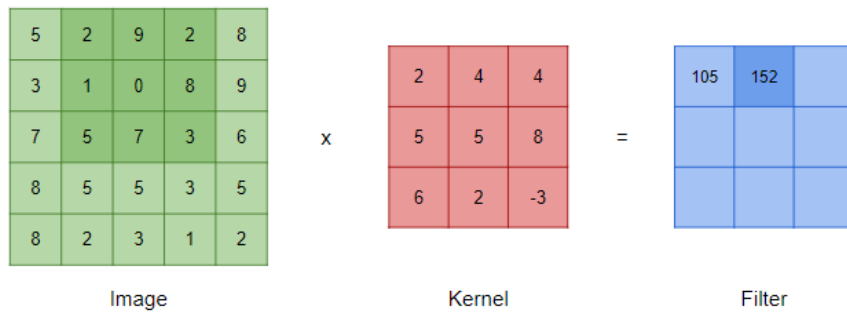


Figure 3.8: Visual example of how a convolution operation works.

Pooling

Pooling is an operation used to compress each feature map, where every patch of the pooling size is reduced to one pixel, as seen in Figure 3.9. The value of this pixel is determined by the type of pooling used. Average Pooling will take the average value within the patch, while Max Pooling will take the maximum value. [27]

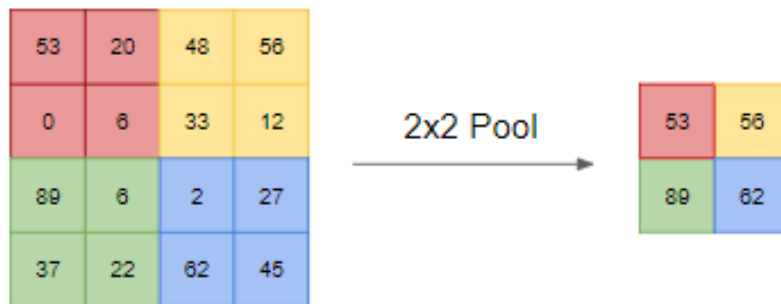


Figure 3.9: Visual example of how a max pooling operation works.

Convolutional Neural Networks

Convolutional Neural Network (CNN) combine the traditional neural network structure with the convolution and pooling layers, in addition to the fully-connected

layer. The first layer in this model is a convolution layer, which extracts simple features and is followed by either additional convolutional layers or a pooling layer. These two-layer types can also be stringed together. The final layer in this model is the fully-connected layer, which performs the classification task. As you get deeper into the convolutional layers, the complexity of the model increases and the layers extract more complex features. [28]

Augmentation

Image augmentation is a way of increasing the size of your dataset through various operations which modify the image in some way, such as rotation, zooming, blurring, shearing, etc. It is a way of not only getting a bigger dataset but also a more diverse one. When it comes to images of handwriting operations such as rotation and morphological methods like erosion and dilation are common. This makes a lot of sense as handwriting often is a bit skewed and the letters can sometimes be of different thicknesses. In Figure 3.10 we can see an example of alef being eroded.

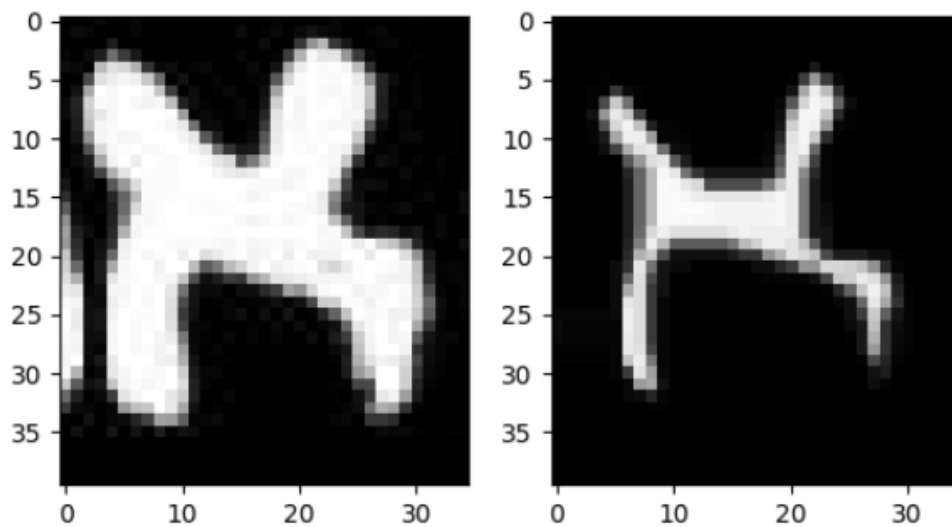


Figure 3.10: The left image is the original Alef while the right one has gone through erosion.

3.3 Technologies

3.3.1 Python

Python is an open-source, object-oriented, interactive and interpreted programming language. It was created by Guido van Rossum in 1989 and is today managed

by The Python Software Foundation¹. [9] Python offers the use of exceptions, modules, dynamic typing, classes and very high-level dynamic data types. Python can be run on almost all operating systems and it is a universal language found in a variety of different applications. Python also includes a lot of third-party modules that are available in the Python Package Index (PyPI) [29]

3.3.2 OpenCV

OpenCV² is an open-source, highly optimized library with focus on real-time applications. It has cross-platform support and its C++, Python and Java interfaces support Linux, MacOS, Windows, iOS, and Android. [30] OpenCV is a tool for performing computer vision tasks and for image processing. It can be used for purposes such as noise removal and image segmentation. Code listing 3.4 shows an example of how OpenCV can be used in Python.

```
1 import imageio as img
2 from PIL import Image
3 import cv2 as cv
4
5 # Read the image using imageio
6 img = img.imread("./image.jpg")
7
8 # Converts the image to gray scale if it isn't already
9 if len(img.shape) == 3:
10     gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
11 else:
12     gray_img = img
13
14 # Adaptive binarization of the image using openCV (cv2) module
15 binarize_img = cv.adaptiveThreshold(src=gray_img, maxValue=255,
16 adaptiveMethod=cv.ADAPTIVE_THRESH_GAUSSIAN_C, thresholdType=cv.THRESH_BINARY,
17 blockSize=39, C=15)
18
19 # Using bilateral blur, which is highly effective at noise removal while
20 # preserving edges.
21 bilateral_blur = cv.bilateralFilter(binarize_img, 9, 150, 150)
22
23 # Saving the binarized and blurred image
24 im = Image.fromarray(bilateral_blur)
25 im.save("/blured_image.jpg")
```

Code listing 3.4: Example of Bilateral Blur using OpenCV. In this code example *import cv2 as cv* has been used to import OpenCV

¹<https://www.python.org/psf/>, visited 15.02.2022

²<https://opencv.org/>, visited 15.02.2022

3.3.3 LabelImg

LabelImg³ is an open-source graphical image annotation tool and it is used for labeling object bounding boxes in images.[31] In LabelImg you can load a folder of images, crop areas of the images and then label those images. An example of LabelImg's user interface can be viewed in Figure 3.11. If you are using LabelImg for making a data set for machine learning, those labels would be your classes. The crops are saved as XML files in PASCAL VOC format, YOLO or CreateML formats. In the YOLO format, the crops are saved in a .txt file where each line represents one crop.

```
1 2 0.568159 0.126410 0.013581 0.009822
```

Code listing 3.5: Example of how a crop is saved in a .txt file using the YOLO format

Listing 3.5 is an example of how the class and coordinates of the crops are stored using YOLO format. The first number in the line indicates the label (class) of the crop. The number corresponds to the line number in a list of classes. The next four numbers indicate the positioning and width and height of the cropped out part of the image. Using code that can be found in Appendix H, we can crop out the images and from the DSS image we used LabelImg on.



Figure 3.11: Example of LabelImg's user interface.[31]

³<https://github.com/tzutalin/labelImg>, visited 17.02.2022

3.3.4 Tesseract

Tesseract is an open source Optical Character Recognition (OCR) engine. It can be used via a command line or via API.[32] Pytesseract, Python-tesseract, is an OCR tool for python. Pytesseract is a wrapper for Google's Tesseract-OCR Engine.⁴ This software can be used to read and recognize text on images. Code listing 3.6 shows an example of Pytesseract being used in Python.

```

1 import pytesseract
2
3 import cv2
4
5 pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.
  exe'
6
7 img = cv2.imread('example.png')
8
9 # Detects characters and draws red rectangles over them
10 h_img, w_img = img.shape
11 boxes = pytesseract.image_to_boxes(img, lang="heb")
12 for b in boxes.splitlines():
13     b = b.split(' ')
14     x, y, w, h = int(b[1]), int(b[2]), int(b[3]), int(b[4])
15     cv2.rectangle(img, (x, h_img - y), (w, h_img - h), (0, 0, 255), 1)
16
17 print("Saving the img with segments was successful:", cv2.imwrite('img_seg.png',
  img))

```

Code listing 3.6: Example of use of Pytesseract. After reading an image, you can use the Pytesseract method *image_to_boxes* that reads each letter and saves its coordinates. You then iterate through each letter and draw red rectangles over each of them. To preview the results, you can save the image.

3.3.5 QT Box Editor

As defined in [33], *QT Box Editor*⁵ is a multi-platform visual editor for *tesseract-ocr* box files (used for OCR training) based on QT4 library. QT Box Editor is used to manually edit box files. A box file is a text file that contains information about each character and the coordinates of its bounding box in an image used for OCR training. The QT Box Editor user interface can be viewed in Figure 3.12.

⁴<https://pypi.org/project/pytesseract/>, visited 04.03.2022

⁵<https://zdenop.github.io/qt-box-editor/>, visited 29.03.2022

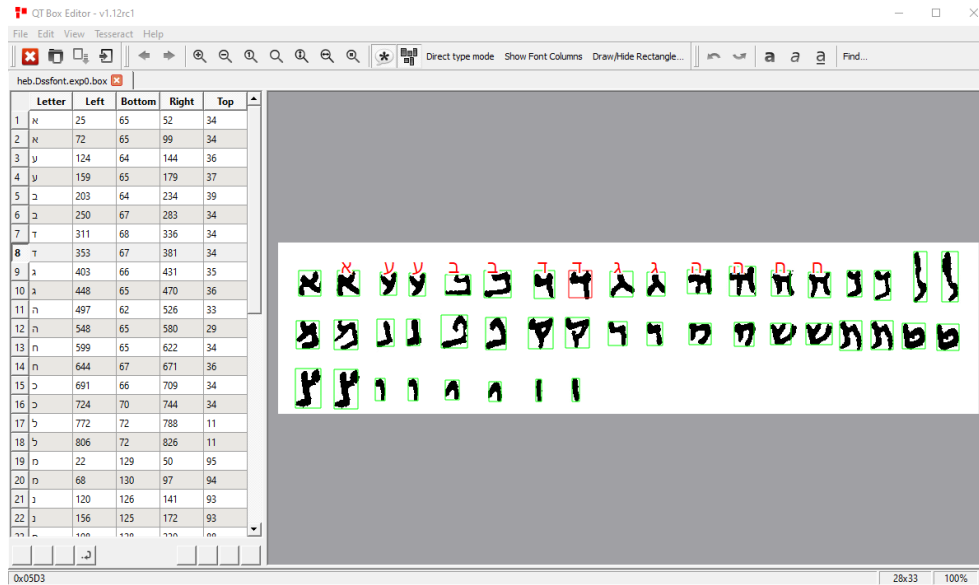


Figure 3.12: Example of QT Box Editor's user interface.

3.3.6 PyTorch

PyTorch⁶ is an open-source deep learning framework, mainly developed by the Meta Platforms (Facebook) AI Research team [34]. It is an optimized tensor library, built to utilize the GPU for quicker training. The framework is deeply integrated into python, making it fast and easy to both read and write [35].

```

1 import torch.nn as nn
2 import torch.nn.functional as F
3
4 class Linear(nn.Module):
5     def __init__(self):
6         super(Linear, self).__init__()
7         self.flatten = nn.Flatten()
8         self.fc1 = nn.Linear(28*28, 256)
9         self.fc2 = nn.Linear(256, 128)
10        self.fc3 = nn.Linear(128, 10)
11
12    def forward(self, x):
13        x = self.flatten(x)
14        x = F.relu(self.fc1(x))
15        x = F.relu(self.fc2(x))
16        x = self.fc3(x)
17        return x

```

Code listing 3.7: Example of use of PyTorch and the creation of a linear feed-forward network. The initialization function defines the network layers, while the forward function is used in the training algorithm.

⁶<https://pytorch.org/>, visited 15.02.2022

3.4 State of The Art

What we go through in this section is the state of the art of image enhancement, image segmentation and machine learning mostly within the field of handwriting or digital text recognition. While what we have gone through at the beginning of this chapter is more general theories and technologies within the field of computer vision, this Section contains more concrete and recent research made by computer vision researchers.

3.4.1 Image Enhancement

When it comes to OCR there are a lot of different image pre-processing techniques that can be used to improve its accuracy. Techniques that are used depend on a lot of different factors. They can depend on what kind of text you are going to recognize, whether it is computer-generated or handwritten. They can also depend on the kind of background the text has. Is the background clean and white or does it have smudges and cracks? Another big factor is how the image of the text has been taken. Has it been taken with a phone camera with low resolution at a slanted angle or has it been taken with a high-resolution camera straight from above? In [14], the authors had to use skewing and sharpening tools to deskew images and get a better OCR accuracy. This is not something we have to worry about as all the DSS images are taken at a straight angle with a high-resolution camera. In [14], the authors also used the morphological transformation opening as they found out that their binarization methods were sometimes turning the characters into blobs. In [36], the authors use a local brightness and contrast adjustment method to handle lighting variations, and they use the Un-sharp Masking method to sharpen the images so they are able to easier extract the text in them.

In [37], Dhali et al. used BiNet, which is a deep-learning-based method designed to binarize the DSS images. This method seems to create very little Salt-and-pepper noise in the background as seen in Figure 3.13, so they do not need to use any image enhancement methods. In [38], Reynolds et al. use dilation where only the outlines of letters may remain after binarization.

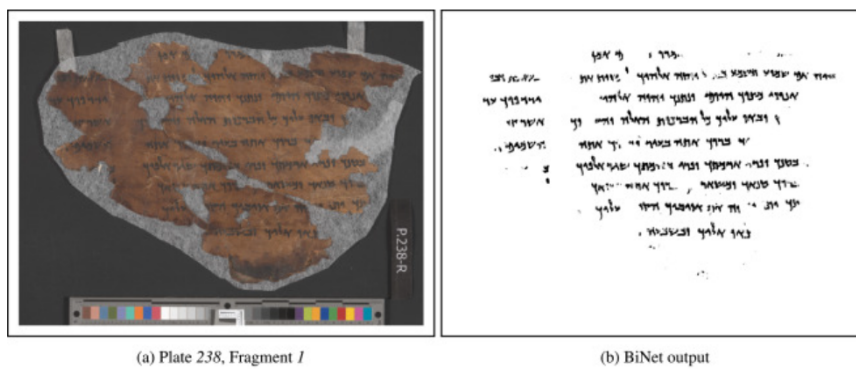


Figure 3.13: Results of binarization using BiNet.[39]

In [40] Nayef et al. they use non-local means denoising together with sparse representations for deblurring digital text. PSNR is also used as an objective measure to compare the deblurred image with the ground truth image. In [41] Sukassini and Velmurugan use median blur to image enhance mammogram images. They also use PSNR to evaluate their results.

3.4.2 Image Segmentation

When working on the DSS, state of the art image segmentation is used for extracting and processing different features like scroll fragments [42] and handwriting styles for historical manuscript dating [39]. We focused instead on segmenting the letters in a scroll. Each letter can then be sent to our classifier which tells the user what letter the classifier thinks it is. The segmented letters can also be saved if the user would like to create or expand a data set. Some letters are also connected so one of our goals is to segment an image with connected letters into individual letters.

In [43], the authors describe how they segmented cursive handwriting. They first correct the images that are slanted and have slopes by using affine 2D transformation. Then they create a skeleton of the image and a histogram of the vertical image pixels quantity. View Figure 3.14 to see an example of a skeletonized image and a histogram of the vertical image pixels quantity. Then they find the segmentation points in the image based on the histogram and a predetermined ideal distance. This method can be used for splitting words with connected letters in the DSS.

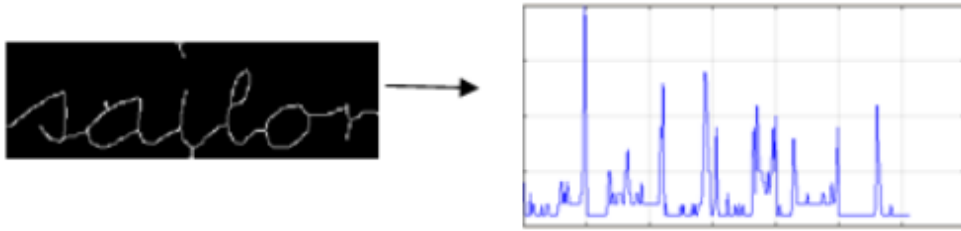


Figure 3.14: Skeletonized image and histogram of the vertical image pixels quantity. Image source: [43]

3.4.3 Machine Learning

State of the art machine learning techniques and methods are used when classifying the DSS characters. The leading solution for character recognition problems involves using OCR, which is a conversion of either handwritten or printed text, to a digital format. In most cases, OCR uses CNN models as the basis of their artificial intelligence, and sometimes OCR is combined with Recurrent Neural Network (RNN). A relevant paper on "Feature-extraction methods for historical manuscript dating based on writing style development" [44] mentions using this CNN approach for another, but relevant dataset. Papers with code⁷ is a website with a collection of state of the art machine learning methods. Here we can see that VAN [45] and StackMix [46] are the best-performing solutions on the "Handwritten Text Recognition" task, and they both utilize CNNs. We have chosen to use convolutional methods for classifying DSS characters because of this powerful performance and ease of implementation.

3.4.4 Summary

In [47] Sudhakaran Jain had a similar approach to what we are going to do in our project. His pipeline can be seen in Figure 3.15. He had a DSS image, did some pre-processing on it, segmented the lines and words and then used a CNN classifier to classify the letters. His pre-processing methods did not include any noise removal or contrast improvement, however, which we are going to implement. Jain uses Otsu binarization, something that we are going to use as well. A lot of work has been done when it comes to using image enhancement to improve OCR accuracy. We wanted to do the same thing just for the DSS. In [47] Jain uses histograms to segment the lines and words in the DSS images. We are going to be using a OCR tool called Pytesseract to directly segment the letters. This makes it so that we do not have to worry about curved lines in the DSS. The kind of line segmentation Jain used struggled to segment curved lines. Jain's classifier uses a sliding window to classify the letters on his segmented words. This is something we do not have to use as our segmentation segments each letter on the DSS images. The only thing

⁷<https://paperswithcode.com/>, visited 23.05.2022

we have to worry about is if our segmentation segments words instead of letters, which is when we will use a word splitter to split the words into letters.



Figure 3.15: Pipeline of Jain's handwriting recognition system.[47]

Chapter 4

Methodology

In this chapter we go through the approaches we used in our project work and explain what choices we made to improve our results when necessary. It contains a walkthrough of methods used in our core topics, namely, dataset (4.1), image enhancement (4.3), image segmentation (4.4) and machine learning (4.5). The link to our GitHub repository and its README can be found in Appendix G.

4.1 Dataset

The dataset contains a set of all the letters in the old Hebrew alphabet that was extracted from the DSS. These were manually cropped by our kind and lovely employer Tabita Tobing, with some help from us. We cropped the letters using a tool called LabelImg (See Section 3.3.3 for more information). The letters were cropped from the DSS and then labeled. The scroll we cropped from was The Great Isaiah Scroll.[4]. The advantage of using LabelImg is that it saves the cropped images in txt-files in the form of coordinates in reference to the DSS image it was cropped from (See Section 3.3.3). One might ask why the tool does not just save the cropped images in the form of images (pixel values) right away instead of coordinates, such that one does not have to convert these coordinates to images later. There is an advantage to doing it this way, however. By only saving the coordinates, we can use those to generate other types of datasets from different variants of the DSS images. For example, we can generate a gray-scale dataset, a binarized one or one that is generated after image enhancement. This way, we can test different dataset formats. Between these methods, we chose to use an Otsu binarized dataset. Tabita Tobing used Otsu binarization when creating her dataset. We wanted all the images in the dataset to be binarized with the same method to ensure that all the images are in the same format.

4.1.1 The Size of the Dataset

'How big a dataset should be?' can be a hard question to answer. In the article called *PHOCNet: A Deep Convolutional Neural Network for Word Spotting in Handwritten Documents* [48] written by Sebastian Sudholt and Gernot A. Fink, the authors found out that when you use a CNN, you do not always need massive amounts of training data while training from scratch. In the article, they state that by using a training set of 3645 images, the authors were able to outperform other methods. They achieved outperforming results by using simple data augmentation and common regularization techniques. [48] Considering this, we figured out that the size of our training set should be somewhere between 3000 and 7000 images. This may also be increased by using different types of augmentation more thoroughly explained in Section 4.5.2. Our dataset, shown in Table 4.1, does not include characters in final form, which are used to represent the letters "Mem", "Nun", "Tsadi", "Pe" and "Kaf" when they appear at the end of a word.

4.1.2 Acquiring the Dataset

As mentioned in the paragraph above, Tabita, with help from us, made the dataset we have used in this project. The dataset was extracted from the Great Isaiah Scroll.[4] Tabita cropped from five different columns in the scroll, and we choose a few columns each to crop from. Tabita cropped about 2500 letters, while our group cropped about 1500. This left us with a dataset of about 4000 letters, which should be more than enough. The only problem was that the dataset ranged from 40 of one letter to 594 of another. Therefore, we decided to focus on the letters with the fewest crops and divided those within our group. Tabita also helped us with this. When we were done with that Tabita had cropped about 3107 letters, while we had cropped about 2841. This left us with a final dataset size of about 5948 letters.

4.1.3 Variance in The Dataset

The dataset contains a lot of characters with different amounts of noise and defects both inside the letters and in the background. There are also some segmented letters with parts of other letters attached to them, because of the writing style of the author and "typeface anatomy". This type of variance can be good because the

Character	Dataset
ALEF	395
BET	198
GIMEL	222
DALET	197
HE	474
VAV	268
ZAYIN	318
HET	260
TET	195
YOD	337
KAF	195
LAMED	364
MEM	239
NUN	191
SAMEKH	165
AYIN	307
PE	195
TSADI	316
QOF	235
RESH	307
SHIN	304
TAV	266

Table 4.1: Distribution of characters in our dataset

classifier might be able to detect letters with noise and defects. It can also be bad because letters with defects might look like different letters. An important characteristic of the letter "Vav" is that the top of the letter has a bend at the top that goes towards the left, and if that letter has a defect where that bend disappears, it can look like the letter "Zayin" (See Figure 3.1). There is also variance between the Hebrew letters themselves which is important to take into consideration. Letters such as "Vav" and "Yod", "Dalet" and "Resh", "Samekh" and "Tav", as seen at 3.1, can be especially hard to differentiate from each other depending on how they are written and how the scribe's writing style is. What information is lost or added during the denoising and binarization steps can also affect this.

4.1.4 Usage of the Dataset

Without any kind of processing, the raw dataset can not be used to train the machine learning model, because neural networks always expect the same input size for each input. Since the images in the dataset vary in size, we need a method to make all the images uniform without losing too much information, so simply resizing and stretching the images is not an option. To do this, we have chosen a method that adds white padding to the images, which makes them all 100x100 pixels in size. The added white space ensures that the letters are centered and maintain their proportions and sizes relative to each other. The pixel size of 100 was chosen to make sure that the new images were always bigger than the pre-processed images, otherwise, the algorithm would break. Before feeding the images to the machine learning model, the pixel values need to be normalized so their value is between 1 and 0.

4.2 System Pipeline

Figure 4.1 shows the process to obtain classified letters from a DSS image.

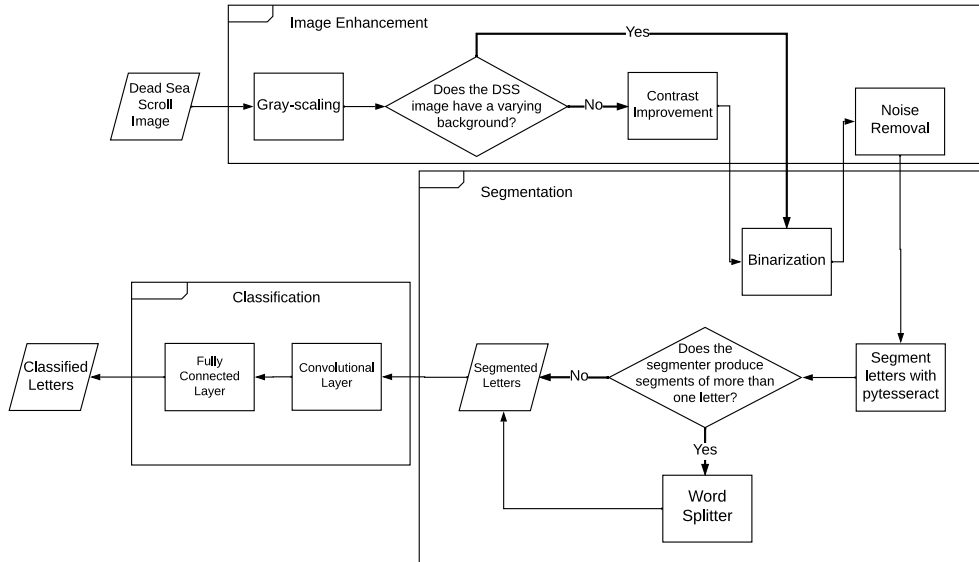


Figure 4.1: This figure gives an overview of the overall process of classifying the letters on a DSS image.

4.3 Image Enhancement

The DSS are very old and fragile. Most of them were written from about 200 B.C. to about 70 A.D.[49] While some of the scrolls are in very good shape, like the Great Isaiah Scroll[4], others are not so well preserved. A lot of them are damaged and have different kinds of stains and cracks. Some of them consist of a lot of fragments as well (See Figure 1.1). That is where image enhancement comes in. If we are to segment the letters on the scrolls, so that they later can be classified, we will have to remove as much of these damages as possible.

4.3.1 Contrast Improvement

We tested different kinds of contrast improvement on The Great Isaiah Scroll. The first approach we tried when it comes to contrast improvement was to use OpenCV's *equalizeHist* method.[11] This improved the contrast between the letters and the background, but it also made the stains in the background in a lot of the DSS images darker and bigger. Because of this we tried OpenCV's *createCLAHE* method, as seen in Code listing 4.1, which utilizes adaptive histogram equalization.[11] This improved the contrast between the letters and the background without making any dark stains in the background darker or bigger.


```
1 import cv2
2 import imageio as img
3 from PIL import Image
4
5 # Read the image using imageio
6 img = img.imread("./img.jpg")
7
8 # Converts the image to gray scale if it isn't already
9 if len(img.shape) == 3:
10     gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
11 else:
12     gray_img = img
13
14 # Performing adaptive histogram equalization on the image
15 clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(80, 80))
16 equalized = clahe.apply(gray_img)
17
18 # Saving the equalized image
19 im = Image.fromarray(equalized)
20 im.save("./equalized_img.jpg")
```

Code listing 4.1: How we use OpenCV to perform adaptive histogram equalization on an image.

4.3.2 Noise Removal

Morphological Transformations

Using morphological transformations such as opening and closing, we have worked on removing Salt-and-pepper noise both in the background and inside the letters we are to segment. The noise in the background of the letters appears as black cracks and dots, while the noise inside the letters are white cracks and dots.

```

1 import imageio as img
2 from PIL import Image
3 import cv2 as cv
4
5 # Read the image using imageio
6 img = img.imread("./img.jpg")
7
8 # Converts the image to gray scale if it isn't already
9 if len(img.shape) == 3:
10     gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
11 else:
12     gray_img = img
13
14 # Inverting the image
15 inverted_img = cv.bitwise_not(gray_img)
16
17 # Creates an elliptical kernel
18 kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (3,3))
19
20 # Performs closing on the image
21 closed_img = cv.morphologyEx(inverted_img, cv.MORPH_CLOSE, kernel)
22
23 # Inverts the image back
24 inverted_back = cv.bitwise_not(closed_img)
25
26 # Saving the image that has gone through closing
27 im = Image.fromarray(inverted_back)
28 im.save("./closed_img.jpg")

```

Code listing 4.2: How we use OpenCV to perform closing on an image.

As you can see from Code listing 4.2, we have to first invert the image using OpenCV's `bitwise_not(img)` method before we perform closing on the image. This is because OpenCV counts the white pixels in a binary image as the foreground and the black pixels as the background. After we have inverted the image we created an elliptical kernel. Closing removes noise inside of the letters and most of that noise is dots, so using an elliptical kernel is the best choice. When the kernel is made we do closing on the image and invert it back again to its original colors. When we perform opening we just switch out `morphologyEx(invertedImg, cv.MORPH_CLOSE, kernel)` with `morphologyEx(invertedImg, cv.MORPH_OPEN, kernel)` and opening is performed instead of closing. We use an elliptical kernel for opening as well as a lot of the noise in the background of the DSS images is dots.

Blur and Denoising Methods

Using OpenCV we have tried different methods for removing noise after binarization on The Great Isaiah Scroll. Binarization is when you make a gray-scale image into a black and white image, which means that all pixel values in the image are either 0 or 255 (0 or 1 if the pixels are normalized to be between 0 and 1). In our case, that means that the text is black and the background is white. How much noise an image will have after binarization varies from image to image and from method to method used for binarization. That is why it is important to choose the correct method for a DSS image. You can read more about this in Section 4.4.1. The different kinds of denoising methods we have tried on these binarized images are median blur, bilateral blur and non-local means denoising methods.

The first method we tried was median blur, by using OpenCV's *medianBlur* method as described in Section 3.2.2. We then tested the method by using methods described in Section 4.3.2. The same trials and testing were also done with OpenCV's bilateral blur and non-local means denoising methods.

```
1 import imageio as img
2 from PIL import Image
3 import cv2 as cv
4
5 # Read the image using imageio
6 img = img.imread("./img.jpg")
7
8 # Converts the image to gray scale if it isn't already
9 if len(img.shape) == 3:
10     gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
11 else:
12     gray_img = img
13
14 # Adaptive binarization of the image using openCV (cv2) module
15 binarized_img = cv.adaptiveThreshold(src=gray_img, maxValue=255,
16 adaptiveMethod=cv.ADAPTIVE_THRESH_GAUSSIAN_C,
17 thresholdType=cv.THRESH_BINARY, blockSize=39, C=15)
18
19 # Doing the non-local means denoising function from the opencv library h
20 # should be 30.0 if the src is a gray image, and 60.0 if the image is binarized
21 denoised_img = cv.fastNlMeansDenoising(src=binarized_img, h=60.0,
22     templateWindowSize=7, searchWindowSize=21)
23
24 # Saving the binarized and blurred image
25 im = Image.fromarray(denoised_img)
26 im.save("./denoised_img.jpg")
```

Code listing 4.3: How we use non-local means denoising.

The first thing this Code listing 4.3 does is to binarize a gray-scale image using OpenCV's *adaptiveThreshold* method. You can read more of this in Section 4.4.1. After that, it uses non-local means denoising to remove noise. The parameter *h* regulates filter strength. Big *h* value perfectly removes noise but also removes

image details, smaller h value preserves details but also preserves some noise. The parameter *templateWindowSize* is the size in pixels of the template patch that is used to compute weights, and *searchWindowSize* is the size in pixels of the window that is used to compute the weighted average for a given pixel.[50] In Figure 4.2 we show a Section of a DSS before and after it has been denoised with non-local means denoising.

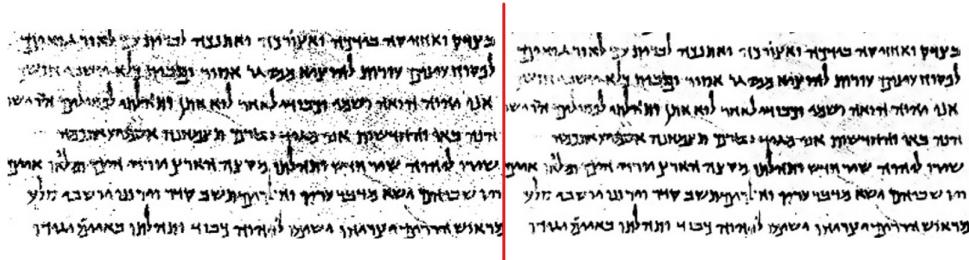


Figure 4.2: A section of The Great Isaiah Scroll column 35 without noise removal (left) and with noise removal (right). The noise removal method used here is non-local means denoising. Image from [4]

Testing the Different Noise Removal Methods

The easiest way to test noise removal methods is by just looking at the results and compare the original image with the processed one. We would say this method of testing is quite effective as it visually is very easy to see if the method removes a lot of noise or not. Another way we have tested the noise removal methods is by testing how well the segmentation works on the image-enhanced images. We can, by doing that, check how many letters our segmentation method can detect and segment on the different image enhanced DSS images. The more letters that it can detect and segment, the better the noise removal method has performed on an image. Checking the PSNR values of images generated by different methods is another way for us to investigate how well these methods perform.

When using PSNR to check how good a noise removal method is, it is necessary to compare the output image to an ideal clean image with the maximum possible power.[6] So what we did was that we cropped a section of a DSS image that had gone through Otsu thresholding and contained some noise. We manually removed all the noise from that section using Microsoft Paint¹ to create an ideally clean image. This image would be our ground truth image. After that, we performed our noise removal methods on that same section (with noise) from the DSS image, and compared the results with the ideally clean image using PSNR.

¹https://en.wikipedia.org/wiki/Microsoft_Paint, visited 04.04.2022

4.4 Image Segmentation

4.4.1 Binarization

Binarization is important for segmenting the letters in the DSS. We binarized the images because we are only concerned with the foreground which is the letters, and do not need their color or gradient. We used two different binarization methods depending on the background. If the background is clear and white we use Otsu's method. Otsu's method works well in those scenarios. If the background contains darker areas we use adaptive thresholding. Otsu does not perform well when the difference between the background and foreground is not large enough. An adaptive thresholding method performs better in these scenarios.

4.4.2 Using Pytesseract to Segment Letters

Figure 4.3 gives an overview of the segmentation process.

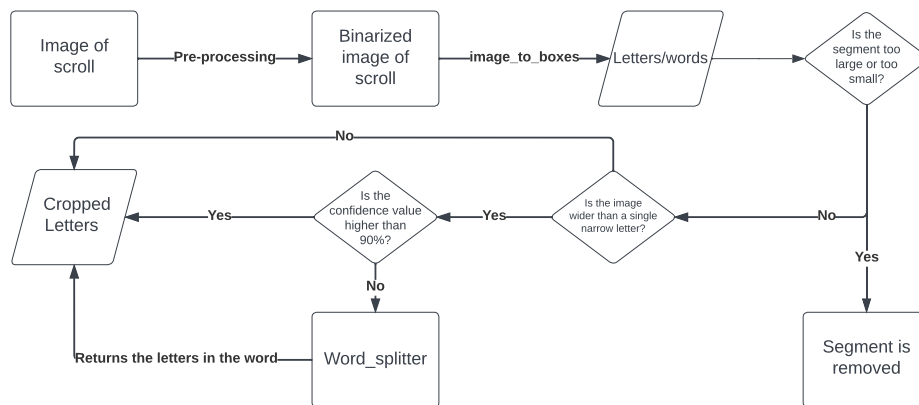


Figure 4.3: Describes the segmentation process.

To extract the letters from the scroll we used the OCR tool Pytesseract. Once we have pre-processed the image we run the method, *image_to_boxes*, by specifying the image we wish to run the method on and our desired language, which in our case is Hebrew.

```
1 boxes = pytesseract.image_to_boxes(image, lang="heb")
```

The method, as seen in the Code listing above, returns a list of all the letters' coordinates and a prediction of what the OCR thinks the letters are. To run the method in Hebrew we download a Hebrew TRAINEDDATA file², which is a model for the OCR. A problem we had with this file was that it was made to recognize

²<https://tesseract-ocr.github.io/tessdoc/Data-Files#data-files-for-version-400-november-29-2016>, visited 26.04.2022

modern Hebrew letters and not the letters in the DSS. To view the difference see Figure 3.1. We decided to create our own custom TRAINEDDATA file. For more details please see Section 4.4.4.

We then have to iterate through the list of all the segments and crop the letters based on the coordinates provided by the *image_to_boxes* method as seen in Code listing 4.4.

```

1  # For each box
2  for b in boxes.splitlines():
3      # Splits the values of the box into an array
4      b = b.split(' ')
5      # Save the coordinates of the box:
6      # x = Distance between the left side of the box to the left frame
7      # y = Distance between the top of the box to the bottom frame
8      # w = Distance between the right side of the box to the left frame
9      # h = Distance between the bottom of the box to the bottom frame
10     x, y, w, h = int(b[1]), int(b[2]), int(b[3]), int(b[4])
11
12     # Crop the image so that we only get the letter/word
13     # Structure image[rows, col]
14     crop = image[(h_img - h):(h_img - y), x:w]

```

Code listing 4.4: They way we iterate through the segments provided by the *image_to_boxes* method, extract each segments coordinates, and crop the image based on those coordinates.

The method will occasionally segment noise, and sometimes also multiple letters into a single segment. This happens because some letters are connected, making Pytesseract interpret it as a single letter. Noise, such as cracks and stains, will also sometimes be incorrectly recognized as a letter. We must therefore filter the results from the *image_to_boxes* to remove segments that are too large and too small. What happens to the segmented image, after the filter has been performed, can be divided into three procedures:

1. The segment gets removed because the segment was too large or too small, or
2. the segment gets passed to the Word splitter, or
3. the segment does not need extra processing, meaning it is a successfully segmented letter.

The code checks if the segment should be passed to the *word_splitter* function. This is done by checking the width of the segment. Some letters can be large and can cause the code to think it is a word with connected letters. Figure 4.4 shows two images with a similar width, where one of the images contains a single wider letter and the other image contains two thin connected letters.

The code, therefore, sends the segmented image to the classifier. If the classifier returns a high confidence value, above 90 percent, the code assumes that it is a single letter. If the confidence value is below the 90 percent threshold, the segment gets passed to the *word_splitter*. The classifier is trained on recognizing letters, not words. Therefore if a word becomes an input into the classifier, it should return a

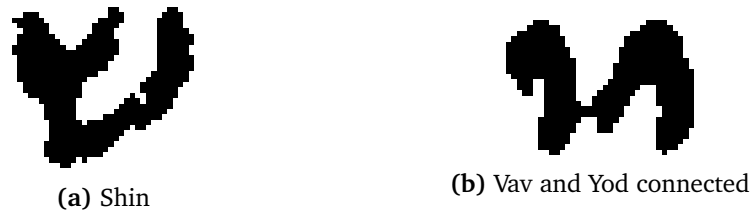


Figure 4.4: A single larger letter and two narrow letters that are connected.

low confidence value.

The letter image is then stored into a Letter object that contains the image, the coordinates, the classification, and the confidence value. The letter is now ready to be sent to the classifier to be classified. This is done by appending the Letter object into an array. The classifier can then iterate through the array and append the letter's classification and confidence value to its Letter object.

The user can also draw rectangles on an image around the letters to visually see the results of the segmenter, like in Figure 4.5 using the *rectangle* method shown in Code listing 4.5.

```
cv2.rectangle(image, start_point, end_point, color, thickness)
```

Code listing 4.5: A method for drawing rectangles on an image.

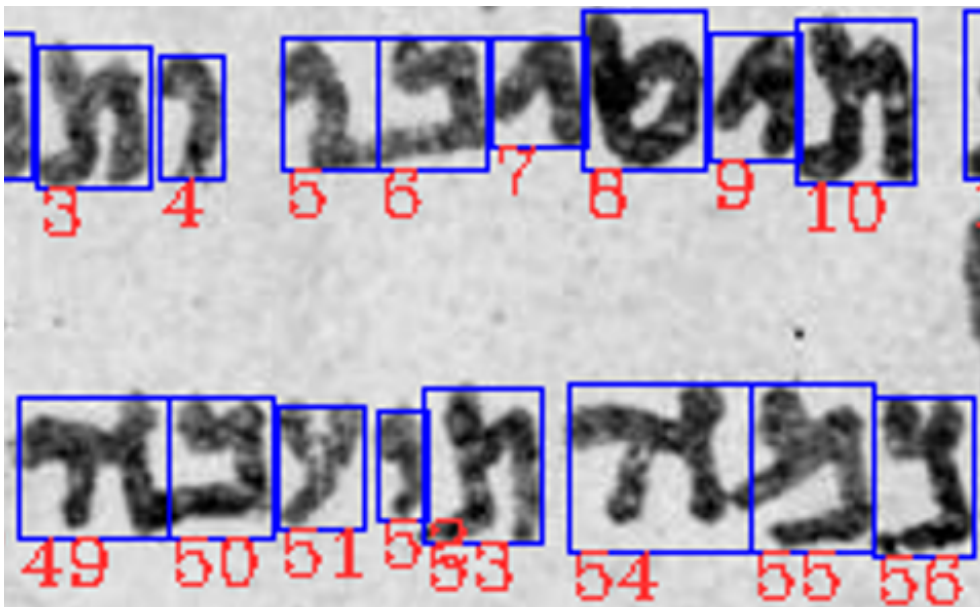


Figure 4.5: Image of scroll, gotten from [4], with rectangles around letters. Includes also the ID of the letter, which does not come with the *cv2.rectangle* method.

One can also save the letters into a folder such as in the Code listing 4.6.

```
1 print(("Saving image number:" + str(id)), cv2.imwrite(str(ID) + 'unique_file_name.png', image_of_letter))
```

Code listing 4.6: Saves the letters image. If the letter image was saved correctly, it will print the ID of the image that was saved and "True".

4.4.3 Word Splitter

Figure 4.6 gives an overview of the Word splitter's process.

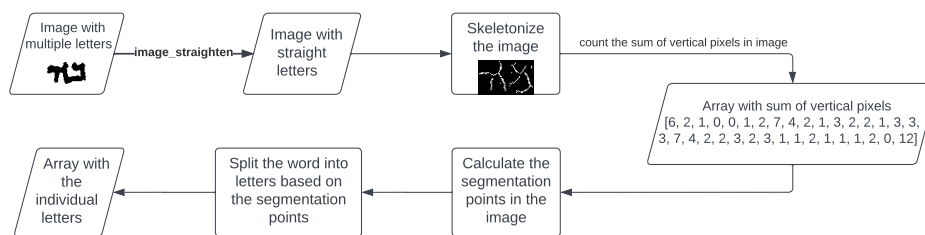


Figure 4.6: Gives an overview of the word splitter's process.

Pytesseract will occasionally segment connected letters because the OCR thinks it is a single letter. Our classifier is made to classify one letter at a time. If one wants to use Pytesseract to expand a dataset, they will need to manually split these words which can be time-consuming. These are the reasons why we worked on a method for splitting words into letters.

We used parts of this article's approach, [43], to segment cursive handwriting to create our word splitter function. From this approach we use these components: straightening the letters, creating a skeleton, and creating a histogram of the vertical pixels in the image. Our word splitter method takes an image with multiple letters as input. This is an example 4.7.



Figure 4.7: Image of the letters "Mem" and "He" connected.

We used a module called *image_straighten.py* from here [51]. What this module does and why we used it will be stated later in this paragraph. After we straightened the letters we created a skeleton of the image. We used this source, [52], to skeletonize our images. We then count the sum of the vertical pixels of the skeletonized image and store it in the array *amount_vert_pixels*. This array contains information about where the straight lines are in the image. If the array

contains an element with a high value, it means there are a lot of pixels in that column of the image, which indicates to us that there is a straight vertical line there. Some lines in certain letters can appear straight to us, but when skeletonized those lines can be slanted. In this scenario, when counting the sum of the vertical pixels, it will be harder for us to tell from the array where the vertical lines are, since the values of the elements in that array will be more evenly distributed. The *image_straighten* module straightens the letters in an image, which prevents this issue.

We looked at a lot of words that had connected letters and we found out that a lot of the letters began with a straight line. We could then use that information to segment the words into letters by detecting the straight vertical lines in the image. An example of this can be seen in this Figure: 4.8.



Figure 4.8: Two letters connected to each other. Each letter begins with a straight line, shown with a red line, when reading from right to left.

The *segmentation_point_finder* method uses the *amount_vert_pixels* array to find the segmentation points, which are points that represent where the word should be split into multiple letters. A *minimum_letter_width* needs to be set. A letter might have two vertical lines close to each other. The *minimum_letter_width* constant will be used to prevent segmentation points from being too close by checking if the distance between the previous segmentation point and the potential new segmentation point is shorter than the *minimum_letter_width*. We defined that constant to be 12 by looking at the thinnest letters, "Vav" and "Zayin", and checked how many pixels in width they were. Our theory is that we should not create a segment that is narrower than the thinnest letter.

The *segmentation_point_finder* iterates through the *amount_vert_pixels* array and appends a segmentation point to the array *seg_points* if it finds a vertical line. This indicates the right side of a letter. If the method finds two columns with no pixels, it will also append a segmentation point to indicate the left side of a letter. If the method was run on the word in this Figure 4.8, the array [18, 0] would be returned. The letter on the right is between *x_value* 18 and the far right side of the image. The letter on the left starts from the far left side of the image to the *x_value* 18.

The code will now split the image using the segmentation points array, *seg_points*. When iterating through that array, it checks if the letter it is segmenting is all the way to the right, in the middle, or on the far left side of a word. This is done

because the code uses the classifier to check if the segmentation of the letter was successful or not. If the segmentation contains only part of a letter, the classifier will return a low confidence value. If this happens the code will extend the width of the image, which should fix the issue of only containing part of a letter. If the letter is all the way to the left of the segment, we do not want to extend the image further to the left. If the letter is all the way to the right of the segment, we would only want to extend the image to the left. Therefore we need to check where the letter is in the segment, to know how we are going to extend the segmented letters.

Once we know if the letter is all the way to the right, left or middle of the word, we segment the letter from the image of the word. The code checks the confidence value of the segmentation of the letter. If the confidence value is above a certain threshold, it will create a Letter object for the image and append the object to the array *segmented_letters_in_word*. If the confidence value is below the threshold, it will extend the image by two pixels. Then we check the confidence value again and check if it is above the threshold or not. If it is not above the threshold, it compares the previous confidence value with the new one. The best confidence value and the amount the image has to get extended to achieve that confidence value gets saved as *best_extend_image*. This is done to know how much the image needs to be extended to get the best confidence value when segmenting the letter. If extending the image of the letter goes out of the bounds of its outer image (such as words or a couple of letters close to each other) with the multiple letters, or if the code extends the image by more than half of the *minimum_letter_width_distance*, the code will extend the image with the *best_extend_image* value to ensure we get the best segment possible. The best segment is a segment that only contains the entire letter we wish to segment. After the letter has been cropped it will create a new Letter object of the image with its corresponding coordinates and append it to the *segmented_letters_in_word* array.

Once all the letters are cropped and have been appended to the array we reverse the array so that the letters are in the correct order. Pytesseract segments the letters by starting on the top left of the image, and then works from left to right, line by line. The word splitter segments the letters from right to left, because Hebrew is read from right to left, and our theory is that most letters start with a straight line. The word splitter function will then return the array with all the Letter objects.

4.4.4 Custom TRAINEDDATA File

Pytesseract does not have a TRAINEDDATA file for the Hebrew used in the DSS. This is a problem because when Pytesseract tries to recognize the letters it is looking for modern Hebrew characters, which look different from the ones in the DSS. An example of the difference can be viewed in Figure 4.9. This has caused errors in our segmentation such as overlapping segments and segments that are very

large and span over multiple words and lines.



(a) The letter "mem" found in the first para- (b) The letter "mem" in modern Hebrew. Im-
 graph in column 35 in 1QIsa^a. image found in Figure 3.1.

Figure 4.9: A letter from the DSS and the same letter from modern Hebrew.

We decided to create our own custom TRAINEDDATA file with our own dataset. We followed all of the steps listed here: [53]. We used the article [54] to get a better understanding of the steps in the previous source. To do this we needed to create a font with the letters from the DSS. We used the website Calligraphr³ to create our own font. Finally, we created our custom TRAINEDDATA file with the steps found in Appendix E.

4.4.5 Calculating the Intersection over Union Score

We tested most of our image segmentation methods by reviewing our results visually. We did this by creating rectangles around each segment and drawing them onto the original image. At the beginning of the project, this worked well since the problems we were experiencing were easily spotted on the image with the rectangles. Longer into the project it became harder and harder to evaluate if the new methods we were working on were improving the results. This is when IoU is helpful. We can use the IoU score to determine how well the segmenter segmented the letter and calculate the average IoU score of all the letters.

We started by manually creating a ground truth using LabelImg, see Section 3.3.3 for more information. The ground truth is saved in the YOLO format in a text document. We then used those coordinates to crop each letter. We created a Letter object for the cropped image which also includes the letter's coordinates. Then we put each Letter object into an array. To get the predicted segmentation of the letters we ran the segmenter on the same image which returns an array of all the letters.

To calculate the IoU score we needed to compare the right letters. To do this we used the method in this article [55], which takes two rectangles and checks if they overlap each other. If they don't overlap it returns False, otherwise it returns True. We took a letter from the ground truth array and checked if a letter from the segmenter overlaps it. If it does, we calculate the IoU score using the method in the article [22] and append that score to an array that contains all the IoU scores.

³<https://www.calligraphr.com/en/>, visited 03.05.2022

After iterating through all the ground truth letters and calculating the IoU score to its overlapping segment from the segmenter, we calculated the average IoU score.

4.5 Machine Learning

4.5.1 Model

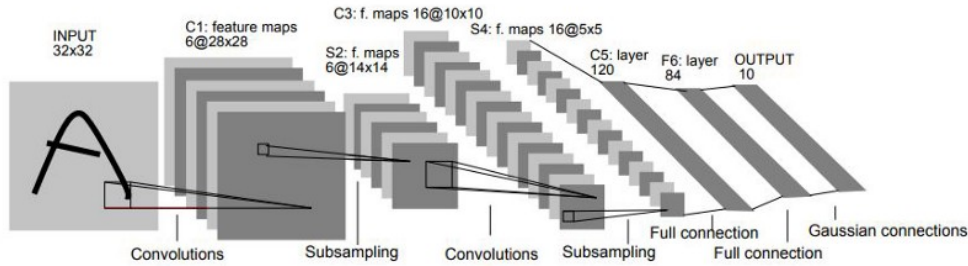


Figure 4.10: Figure detailing the architecture of the LeNet model [56]

We based our neural network as seen in Code listing 4.9 on the convolutional architecture LeNet [56] as proposed by Yann LeCun, which can be seen in Figure 4.10. It was chosen because it was easy to understand, but also of its solid structure and good performance, which suited our needs. Our network is divided into two parts, one for feature extraction and the second for classification.

```

1 self.convolutional = nn.Sequential(
2     nn.Conv2d(1, 6, 5),
3     nn.ReLU(),
4     nn.MaxPool2d(2,2),
5     nn.Conv2d(6, 16, 5),
6     nn.ReLU(),
7     nn.MaxPool2d(2,2)
8 )

```

Code listing 4.7: Implementation of the feature extraction layers in PyTorch

The first part takes a 100x100x1 image as input and feeds it through two sets of convolutional pooling layers feeding into each other, for efficient feature extraction. An example of this structure can be seen in Code listing 4.7. The convolutional layers use a 5x5 kernel with a stride of one, without applying any padding to the image. Before the feature maps are sent to the 2x2 max pooling layers, we used a ReLU activation function to add non-linearity. The output of the feature extraction part is a 22x22x16 feature map, comprised of different image features.

```
1 size = int((size/4) - 3)
2 self.fullyconnected = nn.Sequential(
3     nn.Linear(16 * size * size, 256),
4     nn.ReLU(),
5     nn.Linear(256, 128),
6     nn.Sigmoid(),
7     nn.Linear(128, 22)
8 )
```

Code listing 4.8: Implementation of the classification layers in PyTorch

This map is fed into the classification portion, which consists of a fully connected layer with an output of 22 classes as seen in Code listing 4.8. The fully connected layer is made of one input, hidden and output layer, going from 256, 128 and 22 nodes. In PyTorch, we have used the neural network modules `Linear`⁴ methods, which are equivalent to `Dense` in TensorFlow. For the last activation function before the output layer, we have used a Sigmoid function instead of ReLU, as to properly display the confidence values as percentages. For this model, we chose to use Cross-Entropy Loss for our loss function, and Adam for our optimizer. Cross-Entropy Loss is a good function to use for multi-label classification problems, where the function output is a probability between 0 and 1.

⁴<https://pytorch.org/docs/stable/generated/torch.nn.Linear.html>, visited 03.04.2022

The main difference between the LeNet architecture and the architecture proposed in our model is the image input size, and the use of ReLU activation functions between the dense layers, instead of Sigmoid. These changes were made while tweaking the model early, and seemed to give better results in our case.

```

1 class Convolutional(nn.Module):
2     def __init__(self, size):
3         super(Convolutional, self).__init__()
4         self.convolutional = nn.Sequential(
5             nn.Conv2d(1, 6, 5),
6             nn.ReLU(),
7             nn.MaxPool2d(2,2),
8             nn.Conv2d(6, 16, 5),
9             nn.ReLU(),
10            nn.MaxPool2d(2,2)
11        )
12
13        size = int((size/4) - 3)
14        self.fullyconnected = nn.Sequential(
15            nn.Linear(16 * size * size, 256),
16            nn.ReLU(),
17            nn.Linear(256, 128),
18            nn.Sigmoid(),
19            nn.Linear(128, 22)
20        )
21
22    def forward(self, x):
23        x = self.convolutional(x)
24        x = torch.flatten(x, 1)
25        x = self.fullyconnected(x)
26        return x

```

Code listing 4.9: Implementation of the convolutional neural network structure in PyTorch

4.5.2 Model Improvements

To combat the issues resulting from using a small dataset for machine learning, the group has tried several methods to increase the models' performance, including different machine learning techniques and augmentation.

Transfer Learning

In an attempt to use transfer learning to increase model performance, we tried training our machine learning model with the Modified National Institute of Standards and Technology (MNIST) dataset [57]⁵, by switching out the output layer for a layer with 10 nodes instead of 22. With this MNIST dataset trained model, we froze all the weights in the fully connected layer with the method shown in Code listing 4.10, and then exchanged the last layer with a new dense layer with 22

⁵<http://yann.lecun.com/exdb/mnist/>, visited 08.05.2022

nodes. This would "reset" the weights and unfreeze them. When we then trained the model with our dataset, it would only affect the weights in the last layer. We did not try other pre-trained models because of difficulties with either getting those to work with our dataset format and size and difficulties with changing their layer structure.

```
1 model_conv = model.Convolutional()
2
3 # Freeze the weights
4 for param in model_conv.parameters():
5     param.requires_grad = False
```

Code listing 4.10: Method used for freezing weights in the fully connected layers for our model

Augmentation

In an article called *A Digital Palaeographic Approach towards Writer Identification in the Dead Sea Scrolls* by Maruf A. Dhali et al., they propose using augmentation if you need a bigger and more diverse dataset [58]. Throughout the process of acquiring our dataset, we have tried various types of augmentation to increase the dataset size. All have given various kinds of results. The first type of augmentation we tried was random distortions. The changes in result from these were very small, so we quickly moved on to more common handwritten augmentation methods like rotation, dilation and erosion. We first tried doing the same augmentations for every letter, this did not provide any improvements to the model so we switched over to doing some random augmentations to randomly selected images.

Chapter 5

User Interface

In Chapter 4 we went through how we used image enhancement and image segmentation to improve the accuracy of our machine learning model. What we wanted from this user interface was to put those three parts together into one program so that both our group and the user can see how they worked together. The user interface also makes it easier for us to test our segmentation and machine learning, and it makes it easier to showcase it and explain it to others. This chapter goes through how the user interface was planned and made.

5.1 Functional Requirements

In Section 1.3 we stated that the user should be able to extract features from the DSS through the use of an easy and simple user interface. The user interface should also be able to run on Windows 10 and Windows 11.

5.1.1 User Patterns

Our employer has not given us any functional requirements when it comes to the user interface, other than that it should be able to help them extract features from the DSS easier and faster than how they do it today. Since they did not specify any requirements, we did that ourselves. Figure 5.1 shows the use case diagram for our user interface.

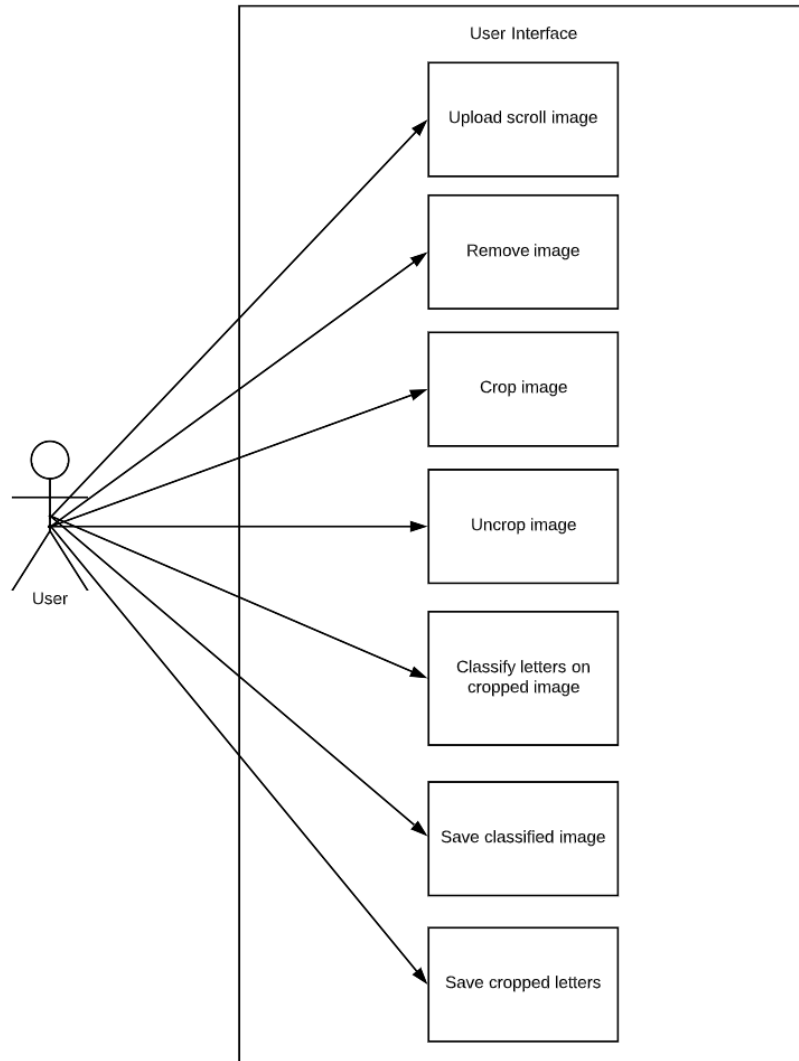


Figure 5.1: Use case diagram

Table 5.1: User patterns

User Pattern: Open an image.

Goal: Open a new image and display it in the user interface.

Description: The user can open an image by opening the file explorer and dropping the image in the "Drop here zone", or press the "Open Image" button and the file explorer will be opened so the user can select an image. The image and its path then gets displayed.

User Pattern: Remove an image.
Goal: Remove the displayed image from the user interface.
Description: The user removes the displayed image and its path.

User Pattern: Crop an image.
Goal: Crop the displayed image on the user interface.
Description: Crops the image and keeps the crop at the same position on screen.

User Pattern: Uncrop an image.
Goal: Get the original image to be displayed again.
Description: Displays the original image that was cropped.

User Pattern: Saves an image.
Goal: Save the displayed image to the pc.
Description: Opens the file explorer and lets the user decide file name, file format and file location.

User Pattern: Classify the Hebrew letters in an image.
Goal: Classify the Hebrew letters in the image.
Description: The user interface will classify the letters in the image and labeled boxes will appear over them. The labels on the boxes correlates to what the machine learning classifies the letter as.

User Pattern: Save cropped letters in image.
Goal: Save the cropped letters in image in a folder.
Description: After the image has been classified the user interface will let the user save the cropped letters in a folder on their computer.

5.2 Sketch

At the beginning of the project, we sketched out how we wanted our user interface to look based on what functionality we specified. The final product turned out to be very similar to our sketch. The sketch shows a drag-and-drop area where the user can drag and drop DSS images. When the user has dropped or opened an image, the filename will appear above the displayed image. Under the drag-and-drop area, you can see the buttons with most of the functionality described in Section 5.1.1.

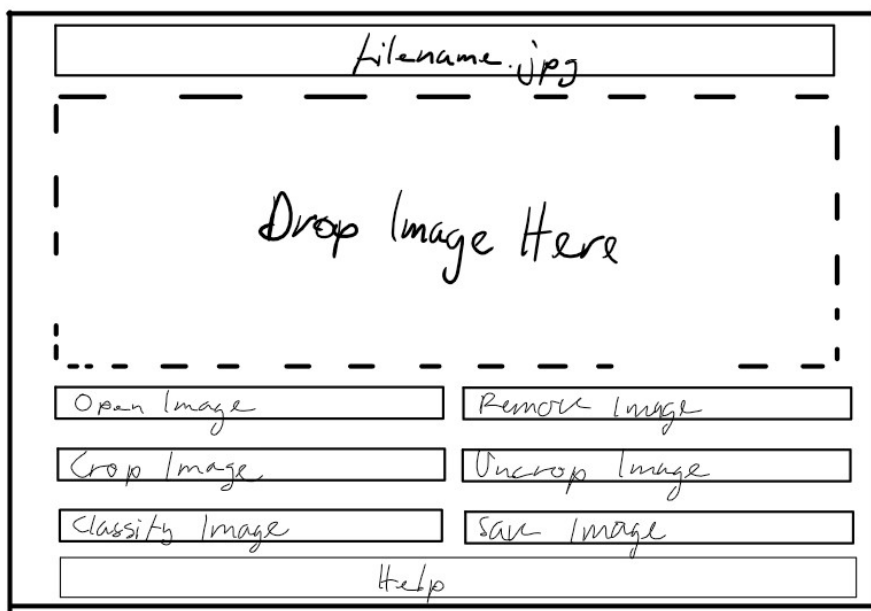


Figure 5.2: Simple sketch of our user interface.

5.3 Method

For coding the user interface we decided to stick with Python as we had used that for all the other coding done in the project so far. This also made it easier for us to put the image enhancement, image segmentation and the machine learning code together to create the user interface. It will also make it easier for others to modify and expand upon our solution, which was one of our goals. To code the user interface we decided to use PyQt5¹.

¹<https://www.riverbankcomputing.com/static/Docs/PyQt5/>, visited 18.03.2022

5.3.1 PyQt5

PyQt5 is a Python binding of the graphical user interface toolkit Qt version 5. Qt is a set of C++ cross-platform libraries that among other things include traditional user interface development. PyQt5 is implemented as more than 35 extension modules and is made by Riverbank Computing Limited².^[59]

```
1 import sys
2
3 from PyQt5.QtCore import QSize, Qt
4 from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton
5
6
7 # Class that represents the main window of the application
8 class MainWindow(QMainWindow):
9     def __init__(self):
10         super().__init__()
11         # Sets the window title
12         self.setWindowTitle("My App")
13
14         # Creates a button
15         button = QPushButton("Press Me!")
16
17         # Sets the size of the window
18         self.setFixedSize(QSize(400, 300))
19
20         # Set the button as the central widget of the Window.
21         self.setCentralWidget(button)
22
23
24 app = QApplication(sys.argv)
25
26 window = MainWindow()
27 window.show()
28
29 app.exec()
```

Code listing 5.1: Simple example of how to create a simple window with PyQt5

²<https://www.riverbankcomputing.com/>, visited 18.03.2022

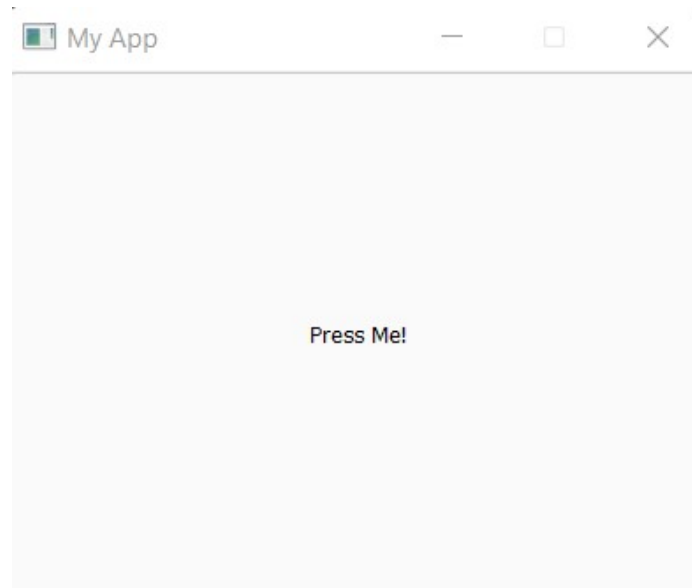


Figure 5.3: This is what Code listing 5.1 produces.

5.3.2 Implementation of Image Enhancement, Image Segmentation and Machine Learning

Our user interface consists of two python files. One of them uses PyQt5 to create the actual user interface and the other uses our image enhancement, image segmentation and machine learning to classify the image that the user has provided. The latter file will take the DSS image, image enhance it, segment the letters and then send those segmented letters to the machine learning to be classified by the model. The machine learning will then send back the segmented letters with a label containing the name of the predicted letter, and a confidence value saying how confident the model is that the letter was predicted correctly. Rectangles will then be drawn over the letters labeled with the prediction and the confidence value.

When a user opens an image, radio buttons will appear asking the user if the background of the image is varied or not. This is because we have implemented two types of image enhancement processes. One is to be used if the DSS image has a varied background and the other is to be used if the DSS image has a clear and white background. You can read more about this in Section 7.1.4.

5.3.3 Design

In our goals, we stated that we wanted to make a simple and intuitive user interface for our employers. That also means that the design should be simple and intuitive. In making the user interface we tried following the fundamentals of interaction design. [60] The third one of these principles concerns signifiers. This means that we signify to the users what actions are possible in a good way. There

are not a lot of actions possible in our user interface, but all of them are signified to the user in a good way, either through buttons or labels. The buttons are arranged in a sort of chronological order of how a user would typically use the user interface, starting from the upper left "Open Image" button to the "Save Image" button in the lower-left corner.

The fourth concept from the fundamentals of interaction design is feedback. The user should get feedback in the form of error messages, confirmations and task statuses, as seen in Figure 5.4. The user will get an error message if they try to remove an image when no image is displayed or if they try to un-crop an image that has not yet been cropped. The user will also get a confirmation message when an image has been classified, telling them that the classification is complete. The error messages work as constraints for the user, which touches upon the sixth concept from the fundamentals of interaction design. These constraints are there as guiding rails to guide the user to do certain tasks in a particular order. Another form of feedback is task status in the form of loading animations. When an image is being classified a loading animation will appear telling the user that something is being processed. During classification, the buttons will be disabled, so that the user cannot click the buttons and crash the program. Another form of task status the user will get is in the form of a zoom level label, which will tell the user how far zoomed in or zoomed out they are. These kinds of constraints and feedback to the user also follow the seventh concept concerning consistency. The user will be provided with the same response from the system for the same actions.

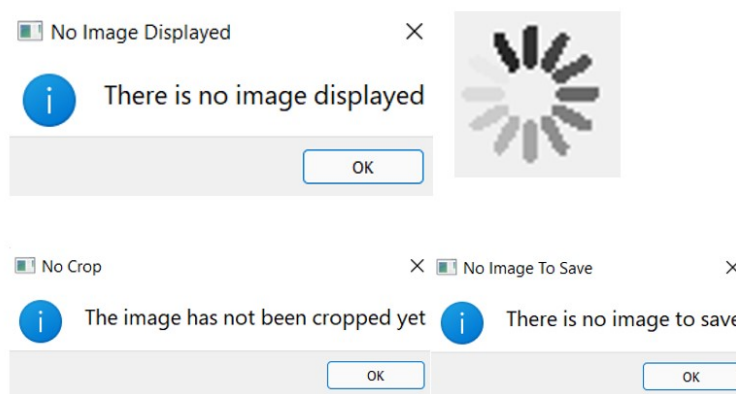


Figure 5.4: These are the kind of error messages and loading animations that will give feedback to the user.

5.4 Results and Discussion

5.4.1 Flow Chart

Figure 5.5 shows the complete flow of our user interface. It starts with the user opening the user interface and ends with the user either getting and saving the results or not getting any results.

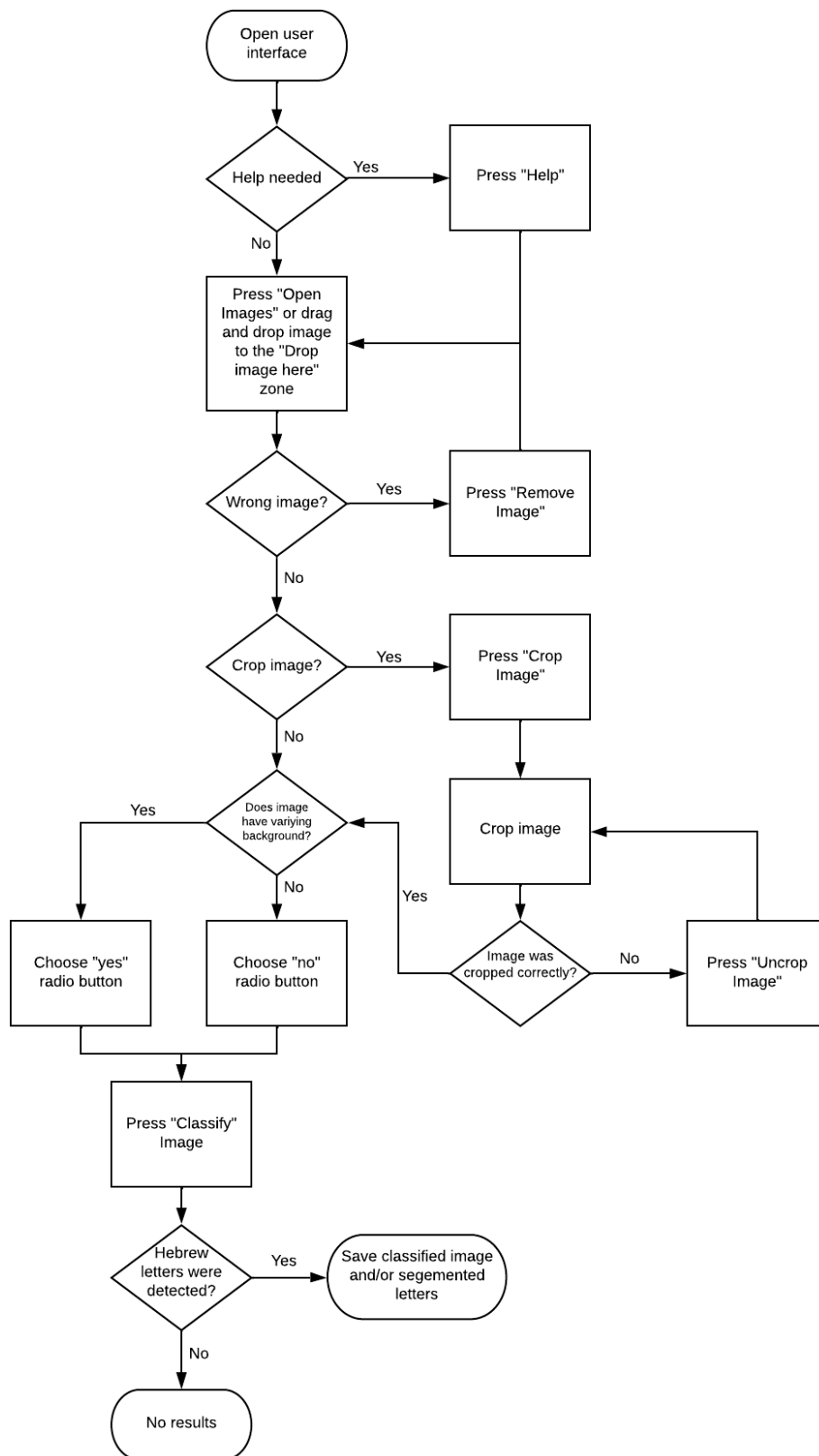


Figure 5.5: Flowchart for user interface.

5.4.2 Final Solution

A link to the user interface's GitHub repository and its README can be found in Appendix F:

Figure 5.6 shows what the user is met with when they open the user interface. Here they can press help if they need help with getting started with the user interface, or they can upload an image to the user interface either by using the "Open Images" button or by dropping an image in the "Drop image here" zone. If the user chose the wrong image or if they want to change the image they can press the "Remove Image" button. Figure 5.7 shows what the user interface looks like when an image has been uploaded. A zoom label will appear letting the user know how far in or out they have zoomed and radio buttons will appear that are needed for the classification part of the user interface.



Figure 5.6: This is what our user interface looks like.

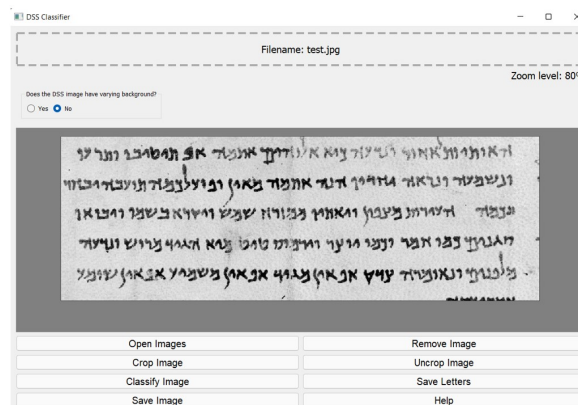


Figure 5.7: This is what our user interface looks like when an image has been uploaded. DSS image gotten from: [4]

When the user has uploaded an image they can choose to crop it if there are

only certain areas of the image that they want to classify. Figure 5.8 shows what happens when the user clicks the "Crop Image" button and Figure 5.9 shows what the result of the cropping looks like. When an image has been cropped the user can un-crop the image using the "Uncrop Image" button.

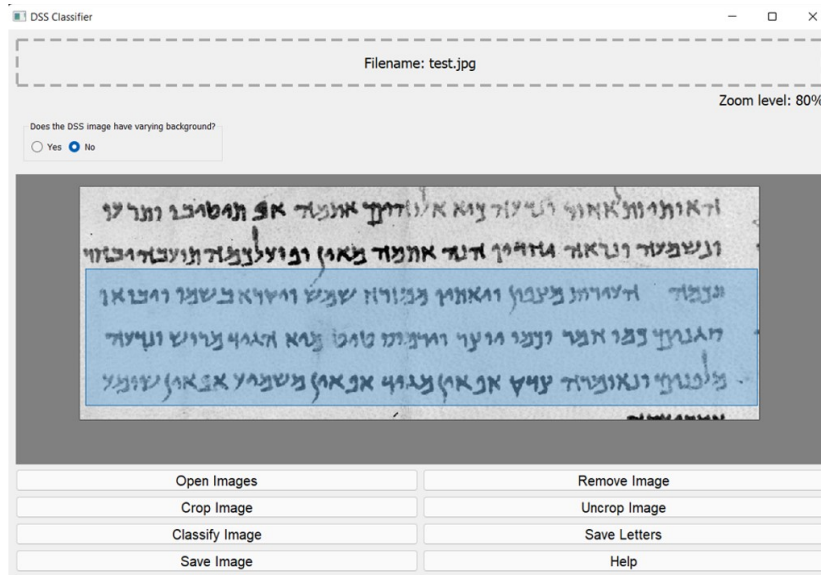


Figure 5.8: Cropping in our user interface. DSS image gotten from: [4]

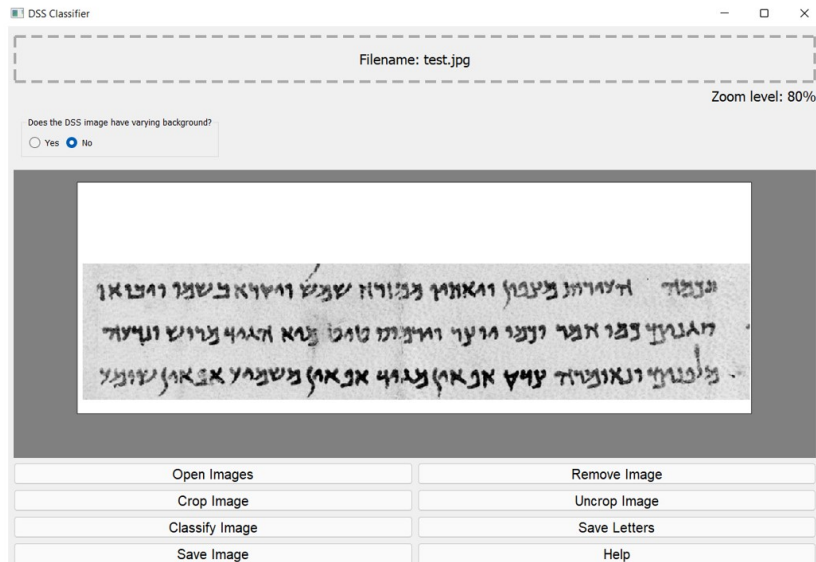


Figure 5.9: What a cropped image looks like in our user interface. DSS image gotten from: [4]

Once an image has been uploaded to the user interface radio buttons will appear asking the user if the DSS image has a varying background. The user will then have to choose "Yes" or "No" here accordingly. After that, the user can proceed to classify the image using the "Classify Image" button. Figure 5.10 shows what the image looks like when it has been classified. Once the image has been classified the user can choose to download the classified image with the "Save Image" button and/or save the segmented letters with the "Save Letters" button. The letters will then be downloaded in a folder called "letters" in the user interface folder.

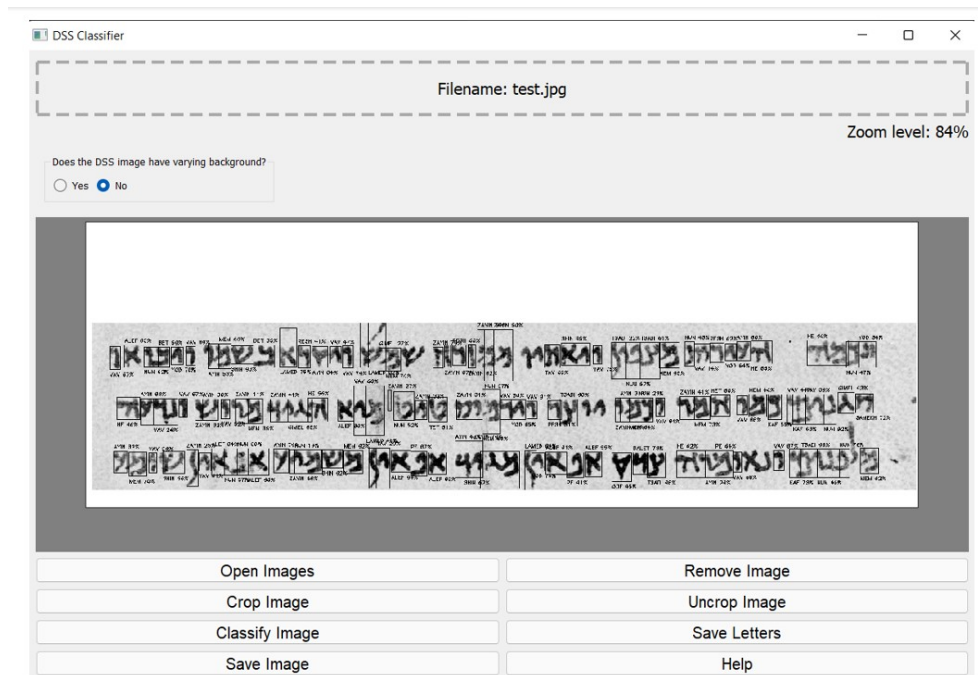


Figure 5.10: What a classified image looks like in our user interface. DSS image gotten from: [4]

5.4.3 Classification Times

We used the Great Isaiah Scroll column 35 when we tested the user interface, which has a size of 1999x2760 pixels. We tested the classification time by pressing the timer when we hit the "Classify Image" button and stopping it when the DSS image was classified. Using the Great Isaiah Scroll column 35 we got classification times of around 1 minute on average.

5.4.4 Future Improvements to the User Interface

In Section 1.3 we stated that the user should be able to extract features from the DSS through the use of an easy and simple user interface. Since the user interface is easy and simple and made mostly to tie together the image enhancement, image

segmentation and machine learning, there are a lot of possible improvements that could be made. Here is a list of improvements we feel could be done in the future:

- It would be great if the rectangles around the letters were clickable. If clicked they could show the top five confidence values for example.
- It would also be nice if the user could alter the classification results i.e. the labels under/over the letters if the user notices that the model has classified wrongly. The model could then also learn from the correction of the user.
- The same could also be done about the letter segmentation. If the user notices that a letter has been segmented wrongly he can alter the rectangle around it.
- When an image has been uploaded it would be nice if the user could manually remove noise with some sort of paintbrush or eraser tool. This could improve the precision of the classifier.
- It would be nice if the user could cancel the classification process when that has been started.
- Right now in our user interface we use our own model, but it would be nice to give the user an option to use their own model if wanted.
- Since the labels added to the rectangles drawn over the letters are so small the text can sometimes be a bit hard to read. A thing we could have done is made the text more readable.

Chapter 6

Quality Assurance

Our group has used different tools and methods to both aid in keeping an organized workflow, and to ensure the code quality is maintained.

6.1 Code Quality

The group decided to adhere to the "pythonic" way of writing code, as a way to both leverage Python and make the different code parts more cohesive. This means that the group tried to follow the PEP 8 ¹ guidelines as much as possible, as a way to increase readability. This means using the naming conventions snake_case for variables and PascalCase for class names. Some external packages do not follow the exact same naming convention, so some methods in our code will be camel-Case as well.

6.1.1 Tools

To help keep the project's dependencies organized, our group has used a combination of a virtual environment and python's standard package management system, pip, to install all the required python packages. This approach keeps the project packages separate from other packages on the system, and makes it easy and quick to delete or reinstall packages.

Linting is a process that analyses how the code runs and highlights syntactical and stylistic problems in the code, which makes it easier to avoid errors or bad practices while writing code.

IntelliSense is a tool that makes writing code easier and less prone to errors. It is a general term for a bunch of useful features which include code completion, content assist and code hinting.

¹<https://peps.python.org/pep-0008/>, visited 18.05.2022

The group is split between working on PyCharm and Visual Studio Code (VSC), which means there is some difference between the development environment tools. PyCharm has built-in tools for Linting, IntelliSense and debugging, while VSC requires external extensions. The group has decided to use Microsofts Python and Pylance extensions to enable this functionality because they are the most widely used and offer an all-in-one solution.

We also utilized the development platform GitHub for various reasons, with the main one being a hosting platform with version control, which would make collaborating easier. We also used GitHub as a tool to track and plan progress through development, through the use of GitHub issues and a Kanban board.

6.1.2 Documentation

While not spending time creating diagrams or documents specifically for documenting the code, the group would thoroughly utilize code comments as a means to document the functionality and variables of the code. This would make it easier for fellow group members to know what the code does, and make it easier to edit later. We also made READMEs for both our user interface repository and main repository, to explain how to run our code.

Chapter 7

Results

In this chapter we go through the results gotten from our methods described in Chapter 4 (Methodology). It contains a walk through of results obtained in our core topics image enhancement (7.1), image segmentation (7.2) and machine learning (7.3).

7.1 Image Enhancement

7.1.1 Adaptive Histogram Equalization

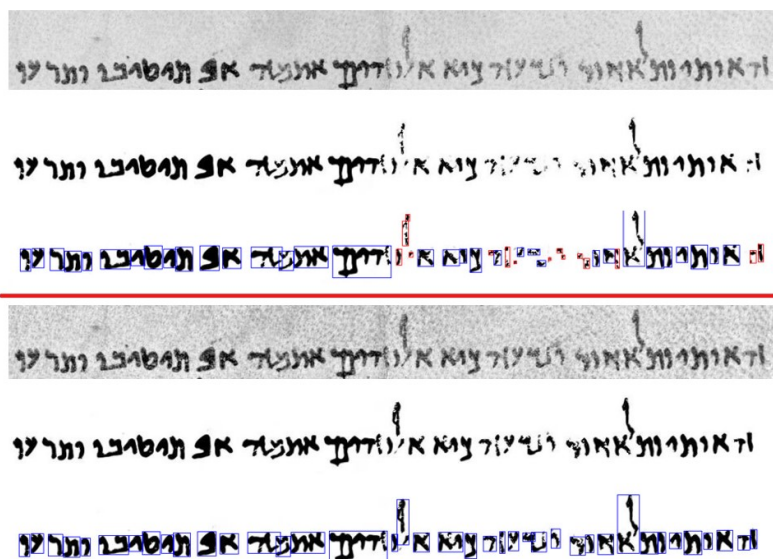


Figure 7.1: Adaptive histogram equalization results of a section of The Great Isaiah Scroll column 35. Above the red line: grayscale, Otsu binarization, Pytesseract segmentation. Below the red line: grayscale that has been adaptive histogram equalized, Otsu binarization, Pytesseract segmentation. DSS images gotten from: [4]

From Figure 7.1 we can see the difference adaptive histogram equalization has on Otsu binarization and Pytesseract¹ segmentation. Without adaptive histogram equalization, some letters disappear when Otsu thresholding is performed. This in turn makes it impossible for Pytesseract to segment those letters. With adaptive histogram equalization, however, Pytesseract can segment almost all of the letters correctly.

There are some of the columns in The Great Isaiah Scroll that have some darker regions in the background. When doing adaptive histogram equalization on those columns, the darker regions get darker, and Otsu thresholding performs poorly. Adaptive histogram equalization should not be used on those kinds of DSS images. An example of such a DSS image can be seen below in Figure 7.2. Here we can see (from the lower image) that the background, especially in the middle of the column is very dark, and Otsu thresholding will perform poorly in this region.

¹<https://github.com/tesseract-ocr/tesseract>, visited 04.04.2022

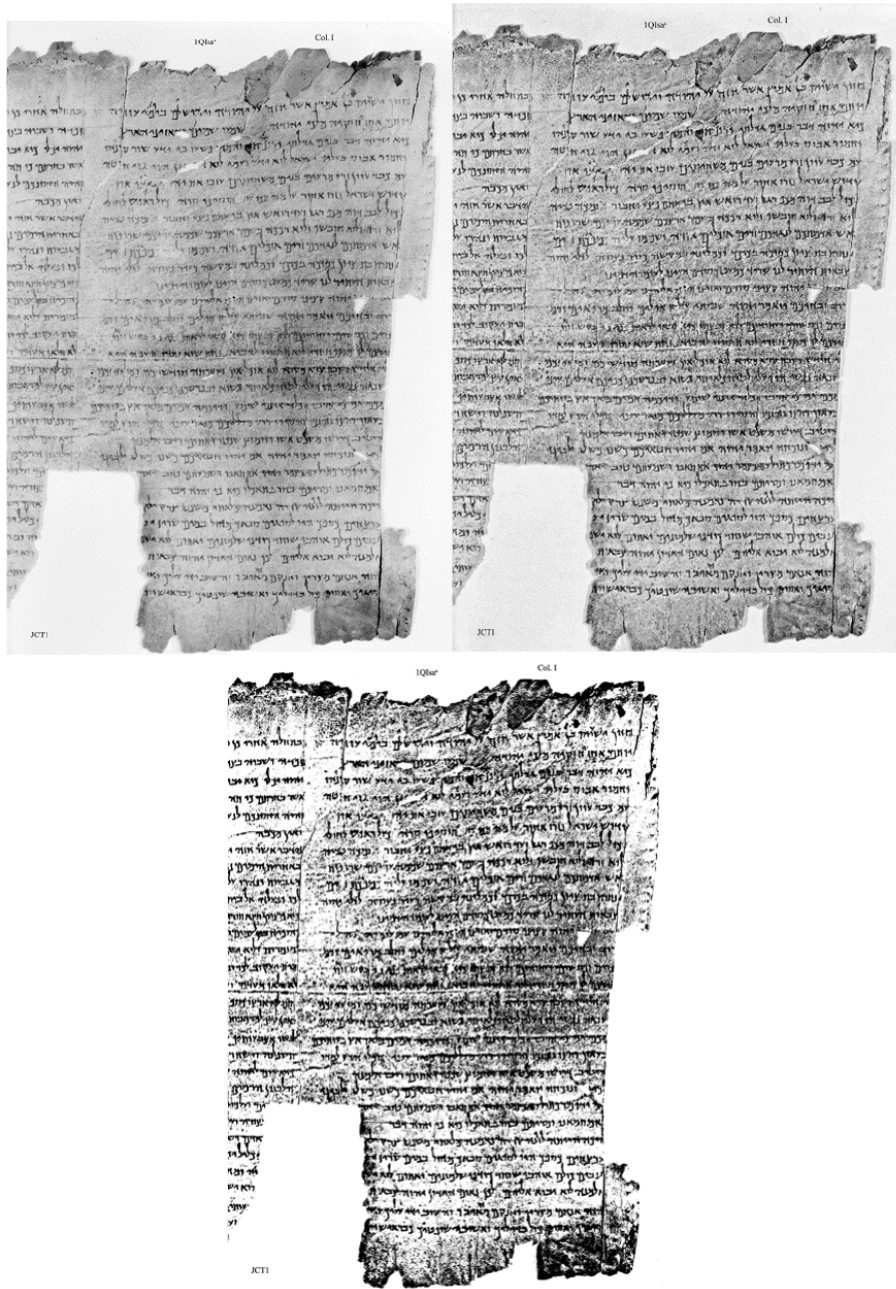


Figure 7.2: The upper left is an image of The Great Isaiah Scroll column 1, which is an example of a DSS image with dark regions in the background. Here it is especially visible in the middle of the image. The upper right is the same image after adaptive histogram equalization and the lower is an image of the equalized image gone through Otsu thresholding. DSS image gotten from: [4]

7.1.2 Morphological Transformations

When it comes to morphological transformations we have found opening and closing the most useful, when it comes to improving the image segmentation and the machine learning accuracy. We found that closing with an elliptical kernel of size 3x3, removes noise in the letters efficiently, as can be seen in Figure 7.3. See Code listing 4.2 for information on how we do that.

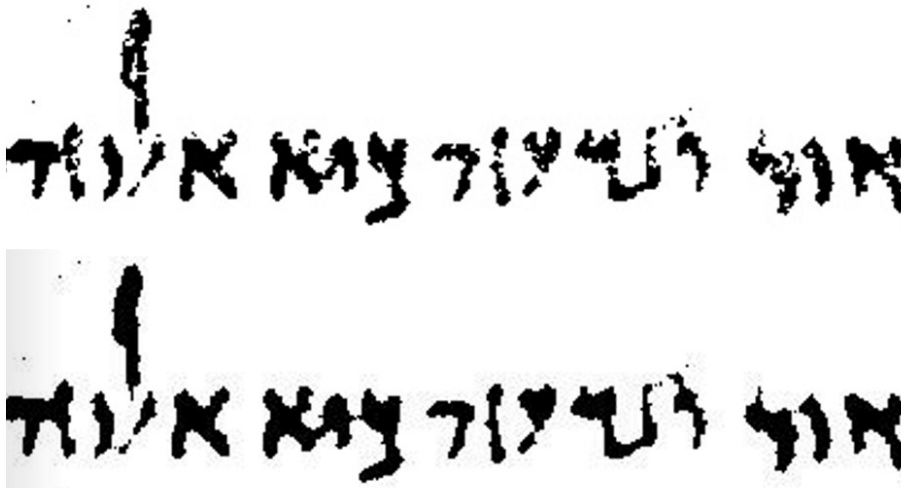


Figure 7.3: Example of closing performed on a DSS image. The first line is without closing and the last line has been closed with an elliptical kernel size 3x3. DSS image gotten from: [4]

We found that opening was a technique we could use after adaptive binarization to remove noise. It can also be used after Otsu thresholding, but it will then remove some of the letters, as seen in Figure 7.4. Some of the letters may also be lost when opening is used after adaptive thresholding, but opening is needed here since there often is a lot of noise. We perform opening almost the same way as we do closing in Code listing 4.2. The only difference is that we switch out `morphologyEx(invertedImg, cv.MORPH_CLOSE, kernel)` with `morphologyEx(invertedImg, cv.MORPH_OPEN, kernel)`.



Figure 7.4: An example of when opening with an elliptical kernel size 3x3 is performed on a DSS image that has undergone Otsu thresholding. Some of the letters are almost completely removed. DSS image gotten from: [4]

7.1.3 Blur and Denoising Methods

Visual Results

Testing if noise removal methods work or not is not that hard. It is often very easy to visually spot the difference between a good and badly denoised image.

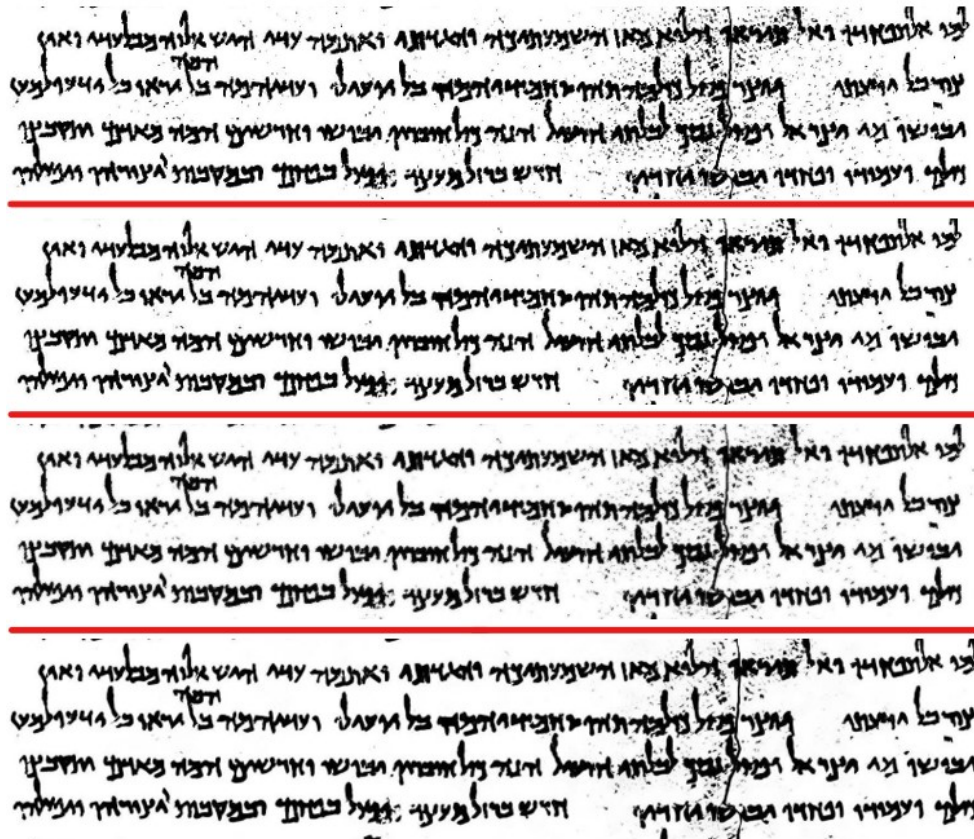


Figure 7.5: Image enhancement results of a section of The Great Isaiah Scroll column 35. From the top: the original Otsu binarized image, the Otsu image gone through median blur, the Otsu image gone through bilateral blur, the Otsu image gone through non-local means denoising. DSS images gotten from: [4]

As we can see from Figure 7.5 the denoising method that visually looks best is non-local means denoising. The denoising methods seem to gradually become better as we move downwards, median blur being the worst, and non-local means denoising being the best. These were the OpenCV methods we used to blur the three last images in Figure 7.5:

```
1 median = cv.medianBlur(imGray, 3)
```

Code listing 7.1: Median blur.


```
1 bilateral_blur = cv.bilateralFilter(img,9,150,150)
```

Code listing 7.2: Bilateral blur.

```
1 means_denoised = cv.fastNLMMeansDenoising(src=img,h=60.0, templateWindowSize=7,
      searchWindowSize=21)
```

Code listing 7.3: Non-local means denoising.

Segmentation Results

We tested the noise removal methods by trying to segment the denoised images using Pytesseract. We could then count how many letters Pytesseract was able to segment on the differently denoised images. The more letters Pytesseract was able to segment, the better the noise removal method. For testing, we used The Great Isaiah Scroll column 35 which had undergone Otsu thresholding. These are the results we got.

Noise removal method	Amount of letters segmented
No noise removal	1382 letters
Median blur	1431 letters
Bilateral blur	1514 letters
Non-local means denoising	1538 letters

Table 7.1: Segmentation results on The Great Isaiah Scroll column 35 using different types of noise removal methods.

Pixel to Noise Ratio (PSNR) Results

When using PSNR to test the different noise removal methods we used a cropped out section of The Great Isaiah Scroll column 35. The same section that can be seen in Figure 7.5. The section had undergone Otsu thresholding.

Noise removal method	PSNR score
Median blur	33.5 dB
Bilateral blur	33.9 dB
Non-local means denoising	34.5 dB

Table 7.2: PSNR results on The Great Isaiah Scroll column 35 using different types of noise removal methods.

The higher the PSNR score the better the noise removal method. The one that gives the highest score is non-local means denoising, which also matches the results from Section 7.1.3. When it comes to the non-local means denoising score it is higher than the one they obtained in [40], which was 25.45 when using non-local

means denoising together with sparse representations. This might be because they had Gaussian noise in their image while we have Salt-and-pepper noise. In [41] they got a PSNR score between 30.5 and 34.6 using the median blur method. These results are very similar to our own.

7.1.4 Image Enhancement Process

When it comes to the different image enhancement methods we can use on a DSS image, there seem to be two different main scenarios. The first scenario is if the image has a clear and mostly white background. If that is the case the best approach seems to be to first do adaptive histogram equalization, Otsu binarization, then closing before finally doing noise removal. If the DSS image has a background with dark areas, like stains, another approach should be taken. Adaptive binarization should then be used on the image, before doing opening then closing and finally noise removal. These two different approaches are illustrated in Figure 7.6.

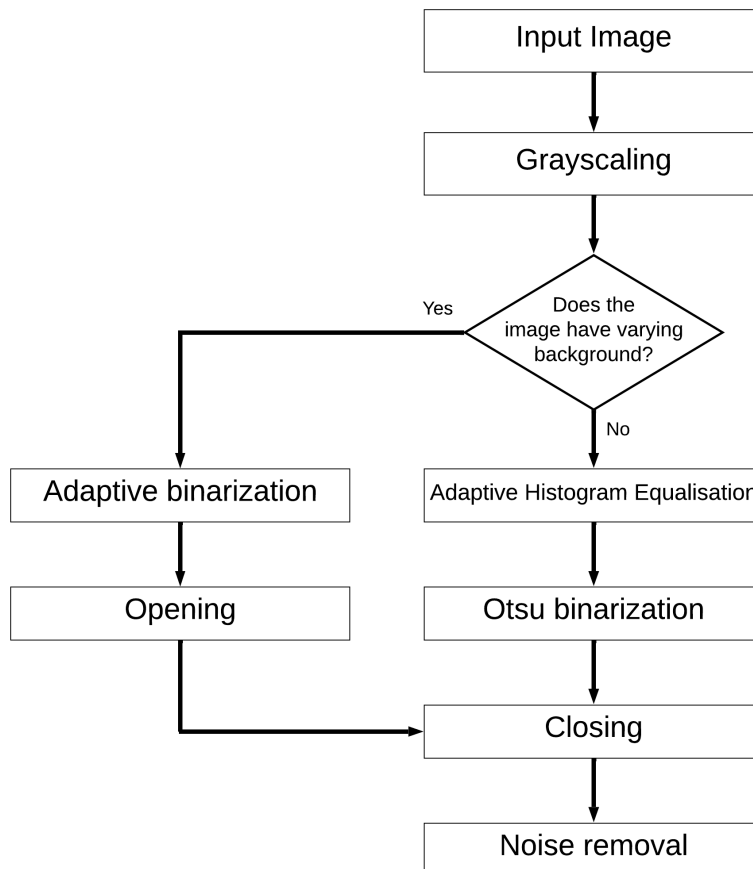


Figure 7.6: Flowchart of image enhancement process.

7.2 Image Segmentation

The figure below shows the resulting image when we draw rectangles over all the segmented letters while using the word splitter and our custom TRAINEDDATA file. The DSS images used in this section are from [4].

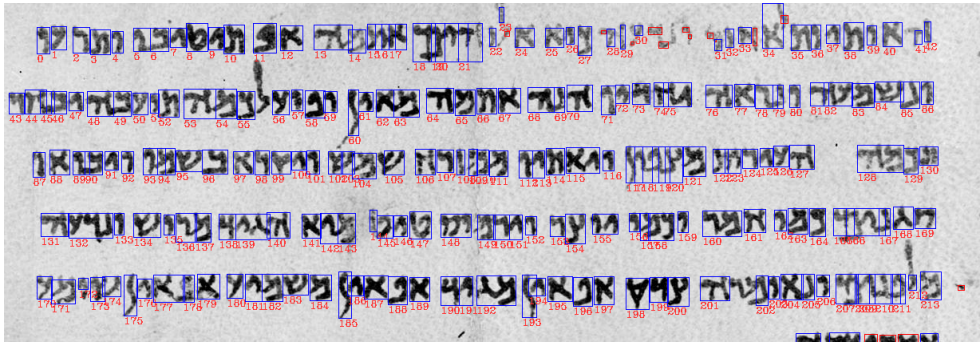


Figure 7.7: The segmenter with the word splitter performed on a paragraph from the Great Isaiah Scroll column 35 using our custom TRAINEDDATA file. Blue rectangles are successful segments while the red rectangles are segments that are either too large or too small. Only the letters surrounded by blue rectangles are saved.

In the sections below we show the results of our binarization methods, our results from Pytesseract with and without our word splitter, our results from Pytesseract with and without our custom TRAINEDDATA file, and our average IoU score.

7.2.1 Binarization Results

We have found out that two different types of binarization should be used for two different types of DSS images. If the DSS image has a mostly white background Otsu binarization seems to be the best method. When the DSS image has varying background adaptive binarization should be used. The varying background is typically when the background is not totally white but has darker areas. The adaptive binarization will create a considerable amount of noise, but it will still perform better on these columns with darker areas than Otsu binarization.

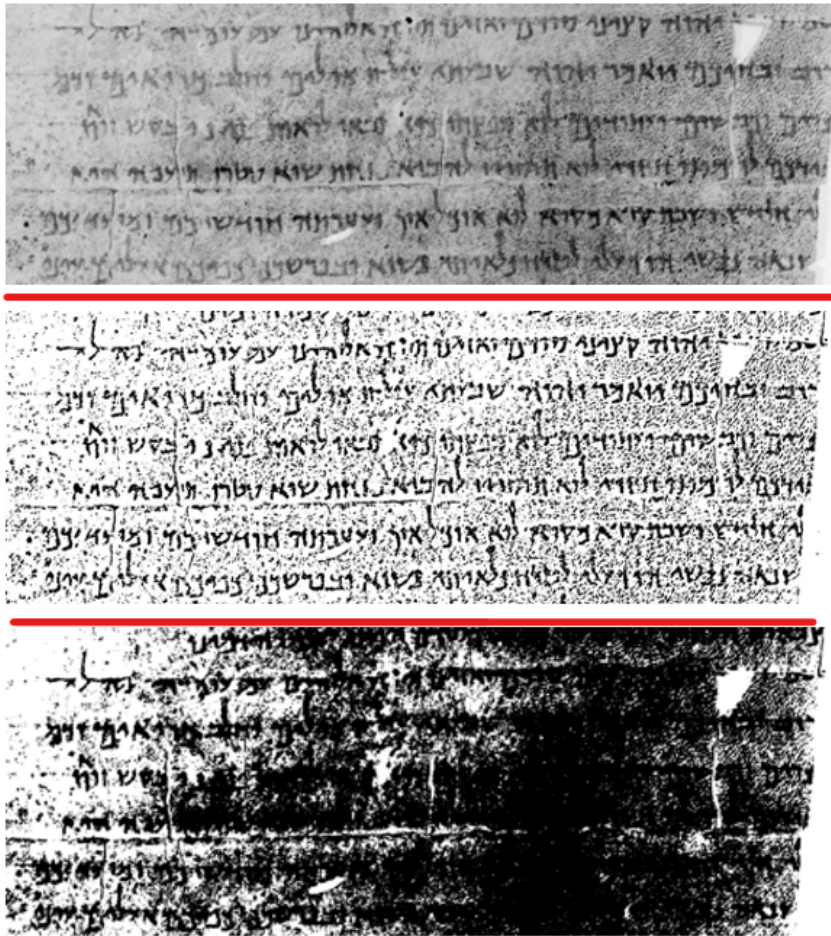


Figure 7.8: This is a paragraph from the Great Isaiah Scroll column 1, which has a lot of darker areas in the background. The upper image is gray-scale, the middle image is the gray-scale image gone through adaptive binarization, while the lower one has gone through Otsu binarization.

As we can see from Figure 7.8, Otsu binarization makes the letters in the paragraph unreadable. Adaptive binarization does create a lot of noise, but some of this can be removed as described in Section 7.1.2 and Section 7.1.3. It gives much better results than the Otsu binarization.

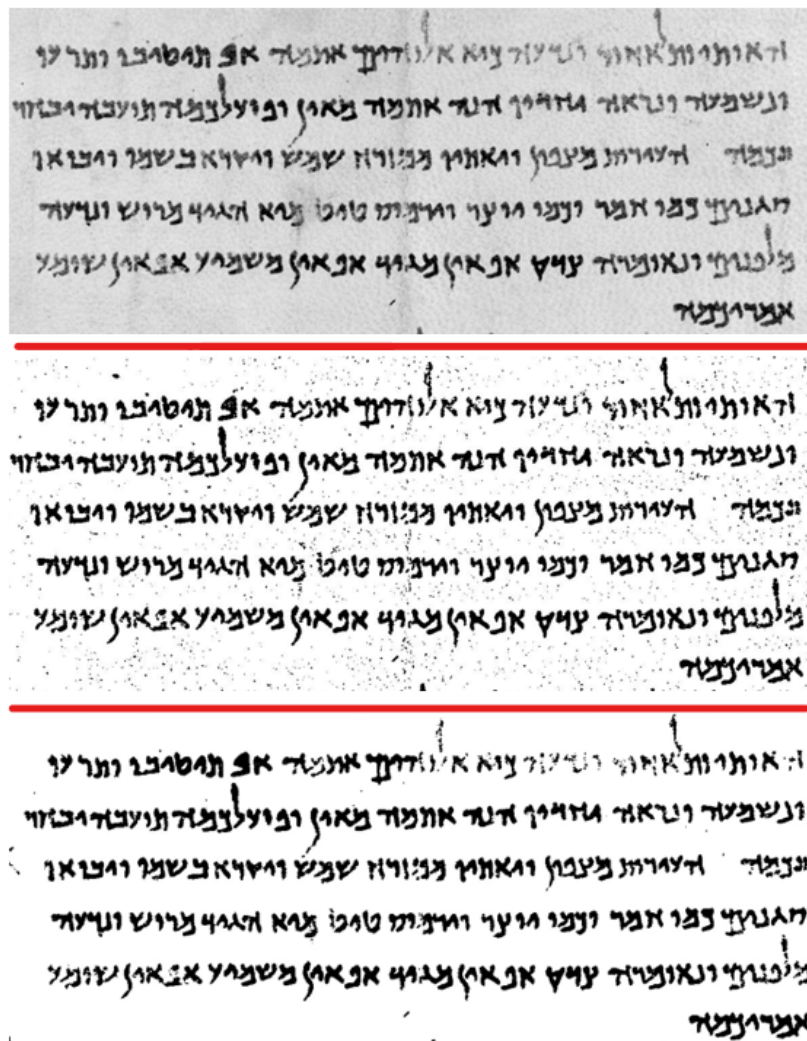


Figure 7.9: This is a paragraph from the Great Isaiah Scroll column 35, which has a clean white background. The upper image is gray-scale, the middle image is the gray-scale image gone through adaptive binarization, while the lower one has gone through Otsu binarization.

Figure 7.9 shows binarization results from a column with a cleaner and whiter background. We can here see that Otsu binarization is the superior method as it does not create as much noise as adaptive binarization.

7.2.2 Pytesseract Results

In Figure 7.10, we can see the results of the segmenter without the word splitter. In this example, almost all of the letters are either individually segmented or segmented into smaller groups. The segmenter struggles segmenting the letter

"Lamed", see Figure 7.11a. In the top row, where we can see a few red rectangles, we can see the segmenter struggle, as seen in Figure 7.11b. The largest problem is that a lot of the letters are grouped into smaller segments, which is why we worked on the word splitter.

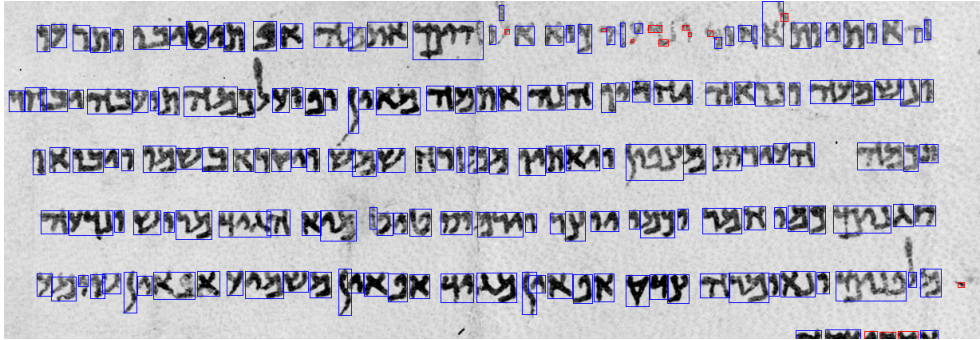
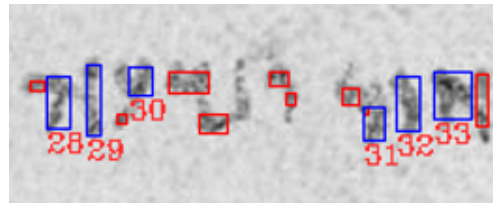


Figure 7.10: The segmenter without the word splitter performed on a paragraph from the Great Isaiah Scroll column 35 using our custom TRAINEDDATA file. Blue rectangles are successful segments while the red rectangles are segments that are either too large or too small.



(a) An example of the letter "Lamed" not being segmented.



(b) Example of the segmenter not segmenting the letters properly.

Figure 7.11: Examples of problems with the segmenter.

7.2.3 Word Splitter Results

The segmenter checks if a wide segment is just a large letter or two thin letters connected. The Figure 4.4 provides an example of why this needs to be checked, and more can be read in Section 4.4.2. The segmenter uses the classifier to check if the segment should be sent to the word splitter. The results of this process and the word splitter can be divided into four groups:

- The word was split successfully.
- The word splitter failed to split the word correctly.
- The classifier returned a high confidence value when classifying an image with multiple letters, causing the code to think that it is a single letter. The word was then not passed to the word splitter.
- The classifier returned a low confidence value when classifying an image

with a single letter, causing the code to believe that it is not a single letter. The image was passed to the word splitter.

Examples of segments that were split correctly

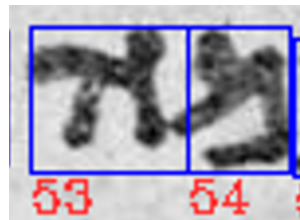


Figure 7.12: The letters "He" and "Mem", number 53 and 54, found in Figure 7.7, were successfully split.



Figure 7.13: The letters "He" and "Nun", number 68 and 69, found in Figure 7.7, were successfully split.



Figure 7.14: The letters "Yod" and "Mem", number 142 and 143, found Figure 7.7, were successfully split.

Examples of segments that were split incorrectly



(a) The letters between 18 and 21, found in Figure 7.7, were unsuccessfully segmented.



(b) The red lines show the initial segmentation points for this segment.

Figure 7.15: These Figures show how the segment was split, and where the initial segmentation points were in the image.

In this example, 7.15a, the word splitter failed to split this segment properly. The initial segmentation points (pixel value points) were: [57, 40, 23, 0], which can be viewed in Figure: 7.15b. The letter all the way to the right, 7.16d, has segmented the columns correctly, but the image contains a lot of white space at the bottom. This has happened because there are a few black pixels at the bottom of the image. When running *cv2.findNonZero* in the *image_cropper* function the few black pixels will be found by the OpenCV method and will treat those pixels as if they are part of the letter above it, which leads to the letter not being cropped correctly. The third letter, 7.16c, had good initial segmentation points, which can be seen in Figure 7.15b, and was close to being segmented correctly, but the classifier never returned a confidence value above 60 percent. The *word_cropper* tried to extend the image to check if it could create a segment that would return a confidence value above 60. The resulting extended image of the letter, which includes some of the letters around it, had a higher confidence value than narrower images of it. The first and second images, 7.16a and 7.16b, contain the single letter "final mem", which means the word splitter has split a single letter into two images.

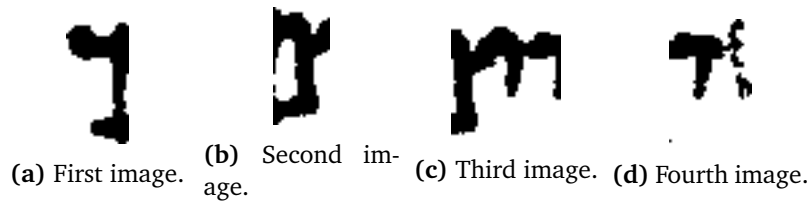
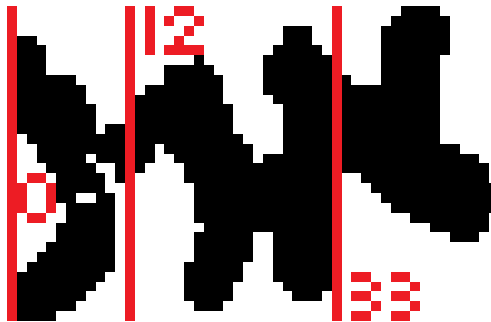


Figure 7.16: Resulting images from splitting this image: 7.15a



(a) The letters between number 15 and 17, found in this Figure 7.7, were unsuccessfully segmented.



(b) The red lines show the segmentation points for this segment.

Figure 7.17: These Figures show how the segment was split, and where the initial segmentation points were in the image.

In this example, 7.17a, only one of the letters was successfully segmented. The initial segmentation points (pixel value points) were $[33, 12, 0]$, which can be viewed in Figure: 7.17b. Figure 7.18 shows the resulting images when splitting this segment. The word splitter incorrectly split the first letter, "Tav" into two images, 7.18a and 7.18b. The third letter "Alef", 7.18c, has been correctly segmen-

ted. As you can see in Figure 7.17b with the initial segmentation points, the initial segmentation point is in the middle of the letter "Alef". The *word_cropper* method returned a low confidence value when trying to crop that letter with that segmentation point. It, therefore, extended the image to the left until the classifier believed the image was an "Alef" with a roughly 80 percent confidence value. This is an example of the classifier helping the word splitter segment letters.

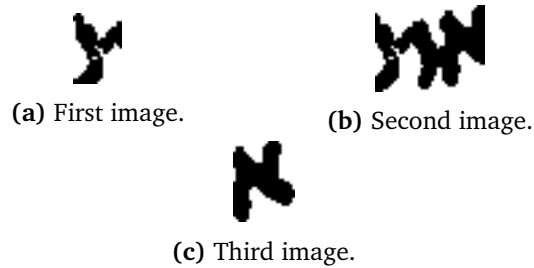


Figure 7.18: Resulting images from splitting this image: 7.17a

Example of a segment that was incorrectly not sent to the word splitter

In this example, 7.19, the segment with multiple letters does not get sent to the word splitter. This happened because the classifier believed the segment was a "He" with a confidence value of ca. 98 percent, making the segmenter believe that the segment is a single letter.

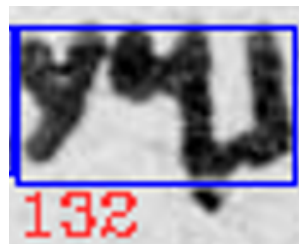


Figure 7.19: This segment contains three letters, but it did not get sent to the word splitter. This can be found in Figure 7.7, by looking for segment number 132.

Example of a segment that was incorrectly sent to the word splitter

Here is an example of a letter being incorrectly sent to the word splitter:

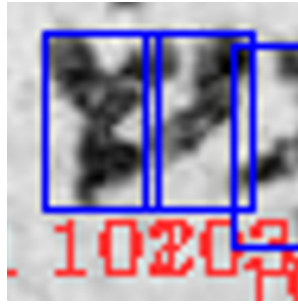
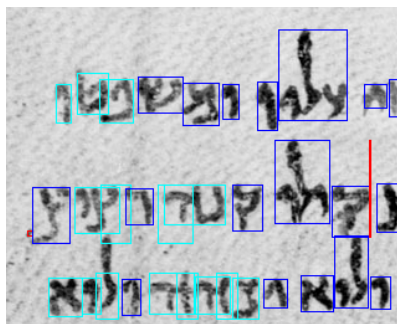


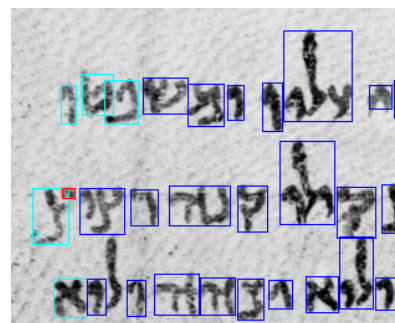
Figure 7.20: The letter "Shin" has been incorrectly split into two. This can be found in Figure 7.7, by looking for the segments between number 101 and 104.

In this example, 7.20, the letter "shin" is split even though it is a single letter. It is a wide letter which means the segmenter needs to check if it is a large letter or two thin connected letters. The classifier returned a confidence value of ca. 66 percent when classifying this segment, which is below our threshold of 90 percent.

7.2.4 Custom TRAINEDDATA Results



(a) Using the modern Hebrew TRAINEDDATA file.



(b) Using our custom TRAINEDDATA file.

Figure 7.21: The results when we draw rectangles over the segmented letters. Dark blue = successful segment and not overlapping any other segments, light blue = successful but overlapping, red means the segment was either too large or too small.

In Figure 7.21, you can see that the custom TRAINEDDATA file removes the unwanted overlapping segments. The unwanted overlapping segments can be found in the second and third lines. Some of these letters have multiple rectangles over them that cover only parts of a letter. Not all overlapping segments are unwanted. The rectangles around the three letters in the top left of the Figures are overlapping, but each rectangle covers each letter entirely only once.

7.2.5 IoU Results

We calculated the average IoU score of the letters we have segmented manually. An image of the letters we have segmented manually can be seen in Figure 7.22. We then took each letter we have segmented manually and calculate the IoU score to a letter that overlaps it from the segmenter. A figure of the letters we have manually segmented with the letters from the segmenter can be viewed in Figure 7.23. The letters from the segmenter can be viewed in Figure 7.7. After calculating the IoU scores we got an average IoU score of ca. 0.464.

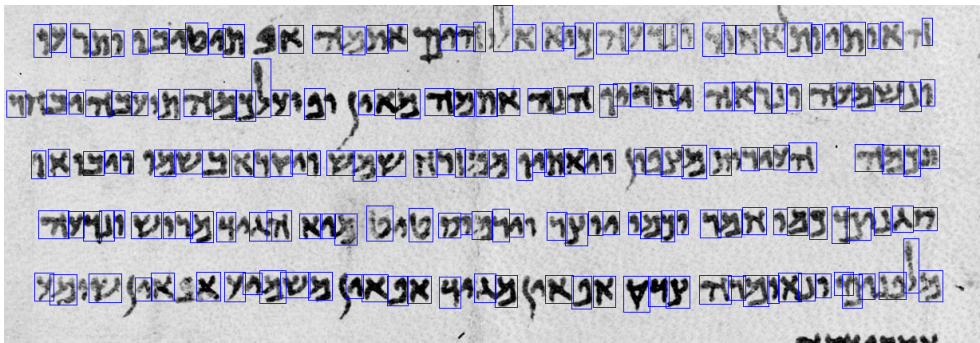


Figure 7.22: The letters we have manually segmented have a blue rectangle around them.

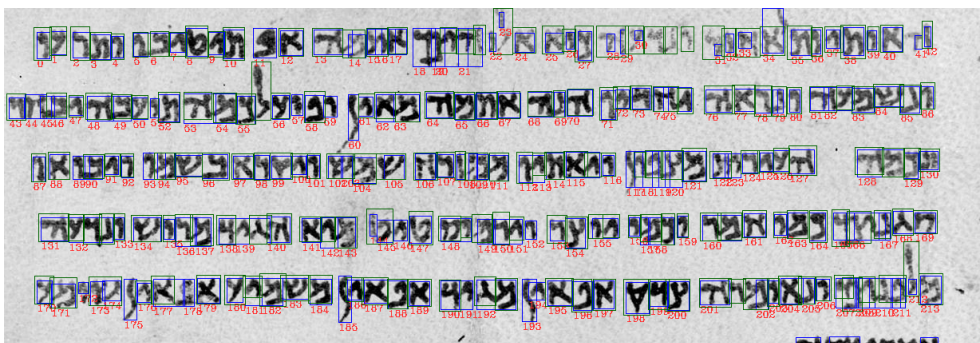


Figure 7.23: The letters we have manually segmented have a dark green rectangle around them while the automatically segmented letters have a blue rectangle around them.

7.3 Machine Learning

	Accuracy	Loss	Epochs
Training	99.9%	0.0026	40
Validation	96.7%	0.1183	40

Table 7.3: Training results our dataset at the final epoch

As seen in Table 7.3, when training our neural network model for 40 epochs, we get an average loss of 0.0026 and an average accuracy of 99.9% over the last epoch. The average validation accuracy is a little lower at 96.7%, but the loss has increased to 0.1183.

	Accuracy	Loss	Epochs
Training	100%	0.0001	21
Validation	98.0%	0.0567	21

Table 7.4: Training results on the MNIST dataset at 21 epochs

Looking at Table 7.4 for comparison, we have the same model trained on the MNIST dataset, which peaked at epoch 21 and achieved at its best an average validation loss of 0.0567 and a validation accuracy of 98.0%. The average training loss and accuracy at this point were 0.0001 and 100%. Looking at the last epoch for this trained model, the average validation loss increased to 0.0843, and the training loss decreased to 0, presumably because it is below the displayed range.

	Accuracy	Epochs
w/o augmentation	92.5%	100
w/ augmentation	93%	100

Table 7.5: Results as described in this paper [47]

We can also compare this to the paper by Sudhakaran Jain [47] mentioned in Section 3.4.4. They achieved results seen in Table 7.5.

Figure 7.24 and 7.25 show the evolution of the average loss and accuracy across the epochs for the models trained on our and the MNIST dataset respectively.

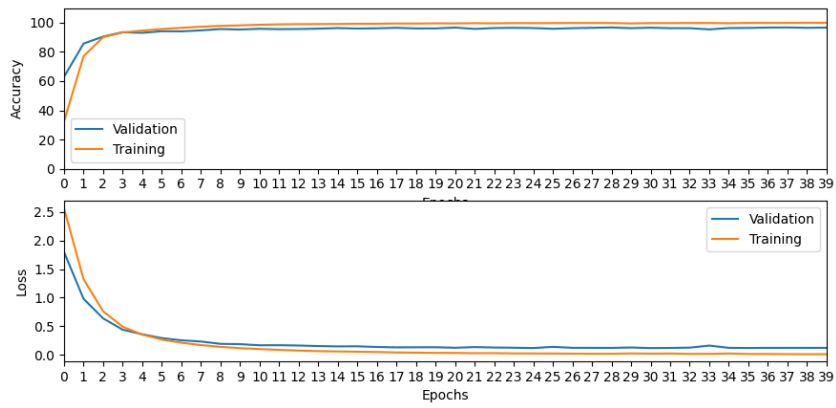


Figure 7.24: Graph showing average loss (bottom graph) and accuracy (top graph) for both training (orange) and validation (blue) through 40 epochs, trained on the DSS dataset

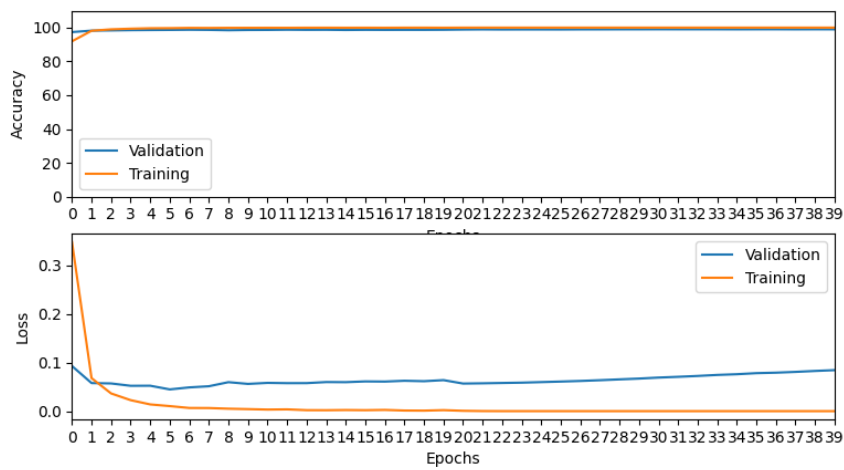


Figure 7.25: Graph showing average loss (bottom graph) and accuracy (top graph) for both training (orange) and validation (blue) through 40 epochs, trained on the MNIST dataset

classes	precision	recall
alef	1	0.97
bet	0.97	0.97
gimel	0.92	1
dalet	0.95	0.9
he	0.98	0.98
vav	0.91	0.93
zayin	0.95	0.94
het	1	0.9
tet	0.97	0.97
yod	0.99	0.97
kaf	1	0.95
lamed	1	1
mem	0.94	0.96
nun	0.93	0.95
samekh	1	0.94
ayin	1	0.98
pe	0.9	0.9
tsadi	0.97	0.98
qof	0.98	0.98
resh	0.9	0.98
shin	1	0.98
tav	0.95	1

Table 7.6: Shows all the precision and recall metrics for all letters

		Predicted Labels																					
		alef	bet	gimel	dalet	he	vav	zayin	het	tet	yod	kaf	lamed	mem	nun	samekh	ayin	pe	tsadi	qof	resh	shin	tav
True Labels	alef	76	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	bet	0	39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	gimel	0	0	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	dalet	0	0	0	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	0	0
	he	0	0	1	0	93	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	vav	0	0	0	0	0	49	3	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
	zayin	0	0	0	0	0	1	61	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
	het	0	0	0	1	0	0	0	48	0	1	0	0	0	0	0	0	0	0	0	1	0	1
	tet	0	0	0	0	0	0	0	0	39	0	0	0	0	0	0	0	0	0	0	0	0	0
	yod	0	0	0	0	0	2	0	0	0	66	0	0	0	0	0	0	0	0	0	0	0	0
	kaf	0	0	0	0	0	0	0	0	0	0	37	0	0	0	0	0	0	0	0	1	0	0
	lamed	0	0	0	0	0	0	0	0	0	0	0	73	0	0	0	0	0	0	0	0	0	0
	mem	0	0	0	0	0	0	0	0	0	0	0	0	46	0	0	0	1	0	0	0	0	0
	nun	0	0	0	0	0	0	0	0	0	0	0	0	0	37	0	0	1	0	0	0	0	0
	samekh	0	0	0	0	0	0	0	0	0	0	0	0	1	0	31	0	0	0	0	0	0	1
	ayin	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	61	0	0	0	0	0	0
	pe	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	35	0	0	0	0	0
	tsadi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	63	0	0	0	0
	qof	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	46	0	0	0
	resh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	61	0	0
	shin	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0
	tav	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	54

Figure 7.26: Shows the confusion matrix for validation in this models last epoch. Rows show the true label, while columns show the predicted label.

	Accuracy	Loss	Epochs
Training	91.0%	0.3932	40
Validation	84.3%	0.5245	40

Table 7.7: Training results with transfer learning while using the model in Table 7.4 as a base

7.3.1 Prediction Results

To test how our model would perform in a real-world scenario, we extracted a set of images for each character in the alphabet from the column 35 [4]. We then chose one image of reasonable quality for each letter and ran them through the model. It classified 20 out of 22 correctly as seen in 7.29, but as seen in 7.27 4 of

these predictions were under 80% confidence, and should therefore be categorized as unsure.

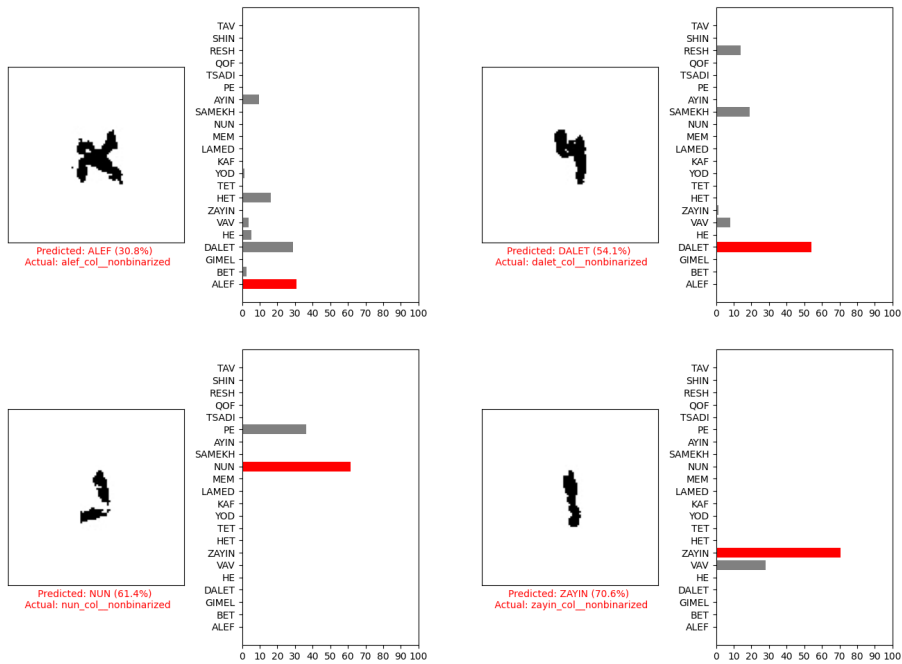


Figure 7.27: Shows the predictions with a confidence score of less than 80%, with a bar graph of the other confidence values to visualize what characters they might have been confused with.

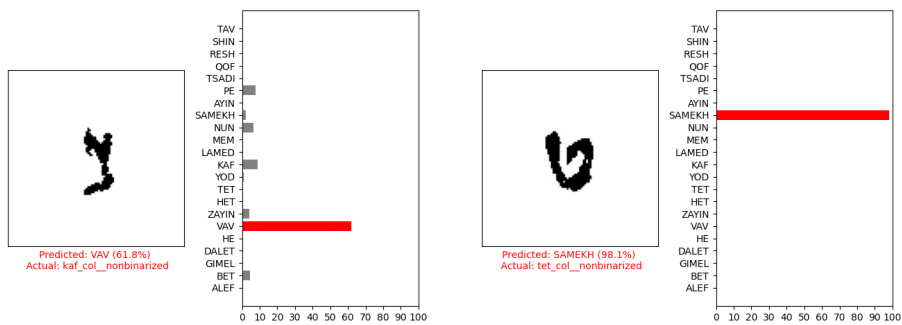


Figure 7.28: Displays the two characters that the classifier didn't classify correctly, with a bar graph of all other confidence values.

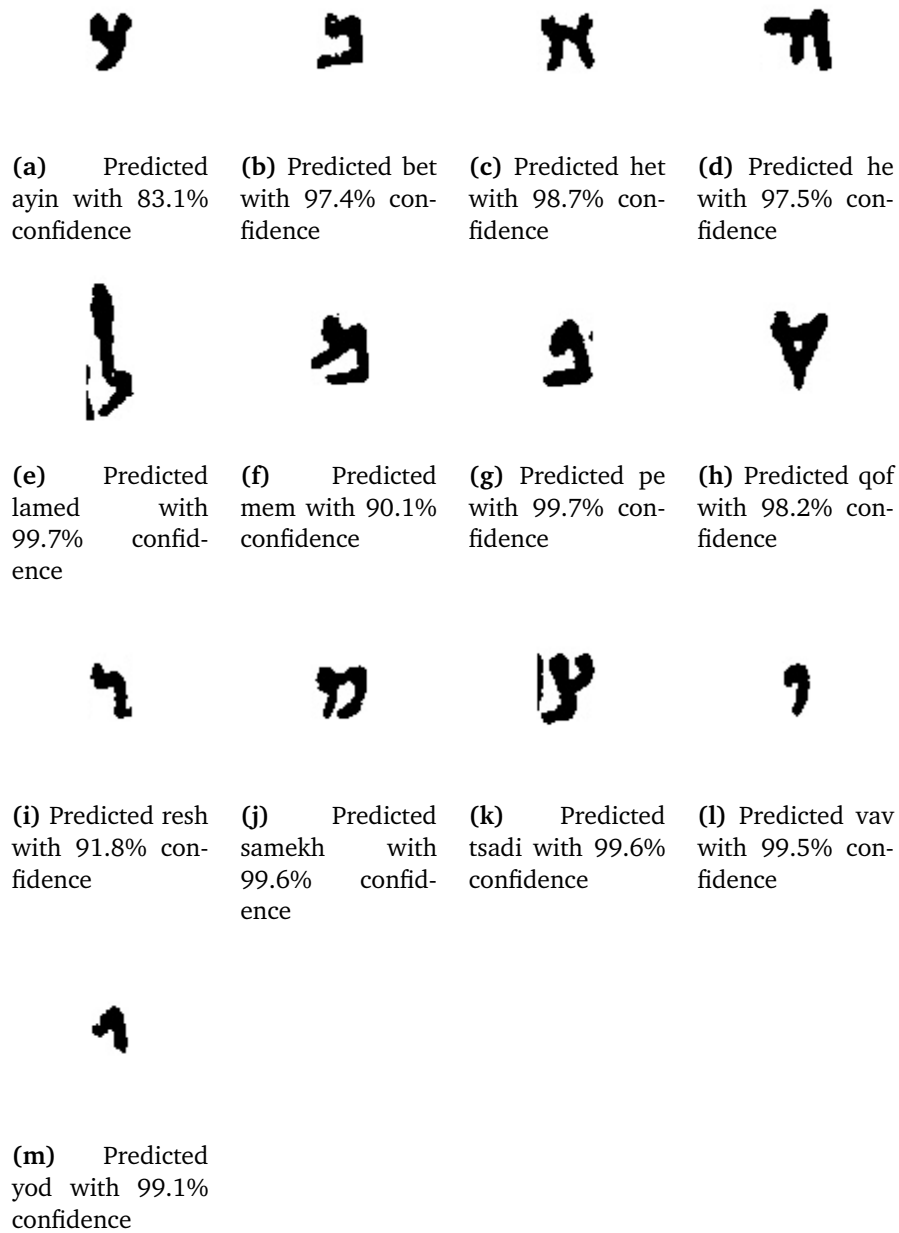


Figure 7.29: Shows all the predictions for characters with a confidence score about 80%

We included Figure 7.30 to show differences between training metrics for different epoch lengths as a basis for why we chose to train our model with 40 epochs.

7.3.2 Comparison between epochs

Epoch Comparison					
Attempt	Number of Epochs	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1	10	0.061308	97.8	0.198547	95.5
2	10	0.068302	98.3	0.183137	95.6
3	10	0.136921	97.8	0.190095	95.1
1	20	0.030963	99.5	0.132799	96.2
2	20	0.587843	99.6	0.118247	96.6
3	20	0.041501	99.5	9.134601	96.2
1	40	0.001996	99.9	0.105825	96.7
2	40	0.002894	99.9	0.114878	97.7
3	40	0.00444	99.9	0.107965	96.7
1	80	0.000438	100	0.12227	96.8
2	80	0.000546	100	0.108612	97.5

Figure 7.30: Comparison for loss and accuracy between different epoch lengths

Chapter 8

Discussion

This chapter contains a discussion of how well we feel our project process has gone and a discussion regarding the results we have achieved throughout the process. Section 8.1 is about our project process and Section 8.2 is about our technical results.

8.1 Project Process

8.1.1 Working Environment

Most of our communication within the group and with our employers and supervisors has happened online. This was somewhat due to *COVID-19* restrictions at the beginning of the project period, and also due to the fact that not all of our group members were present in Gjøvik throughout the whole project period. Another reason is that it was the most convenient for us and we liked this way of working. We were at home able to use our home desktop PCs with extra screens. Although we liked this way of working, it would have been good to possibly meet physically at least once a week, just to change the working environment and possibly increase motivation for the project.

8.1.2 Planning of the Project

We did a good job at the beginning of the project when it comes to making a good project plan. There are certain things however that we feel like we could have agreed upon before we started the project. We should have planned better different standards we were going to use. An example is that we should have agreed upon a standard for naming things like files, folders, variables and methods. Do we use camelCase naming (*imageEnhanced.jpg*) or maybe snake_case (*image_enhanced.jpg*)? We were a long way into the project work before we agreed upon what we wanted to use. For more information on this see Section 6.1. We also struggled with creating a Gantt-diagram we could follow throughout the entire project. In the fourth phase of our project, we diverged from our Gantt-diagram

quite a lot. This happened because certain tasks took longer than predicted, so we did not have time to "classify letters in fragmented scrolls" or "classify authorship and time period". What we did in each phase of our project can be viewed in Section 2.1.1.

8.1.3 Meetings

The meetings with our employers and the meetings with our supervisors were both held on the same day. We first got to have a discussion with our employers showing them what we had done since the last meeting, and then agreeing upon what to do until the next one. Both Sule and Tabita were very good at giving us constructive feedback about our work and at guiding us in the right direction. After that meeting, we got to have a meeting with our supervisors, where we could ask questions about every technical part of the project. At the beginning of the project process, we mostly asked questions about computer vision and machine learning, then later one more about the structure and contents of our report. Marius and Aditya both had a lot of knowledge and experience with computer vision, machine learning and how to write a bachelor thesis. At the beginning of the project process, we had weekly meetings with our employers and supervisors, but later in the project process, we sometimes canceled meetings as we did not feel the need to have them as often.

8.1.4 Development Process

We found the use of agile development and Kanban very useful for this type of project. The Kanban board we used on GitHub gave us a very good overview of the project process. It made it very clear what tasks needed to be done, what tasks were under work and what tasks were done.

8.2 Technical Results

8.2.1 Modifiable and Expandable Code

One of our effect goals, as seen in Section 1.3.3, was that "our solution should be modifiable and expandable by others outside our group". An argument to back up this is that our solution is written only in Python. This makes it easier for others to modify or expand our solution as they only have to deal with one programming language. Our code is also heavily documented in the form of comments in code, this report and two README files. The external packages we have used are also well documented by their creators and community. This makes it much easier for others to understand our code. Furthermore, we could say that our code is modular, where code that deals with one specific functionality is separated from another.

8.2.2 Image Enhancement

In Section 3.4.1 we mention that Dhali et al. in [37] used BiNet to binarize DSS images. Binarization using BiNet creates very little noise, making noise removal methods unnecessary. The reason we did not use BiNet is that we wanted a more simple binarization solution. We also thought it might slow down the process of extracting features from the DSS since it utilizes deep learning, which sometimes can be a time-consuming and heavy process. One of our result goals is after all that "we assume that the time it takes to segment and classify letters on a single image of the DSS should not take more than a couple of minutes, preferably not more than 2 minutes" (Section 1.3.2). If we had utilized BiNet it could have improved our segmentation and classification results by removing a lot of noise from our binarized DSS images.

8.2.3 Image Segmentation

Intersection over Union Results

In Section 7.2.5 we wrote that we got an average IoU score of ca. 0.464. Figure 3.4 shows that a score of 0.4 is a poor score. There are multiple potential reasons why we got such a low score. One of the reasons can be that when we cropped the letters manually, we did not crop as close to the letter as possible. This means that in the manually cropped letters there is a white gap between the letter and the border of the image. The letters returned from the segmenter do not have this gap.

Another potential reason for why we get a low IoU score is the way the code selects the letters we should calculate the IoU score on. The code takes a manually segmented letter and checks if any of the automatically segmented letters overlap it. The letters can be very close to each other, and when we check the overlap between manually and automatically segmented letters, we can potentially calculate the IoU score for the incorrect letters. In Figure 8.1 we can see that the green rectangle around the letter number 14, which is a "Mem", has two blue rectangles overlapping it. The green rectangles show the manually segmented letters, and the blue rectangles show how the segmenter segmented the letters. Since the manually segmented letter has two segments overlapping it, it will calculate the IoU score two times, one for each of the segments from the segmenter.

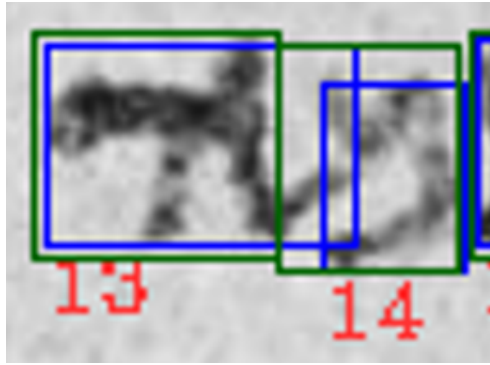


Figure 8.1: A figure showing two letters. The green rectangles show how we manually segmented the letters, and the blue rectangles show how the segmenter segmented the letters. DSS image gotten from: [4]

Word Splitter

In Section 7.2.3 we showed examples of the word splitter succeeding and failing. To the best of our knowledge, there is no work in literature that reports segmentation performance for letters in DSS. When splitting an image containing multiple letters, one of the split images might initially have parts of another letter. Assuming the classifier returns a low confidence value, which it should since the letter contains parts of another letter, the code will only try to extend the width of the image of the letter. This happens because the code does not take into account that the initial split image can be too large. How we implemented our word splitter can be read in Section 4.4.3. An example of this issue can be found in Section 7.2.3 by reading about what happened to the third image, 7.16c, in the results chapter. Future work, when it comes to this issue, is stated in Section 9.2.

Custom TRAINEDDATA File

In Section 7.2.2 we showed the results of Pytesseract with our custom TRAINEDDATA file without the word splitter. In that section we came up with an example of a letter that Pytesseract was struggling to segment, see Figure 7.11a. A reason for this can be our custom TRAINEDDATA file. When we created a font, which was used to create our TRAINEDDATA file, we did not spend enough time making sure that all of the letters in the font had the correct size relative to each other. As seen in Figure 8.2, the letter "Lamed" is only a little taller than the letter "Alef". In Figure 8.3 we can see that the "Lamed" is much taller than the "Alef". The height is one of the characteristics of the lamed and that characteristic might get lost when creating a font without thinking about this issue.

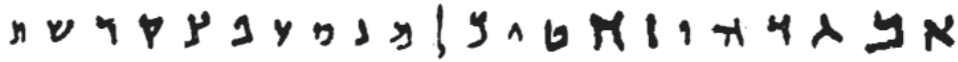


Figure 8.2: A figure showing the letters in our custom font made with letters from the DSS.

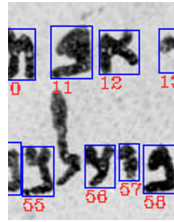


Figure 8.3: A figure showing the letter "lamed", which is between number 55 and 56, and the letter "alef" which is number 12. DSS image gotten from: [4]

8.2.4 Machine Learning

We chose to end our training at 40 epochs because after this point we started to see diminishing returns, as seen in Figure 7.30. Training loss values keep going dramatically down, but validation loss is barely changing. Additionally, the time it takes to train further increases, as well as the strain on our hardware.

The performance at epoch 40, as seen in Table 7.3, is however rather good, and could probably be increased by expanding the dataset further. We can see this by comparing our results to our model trained on the MNIST dataset 7.4, which consists of 70 000 images in total. It is to be assumed that the complexity of our problem is greater with 22 classes instead of 10, which means that our dataset should ideally be quite large. We can also compare our results to the results as described in the paper by Sudhakaran Jain[47], which can also be seen in Table 7.5. Both approaches use a similar CNN approach, but they have an additional 5 classes. This difference in training results could be because of these additional letters, or because our dataset was potentially bigger.

Lastly, we can see in Table 7.7 that our attempt to leverage pre-trained models with the use of transfer learning did not work, as this method performed worse than our standard method without any transfer learning. This could potentially be a result of a number of factors, including a poor setup of the model and an insufficient number of epochs. Most likely, this poor performance is the result of the model learning irrelevant features from the MNIST dataset, or not learning enough relevant features from our dataset.

While the use of binarization to create binary images results in cleaner images, with a distinct letter shape, it might not be the best solution for the problem. In

the process of binarization and thresholding, some details and features may get lost, so it could be beneficial to use grayscale images instead. These would retain much more information about the character shape, while also making it less likely to clip out details through the thresholding. The problem with this approach is the way we resize the images to be the same size, as the original background would remain, while the 'new' background around the letter would be pure white. There is a potential solution to this problem, that we did not have time to try, but includes removing the background from the letter image, but retaining the whole letter in a grayscale format.

Another improvement we could have done is the inclusion of a RNN structure, which has been used by some other OCR based approaches and is often used in language translation tasks. This method essentially gives the neural network some memory, and we believe this could potentially be used to more efficiently classify a word by feeding them into the model as a series of characters.

It is worth quickly mentioning two approaches that we tried, but did not end up including because of how little they affected our performance. The first approach was an attempt to improve the model by splitting it into multiple models, where each model was trained on different subsets of the dataset, in addition to one model trained on the aggregate of the dataset. The idea was that each of these models could specialize in characters that looked similar. This resulted in some of the models doing exceedingly well, while the others did terribly; ergo, it was on average as good as the previous approach. The second approach was an attempt to expand the dataset by using Generative Adversarial Networks (GAN) to generate new samples we could use to train our model. Such networks require a much larger dataset than normal models, so in our case, this would not and did not work.

Chapter 9

Conclusion

Our ultimate project goal was to build a system for recognizing handwritten text using machine learning on intact DSS. We would firstly do image pre-processing (if that was needed), then segment the letters in the images and finally use machine learning to recognize the letters.[2] In this chapter we go through our project goals and possible future work.

9.1 Goals Achieved

We have created a solution for extracting and learning the features from the DSS images, by classifying the different letters on the DSS images. This fulfills our first result goal made for our solution mention in Section 1.3.2. Furthermore, as seen in the Table 7.3 in Section 7.3 our average training and validation accuracy is 99.9% and 96.7%, which is well over our goal of over 90%. If we go by our classification tests in this Section 7.3.1, we would get an accuracy of 90.91%, which is also slightly above our goal. Our average IoU score is 0.464, as shown in Section 7.2.5, which is below our goal of having a score over 0.5. Our next goal considered running times when it comes to our segmentation and classification on a single column of the DSS. As seen in Section 5.4.3 our running times were on average around 1 minute, which is way below our goal of fewer than 2 minutes. We wanted our PSNR value to be over 30 dB for our noise removal methods, as seen in Section 1.3.2. As seen in Section 7.1.3, this was achieved by all of our methods. We were also able to make an easy and intuitive user interface that can be read more about in Section 5.4.2.

9.2 Future Work

One of the things our group would like to work more on is to structure our code-base a bit better. There is some code that could have been cleaner and better structured. Another thing is how the code is run. We could have made it easier for users to run certain parts of our code in the command line. Furthermore, we

could have expanded our dataset to both include more of the letters already in it and also add final letters as additional classes. We could also use augmentation to expand our dataset so that our machine learning model would be able to classify letters that are skewed, upside down, or sideways. When it comes to the image enhancement part, we would like to work more on damaged and/or fragmented DSS as we mostly worked on whole DSS in good shape. We also could have used morphological transformations for removing parts of other letters in our letter segments, to further increase the accuracy of our model. For future work on the user interface see Section 5.4.4.

When it comes to the image segmentation, we would like to improve our custom TRAINEDDATA file for Pytesseract, by training more images and including the final letters. This can potentially improve the number of successful segments and reduce the number of segments that include multiple letters. We would also like to create a way to automatically set the minimum letter width distance constant. The size of the letter might be different in other images, which leads to a tedious process of manually checking the width of the thinnest letter for each image to set that constant.

Another thing we would like to spend more time on is the word splitter. In Section 8.2.3 under "Word Splitter" we discussed how the code will only try to extend the width of a badly segmented letter, not shrink the width. The word splitter will also only extend the cropped letter equally both ways if the letter is in the middle of a word, even though we might only want to extend/shrink the image in one direction. In future work, it would be important for the word splitter to be able to either expand/shrink the image in one or more directions to ensure an optimal segmentation.

As mentioned in Section 8.2.4, we would like to try using grayscale images instead of binary images, as a way to improve the model. Additionally, we would like to see if implementing a RNN structure could improve the model, as well as testing with deeper network structures.

Our employer is expecting to use our results to further help them in their work for date and authorship identification of the DSS.

Bibliography

- [1] J. The Isreali Museum. 'Browsing manuscripts.' (), [Online]. Available: https://www.deadseascrolls.org.il/explore-the-archive/search#q=script_language_parent_en:'Hebrew' (visited on 25/03/2022).
- [2] T. Tobing, 'Handwritten text recognition on the dead sea scrolls,' 2022.
- [3] J. The Isreali Museum. 'Learn about the scrolls.' (), [Online]. Available: <https://www.deadseascrolls.org.il/learn-about-the-scrolls/introduction> (visited on 18/02/2022).
- [4] J. C. T. F. M. C. D. N. F. J. A. Sanders, *Scrolls from Qumrân cave I*. Jerusalem, Albright Institute of Archaeological Research and the Shrine of Book 1974, 1974. [Online]. Available: <https://archive.org/details/qumran>.
- [5] J. Jordan. 'Evaluating image segmentation models.' (), [Online]. Available: <https://www.jeremyjordan.me/evaluating-image-segmentation-models/> (visited on 23/01/2022).
- [6] A. Saha. 'Python | peak signal-to-noise ratio (psnr).' (), [Online]. Available: <https://www.geeksforgeeks.org/python-peak-signal-to-noise-ratio-psnr/> (visited on 21/01/2022).
- [7] E. Tov, *Scribal Practices and Approaches Reflected in the Texts Found in the Judean Desert*. Brill, 2004.
- [8] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Pearson Education, 2008.
- [9] B. J. Nordølum, E. O. Lavik, K. A. D. Haugen and T.-R. T. Kvalvaag. 'Arts-gjenkjenning av fisk.' (), [Online]. Available: <https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2777966> (visited on 15/02/2022).
- [10] S. Haldar, 'Chapter 6 - photogeology, remote sensing and geographic information system in mineral exploration,' in *Mineral Exploration*, S. Haldar, Ed., Boston: Elsevier, 2013, pp. 95–115, ISBN: 978-0-12-416005-7. DOI: <https://doi.org/10.1016/B978-0-12-416005-7.00006-4>. [Online]. Available: <https://www.sciencedirect.com/topics/earth-and-planetary-sciences/image-enhancement>.
- [11] OpenCV. 'Histograms - 2: Histogram equalization.' (), [Online]. Available: https://docs.opencv.org/3.4/d2/d74/tutorial_js_histogram_equalization.html (visited on 29/03/2022).

- [12] A. M. Hambal, D. Z. Pei and F. L. Ishabailu. 'Image noise reduction and filtering techniques.' (), [Online]. Available: <https://www.ijsr.net/archive/v6i3/25031706.pdf> (visited on 16/02/2022).
- [13] OpenCV. 'Morphological transformations.' (), [Online]. Available: https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html (visited on 29/03/2022).
- [14] S. S. Rawat, A. Sharma and R. Gusain, 'Analysis of image preprocessing techniques to improve ocr of garhwali text obtained using the hindi tesseract model,'
- [15] OpenCV. 'Smoothing images.' (), [Online]. Available: https://docs.opencv.org/3.4/d4/d13/tutorial_py_filtering.html (visited on 19/02/2022).
- [16] A. Buades, B. Coll and J.-M. Morel, 'Non-Local Means Denoising,' *Image Processing On Line*, vol. 1, pp. 208–212, 2011, https://doi.org/10.5201/ipol.2011.bcm_nlm.
- [17] MathWorks. 'What is image segmentation?' (), [Online]. Available: <https://se.mathworks.com/discovery/image-segmentation.html> (visited on 16/03/2022).
- [18] Muthukrishnan. 'Otsu's method for image thresholding explained and implemented.' (), [Online]. Available: <https://muthu.co/otsus-method-for-image-thresholding-explained-and-implemented/> (visited on 28/03/2022).
- [19] tutorialspoint. 'Opencv - adaptive threshold.' (), [Online]. Available: https://www.tutorialspoint.com/opencv/opencv_adaptive_threshold.htm (visited on 22/04/2022).
- [20] M. -. S. by thinning of image. 'Otsu's method for image thresholding explained and implemented.' (), [Online]. Available: <https://www.geeksforgeeks.org/mahotas-skeletonization-by-thinning-of-image/> (visited on 29/03/2022).
- [21] OpenCV. 'Canny edge detection.' (), [Online]. Available: https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html (visited on 20/04/2022).
- [22] V. S. Subramanyam. 'Iou (intersection over union).' (), [Online]. Available: <https://medium.com/analytics-vidhya/iou-intersection-over-union-705a39e7acef> (visited on 07/05/2022).
- [23] A. Rosebrock. 'File:intersection over union - poor, good and excellent score.png.' (), [Online]. Available: https://commons.wikimedia.org/wiki/File:Intersection_over_Union_-_poor,_good_and_excellent_score.png (visited on 09/05/2022).
- [24] I. C. Education. 'Machine learning.' (), [Online]. Available: <https://www.ibm.com/cloud/learn/machine-learning> (visited on 22/04/2022).
- [25] I. C. Education. 'Neural networks.' (), [Online]. Available: <https://www.ibm.com/cloud/learn/neural-networks> (visited on 14/04/2022).

- [26] bfortuner (git). ‘Neural networks concepts.’ (), [Online]. Available: https://ml-cheatsheet.readthedocs.io/en/latest/nn_concepts.html (visited on 14/04/2022).
- [27] R. Atienza, *Advanced Deep Learning with Keras*. Packt Publishing Ltd., 2018.
- [28] I. C. Education. ‘Convolutional neural networks.’ (), [Online]. Available: <https://www.ibm.com/cloud/learn/convolutional-neural-networks> (visited on 21/04/2022).
- [29] OpenSource. ‘What is python?’ (), [Online]. Available: <https://opensource.com/resources/python> (visited on 15/02/2022).
- [30] OpenCV. ‘Opencv.’ (), [Online]. Available: <https://opencv.org/> (visited on 15/02/2022).
- [31] LabelImg. ‘Labelimg.’ (), [Online]. Available: <https://github.com/tzutalin/labelImg> (visited on 17/02/2022).
- [32] Google *et al.* ‘Tesseract user manual.’ (), [Online]. Available: <https://tesseract-ocr.github.io/tessdoc/Home.html> (visited on 29/03/2022).
- [33] zdenop. ‘Qt-box-editor.’ (), [Online]. Available: <https://zdenop.github.io/qt-box-editor/> (visited on 29/03/2022).
- [34] S. Chintala. ‘Deep learning with pytorch: A 60 minute blitz.’ (), [Online]. Available: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html (visited on 20/02/2022).
- [35] T. Contributors. ‘Pytorch documentation.’ (), [Online]. Available: <https://pytorch.org/docs/stable/index.html#pytorch-documentation> (visited on 15/02/2022).
- [36] A. E. Harraj and N. Raissouni, ‘Ocr accuracy improvement on document images through a novel pre-processing approach,’ *arXiv preprint arXiv:1509.03456*, 2015.
- [37] M. A. Dhali, J. W. de Wit and L. Schomaker, ‘Binet: Degraded-manuscript binarization in diverse document textures and layouts using deep encoder-decoder networks,’ *arXiv preprint arXiv:1911.07930*, 2019.
- [38] T. Reynolds, M. A. Dhali and L. Schomaker, ‘Image-based material analysis of ancient historical documents,’ *arXiv preprint arXiv:2203.01042*, 2022.
- [39] M. A. Dhali, C. N. Jansen, J. W. De Wit and L. Schomaker, ‘Feature-extraction methods for historical manuscript dating based on writing style development,’ *Pattern Recognition Letters*, vol. 131, pp. 413–420, 2020.
- [40] N. Nayef, P. Gomez-Krämer and J.-M. Ogier, ‘Deblurring of document images based on sparse representations enhanced by non-local means,’ in *2014 22nd International Conference on Pattern Recognition*, IEEE, 2014, pp. 4441–4446.

- [41] M. Sukassini and T. Velmurugan, 'Noise removal using morphology and median filter methods in mammogram images,' in *The 3rd International Conference on Small and Medium Business*, 2016, pp. 413–419.
- [42] G. Levi, P. Nisnevich, A. Ben-Shalom, N. Dershowitz and L. Wolf. 'A method for segmentation, matching and alignment of dead sea scrolls.' (), [Online]. Available: <https://courses.cs.tau.ac.il/~wolf/papers/dsscrolls.pdf> (visited on 22/04/2022).
- [43] F. ., S. Madenda, E. Ernastuti, R. Widodo and R. Rodiah, 'Cursive handwriting segmentation using ideal distance approach,' *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 7, p. 2863, Oct. 2017. DOI: 10.11591/ijece.v7i5.pp2863-2872.
- [44] M. A. Dhali, C. N. Jansen, J. W. de Wit and L. Schomaker, 'Feature-extraction methods for historical manuscript dating based on writing style development,' *Pattern Recognition Letters*, vol. 131, pp. 413–420, 2020, ISSN: 0167-8655. DOI: <https://doi.org/10.1016/j.patrec.2020.01.027>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167865520300386>.
- [45] D. Coquenot, C. Chatelain and T. Paquet, 'End-to-end handwritten paragraph text recognition using a vertical attention network,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2022. DOI: 10.1109/tpami.2022.3144899. [Online]. Available: <https://doi.org/10.1109%2Ftpami.2022.3144899>.
- [46] A. Shonenkov, D. Karachev, M. Novopoltsev, M. Potanin and D. Dimitrov, *Stackmix and blot augmentations for handwritten text recognition*, 2021. DOI: 10.48550/ARXIV.2108.11667. [Online]. Available: <https://arxiv.org/abs/2108.11667>.
- [47] S. Jain, 'A handwriting recognition system for dead sea scrolls,' 2020.
- [48] S. Sudholt and G. A. Fink, 'Phocnet: A deep convolutional neural network for word spotting in handwritten documents,' in *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2016, pp. 277–282. DOI: 10.1109/ICFHR.2016.0060.
- [49] A. Lawler. 'Who wrote the dead sea scrolls?' (), [Online]. Available: <https://www.smithsonianmag.com/history/who-wrote-the-dead-sea-scrolls-11781900/> (visited on 14/02/2022).
- [50] OpenCV. 'Denoising.' (), [Online]. Available: https://docs.opencv.org/4.x/d1/d79/group_photo_denoise.html#ga4c6b0031f56ea3f98f768881279ffe93 (visited on 16/02/2022).
- [51] RiteshKH. 'Cursive_handwriting_recognition.' (), [Online]. Available: https://github.com/RiteshKH/Cursive_handwriting_recognition (visited on 01/05/2022).

- [52] N. Reddy. 'Skeletonization in python using opencv.' (), [Online]. Available: <https://medium.com/analytics-vidhya/skeletonization-in-python-using-opencv-b7fa16867331> (visited on 07/05/2022).
- [53] S. Manoj and L. Mhd. 'How to create traineddata file for tesseract 4.1.0.' (), [Online]. Available: <https://stackoverflow.com/questions/55036633/how-to-create-traineddata-file-for-tesseract-4-1-0> (visited on 03/05/2022).
- [54] B. Zareba. 'How to prepare training files for tesseract ocr and improve characters recognition?' (), [Online]. Available: <https://pretius.com/blog/ocr-tesseract-training-data/> (visited on 03/05/2022).
- [55] S. Srivastava. 'Check if two rectangles overlap in java.' (), [Online]. Available: <https://www.baeldung.com/java-check-if-two-rectangles-overlap> (visited on 09/05/2022).
- [56] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, 'Gradient-based learning applied to document recognition,' 1988.
- [57] 'Gradient-based learning applied to document recognition,' *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [58] M. A. Dhali., S. He., M. Popović., E. Tigchelaar. and L. Schomaker., 'A digital palaeographic approach towards writer identification in the dead sea scrolls,' in *Proceedings of the 6th International Conference on Pattern Recognition Applications and Methods - ICPRAM*, INSTICC, SciTePress, 2017, pp. 693–702, ISBN: 978-989-758-222-6. DOI: 10.5220/0006249706930702.
- [59] PyPI. 'Pyqt5 5.15.6.' (), [Online]. Available: <https://pypi.org/project/PyQt5/> (visited on 18/03/2022).
- [60] A. J. Parmar. 'Lecture 8: Fundamentals of interaction design and user interface.' ()

Appendix A

Standard Agreement



Norwegian University of Science and Technology

STANDARD AGREEMENT

on student works carried out in cooperation with an external organization

The agreement is mandatory for student works such as master's thesis, bachelor's thesis or project assignment (hereinafter referred to as works) at NTNU that are carried out in cooperation with an external organization.

Explanation of terms

Copyright

Is the right of the creator of a literary, scientific or artistic work to produce copies of the work and make it available to the public. A student thesis or paper is such a work.

Ownership of results

Means that whoever owns the results decides on these. The basic principle is that the student owns the results from their own student work. Students can also transfer their ownership to the external organization.

Right to use results

The owner of the results can give others a right to use the results – for example, the student gives NTNU and the external organization the right to use the results from the student work in their activities.

Project background

What the parties to the agreement bring with them into the project, that is what each party already owns or has rights to and which is used in the further development of the student's work. This may also be material to which third parties (who are not parties to the agreement) have rights.

Delayed publication (embargo)

Means that a work will not be available to the public until a certain period has passed; for example, publication will be delayed for three years. In this case, only the supervisor at NTNU, the examiners and the external organization will have access to the student work for the first three years after the student work has been submitted.

1. Contracting parties

<p>The Norwegian University of Science and Technology (NTNU)</p> <p>Department:</p> <p>Institutt for datateknologi og informatikk</p>
<p>Supervisor at NTNU:</p> <p>Aditya Suneel Sole</p> <p>Marius Pedersen</p> <p>email / telephone:</p>

aditya.sole@ntnu.no / 94165542

marius.pedersen@ntnu.no / 93634385

External organization:

Contact person, email address and telephone number of the external organization:

Sule Yildirim Yayilgan, sule.yildirim@ntnu.no, 46623172

Tabita Anggraini Meilita, tabita.tobing@ntnu.no, +62 852 176 976 52

Students:

Erik Dale

Yeshi Jampel Pursley

Håvard Østli Fjørkenstad

Date of birth:

06.01.2000 (Yeshi)

07.01.2000 (Erik)

01.02.2000 (Håvard)

The parties are responsible for clearing any intellectual property rights that the student, NTNU, the external organization or third party (which is not a party to the agreement) has to project background before use in connection with completion of the work. Ownership of project background must be set out in a separate annex to the agreement where this may be significant for the completion of the student work.

2. Execution of the work

The student is to complete: (Place an X)

A master's thesis	
A bachelor's thesis	X
A project assignment	
Another student work	

Start date: 01.01.2022
Completion date: 20.05.2022

The working title of the work is:

Image Content\Hand Writing Analysis of the Dead Sea Scrolls for Provenance

The responsible supervisor at NTNU has the overarching academic responsibility for the design and approval of the project description and the student's learning.

3. Duties of the external organization

The external organization must provide a contact person who has the necessary expertise to provide the student with adequate guidance in collaboration with the supervisor at NTNU. The external contact person is specified in Section 1.

The purpose of the work is to carry out a student assignment. The work is performed as part of the programme of study. The student must not receive a salary or similar remuneration from the external organization for the student work. Expenses related to carrying out the work must be covered by the external organization. Examples of relevant expenses include travel, materials for building prototypes, purchasing of samples, tests in a laboratory, chemicals. The student must obtain clearance for coverage of expenses with the external organization in advance.

The external organization must cover the following expenses for carrying out the work:

No

Coverage of expenses for purposes other than those listed here is to be decided by the external organization during the work process.

4. The student's rights

Students hold the copyright to their works [2]. All results of the work, created by the student alone through their own efforts, is owned by the student with the limitations that follow from sections 5, 6 and 7 below. The right of ownership to the results is to be transferred to the external organization if Section 5 b is checked or in cases as specified in Section 6 (transfer in connection with patentable inventions).

In accordance with the Copyright Act, students always retain the moral rights to their own literary, scientific or artistic work, that is, the right to claim authorship (the right of attribution) and the right to object to any distortion or modification of a work (the right of integrity).

A student has the right to enter into a separate agreement with NTNU on publication of their work in NTNU's institutional repository on the Internet (NTNU Open). The student also has the right to publish the work or parts of it in other connections if no restrictions on the right to publish have been agreed on in this agreement; see Section 8.

5. Rights of the external organization

Where the work is based on or further develops materials and/or methods (project background) owned by the external organization, the project background is still owned by the external organization. If the student is to use results that include the external organization's project background, a prerequisite for this is that a separate agreement on this has been entered into between the student and the external organization.

Alternative a) (Place an X) General rule

	The external organization is to have the right to use the results of the work
--	---

This means that the external organization must have the right to use the results of the work in its own activities. The right is non-exclusive.

Alternative B) (Place an X) Exception

X	The external organization is to have the right of ownership to the results of the task and the student's contribution to the external organization's project
---	--

Justification of the external organization's need to have ownership of the results transferred to it:

Because this work is carried out within the scope of the lying pen of scribes project founded by the research council of Norway.

6. Remuneration for patentable inventions

If the student, in connection with carrying out the work, has achieved a patentable invention, either alone or together with others, the external organization can claim transfer of the right to the invention to itself. A prerequisite for this is that exploitation of the invention falls within the external organization's sphere of activity. If so, the student is entitled to reasonable remuneration. The remuneration is to be determined in accordance with Section 7 of the Employees' Inventions Act. The provisions on deadlines in Section 7 apply correspondingly.

7. NTNU's rights

The submitted files of the work, together with appendices, which are necessary for assessment and archival at NTNU belong to NTNU. NTNU receives a right, free of charge, to use the results of the work, including appendices to this, and can use them for teaching and research purposes with any restrictions as set out in Section 8.

8. Delayed publication (embargo)

The general rule is that student works must be available to the public.

Place an X

X	The work is to be available to the public.
---	--

In special cases, the parties may agree that all or part of the work will be subject to delayed publication for a maximum of three years. If the work is exempted from publication, it will only be available to the student, external organization and supervisor during this period. The assessment committee will have access to the work in connection with assessment. The student, supervisor and examiners have a duty of confidentiality regarding content that is exempt from publication.

The work is to be subject to delayed publication for (place an X if this applies):

Place an X

Specify date

	one year	
	two years	
	three years	

The need for delayed publication is justified on the following basis:

If, after the work is complete, the parties agree that delayed publication is not necessary, this can be changed. If so, this must be agreed in writing.

Appendices to the student work can be exempted for more than three years at the request of the external organization. NTNU (through the department) and the student must accept this if the external organization has objective grounds for requesting that one or more appendices be exempted. The external organization must send the request before the work is delivered.

The parts of the work that are not subject to delayed publication can be published in NTNU's institutional repository – see the last paragraph of Section 4. Even if the work is subject to delayed publication, the external organization must establish a basis for the student to use all or part of the work in connection with job applications as well as continuation in a master's or doctoral thesis.

9. *General provisions*

This agreement takes precedence over any other agreement(s) that have been or will be entered into by two of the parties mentioned above. If the student and the external organization are to enter into a confidentiality agreement regarding information of which the student becomes aware through the external organization, NTNU's standard template for confidentiality agreements can be used.



The external organization's own confidentiality agreement, or any confidentiality agreement that the external party has entered into in collaborative projects, can also be used provided that it does not include points in conflict with this agreement (on rights, publication, etc). However, if it emerges that

there is a conflict, NTNU's standard contract on carrying out a student work must take precedence. Any agreement on confidentiality must be attached to this agreement.

Should there be any dispute relating to this agreement, efforts must be made to resolve this by negotiations. If this does not lead to a solution, the parties agree to resolution of the dispute by arbitration in accordance with Norwegian law. Any such dispute is to be decided by the chief judge (sorenskriver) at the Sør-Trøndelag District Court or whoever he/she appoints.

This agreement is signed in four copies, where each party to this agreement is to keep one copy. The agreement comes into effect when it has been signed by NTNU, represented by the Head of Department.

Signatures:

Head of Department: Marius Pedersen  Date: 31/01/22
Supervisor at NTNU: Aditya Sole  Date: 31-01-2022
External organization (Employer at NTNU): Sule Yildirim Yayilgan



Date: 19.01.2022

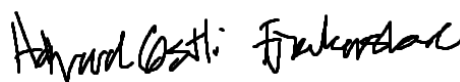
Student:



Erik Dale



Yeshi Jampel Pursley



Håvard Østli Fjørkenstad

Date: 26.01.2022

[1] If several students co-author a work, they can all be listed here. The students then have joint rights to the work. If an external organization instead wants a separate agreement to be concluded with each student, this is done.

[2] See Section 1 of the Norwegian Copyright Act of 15 June 2018 [Lov om opphavsrett til åndsverk]

Appendix B

Confidentiality Agreement



Norwegian University of Science and Technology

Approved by the Pro-Rector for Education 10 December 2020

STANDARD template between a student and an external organization for student work such as master's thesis or another student work in cooperation with an external organization, cf. Clause 9 in the standard agreement on student work carried out in cooperation with an external organization.

Student at NTNU:

Erik Dale

Yeshi Jampel Pursley

Håvard Østli Fjørkenstad

Date of birth:

07.01.2000 - Erik Dale

06.01.2000 - Yeshi Jampel Pursley

01.02.2000 - Håvard Østli Fjørkenstad

External organization:

NTNU

1. The student is to carry out work in cooperation with an external organization that is part of his/her course of study at NTNU.

2. The student undertakes to maintain secrecy regarding what he/she learns about technical equipment, procedures as well as operational and business matters that for competitive reasons have importance for the external organization. It is the responsibility of the external organization to make it absolutely clear what this information includes.

3. The student is obliged to maintain secrecy about this for 5 years from the date he/she completed work at the organization.

4. The confidentiality requirement does not apply to information that:

a) was in the public domain when it was received

b) was lawfully received from a third party without any agreement concerning secrecy

c) was developed by the student independently of information received

d) the parties are obliged to provide in accordance with law or regulations or by order of a public authority.

Signatures

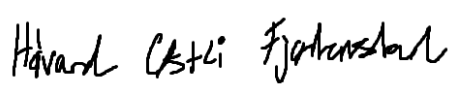
Student:



Erik Dale



Yeshe Jampel Pursley



Håvard Østli Fjørkenstad

Date:

19.01.2022

External organization: Sule Yildirim Yayilgan, NTNU



(Need signatures from employers or something here)

Date: 19.01.2022

Appendix C

Problem Statement

TITLE: image content\hand writing analysis of the Dead Sea Scrolls for provenance

Contact: sule.yildirim@ntnu.no, Tabita.tobing@ntnu.no

Currently we are working on understanding the authorship and date\period of Dead Sea Scrolls within the scope of the Lying Pen project in cooperation with University of Agder. Our particular focus on using image processing and machine learning techniques to help extract and learn the features from the Scroll images. Hence, the machine learning will be able to classify among the authorship and period. The interested student or the student group will be working on image processing techniques to identify the relevant features from the scroll text in the images. Some pre-processing of the images in order to make the content visible may be required as well. There is no need of pre knowledge on image processing techniques to do work. The student(s) will be provided the required material to learn during the project period. If the student(is) further interested in the machine learning aspect, we also have the possibility to provide the possibility to make some initial experiments with the extracted features to learn the date and authorship of the scrolls using machine learning. '

Appendix D

Project Plan



IDATG2900 - BACHELOR THESIS

Project Plan

Authors:

Erik Dale, Yeshi Jampel Pursley, Håvard Østli Fjørkenstad

January, 2022

Table of Contents

1	Goals and Frames	1
1.1	Background	1
1.2	Project Goals	1
1.2.1	Result Goals	1
1.2.2	Effect Goals	2
1.3	Frames	2
1.4	Resource Needs	2
2	Scope	2
2.1	Problem Area	2
2.2	Problem Delimitation	2
2.3	Problem Statement	3
3	Project Organizing	3
3.1	Responsibilities and Roles	3
3.2	Routines and Group Rules	4
4	Planning, Follow Up and Reporting	5
4.1	Main Division of the Project	5
4.1.1	Choice of Software Development Model/Process Framework with Argument	5
4.1.2	Description of how the group will follow the model	5
4.2	Plan for status meetings and decision moments in the period	6
4.3	Plan if some our main goals are not met	6
4.4	Plan if resource needs are not met	6
5	Organizing of Quality Assurance	6
5.1	Documentation, Standards and Configuration	6
5.1.1	Tools	7
5.2	Plan for Inspections and Testing	7
5.2.1	Testing Image Segmentation Results	7
5.2.2	Testing Machine Learning Results	7
5.2.3	Testing Image Denoising	7
5.3	Risk Analysis	8
5.3.1	Risk Assessment	8
5.3.2	Measures	8

6 Plan for implementation	9
6.1 Gantt-diagram	9
6.2 Activities, Milestones and Decision Moments	9
Bibliography	11
Appendix	12
A Standard Agreement	13
B Confidentiality Agreement	24

1 Goals and Frames

1.1 Background

The Bachelor thesis is the final evaluation we as computer engineering students at NTNU Gjøvik go through as bachelor students. We get a real issue from an employer, which is usually a local company, or in our case the university. It is a very good way for us to prepare for our working life after graduation.

The Department of Information Security and Communication Technology (IHK) at NTNU Gjøvik are working on understanding the authorship and date\period of Dead Sea Scrolls within the scope of the Lying Pen project in cooperation with University of Agder. They manually extract the different letters and words from the scrolls, something that takes a lot of time. Therefore they are interested in finding out if image processing and machine learning techniques can help extract and learn the features from the Scroll images. The machine learning will be able to classify the letters and words. Some pre-processing of the images in order to make the content visible may be required as well. In this project we will focus on using machine learning to extract the different letters and words of the scrolls, which later on can be used to determine data and authorship of them.



Figure 1: The figure shows an example of a scroll, This one is fragmented and a little bit damaged. Reference: <https://www.deadseascrolls.org.il/explore-the-archive/image/B-508181> [1]

1.2 Project Goals

1.2.1 Result Goals

- We are to create a solution for extracting and learning the features from the Scroll images, by classifying the different letters on the scroll images.
- The detection of the letters needs a high precision. It should on average have a precision of over 0.9 (90%).
- When it comes to the letter segmentation we would like to have an average "Intersection over Union" score of over 0.5 ($IoU > 0.5$). [2]

- The time it takes to scan and classify letters on each image of the scrolls should be quite low. The biggest images we have worked on are about 2000x2700 pixels and letters on images like that should not take longer than 20 seconds to classify.
- When it comes to the pre-processing part we want our denoising methods to have a PSNR score of over 30 dB.[3]
- The users should be able to extract the features on the scrolls through an easy and intuitive user interface.

1.2.2 Effect Goals

- Our solution should greatly reduce the time it takes for our employers to extract information from the Scrolls.
- Our work should help free up work capacity for our employers, by making it easier for them to extract information from the scrolls.
- Our solution should be modifiable and expendable by others outside our group. Meaning that others outside our group should be able to expand upon our solution.

1.3 Frames

The time frame we have to complete this project in is 10th of January to 20th of May 2022. Our solution and report need to be finished by then.

Technology frames:

- Our code should be able to run on Windows.
- The code we create should be easily modifiable and/or reusable by others.

1.4 Resource Needs

For our machine learning we will need at least 120 crop images of all the 22 letters in the old hebrew alphabet. Our employer will provide us with those. They may need some help from us to crop out the images as this is a long and tedious process.

2 Scope

2.1 Problem Area

The Dead Sea Scrolls are massive collections of biblical and non-biblical texts and manuscripts. They were discovered in the Judean desert in 1947. The Scrolls consist of nearly intact scrolls and thousands of fragments. The scrolls we will be working on are written in old Hebrew.[4] The Department of Information Security and Communication Technology at NTNU Gjøvik is working on improving the way computer vision is used in the analyzing of the Dead Sea Scrolls, especially when it comes to the many Scroll fragments where it can be hard to extract letters and words. Up until this point they have manually extracted the letters from the Scrolls.

2.2 Problem Delimitation

We as developers are just responsible for working on the extraction of information from the Scrolls, not responsible for actually understanding what is written there. The solution we provide is only

going to be used by the Department of Information Security and Communication Technology at NTNU Gjøvik and/or other partners involved in the Lying Pen project.

2.3 Problem Statement

Our task will be to build a system of handwritten text recognition using machine learning on intact scrolls then apply the algorithm to other fragments. Our task will be to first do some image pre-processing (if that is needed), then segment the images and finally extracting the words and letters using machine learning.[4].

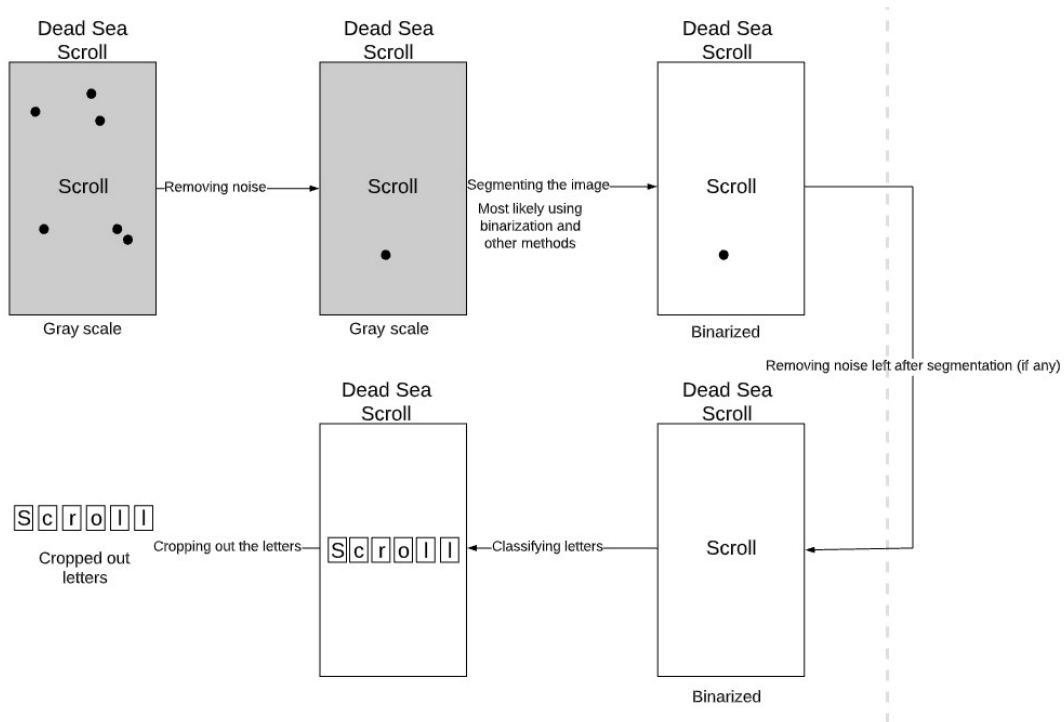


Figure 2: The figure shows the process a Dead Sea Scrolls image has to go through in order for us to classify the words and letters on it.

3 Project Organizing

3.1 Responsibilities and Roles

Roles:

A. Leadership – Project leader (Erik)

Responsibilities: If we disagree, the leader has the final word and during a voting-process he will have two votes. Make sure everyone follows the rules and that everyone has something to do. The leader is the chairman in meetings.

B. Communication responsible (Yeshi)

Responsibilities: Responsible for communication in and out of the group. Meaning things like scheduling meetings with our supervisors and clients.

C. Archive/ Document responsible (Håvard)

Responsibilities: Make sure that the meetings are documented. Has a general overview of all of

our documentation.

Main Work Responsibilities:

A. Image Pre-Processing (Erik)

Responsible for coding the necessary image enhancement techniques to ensure an easier segmentation.

B. Image Segmentation (Yeshe)

Responsible for the code and techniques for segmenting the larger images into lines, words and characters for use in the neural network.

C. Machine Learning (Håvard)

Responsible for the code behind the neural network model and training, which are to be used for classifying the input data.

3.2 Routines and Group Rules

Routines:

With the exception of weekly meetings within the group, with project advisors and with the client, the group does not have any specified routines. Work and other intermittent meetings are done as the group sees necessary.

Group Rules and Procedures:

A. Meetings

Yeshe sends out summons. Two-three times a week, from ca. 10am to 6pm, and of course not when we have other lectures. Yeshe notifies us on Discord.

B. Notification in case of absence or other incidents

You need a reason for absence. You need to notify the group as early as possible if you cannot attend a meeting.

C. Documents

Discord/Github/Google Docs/Google Sheet will be used to share documents, Github and git will be used as versioning.

D. Distribution of tasks

We distribute tasks evenly amongst ourselves so that the amount of work is as equal as possible between us. We also give everyone the opportunity to contribute to what task they want.

E. Policy for monitoring tasks

Kanban-board, this is mostly the leader's task.

F. Submission of team work

We uphold deadlines within the group.

G. Timesheet

After all work we do we note down when we worked and what work we did in a timesheet table in excel.

H. Attendance and preparation

Accepted meeting time: 10am - 6pm, and not during other lectures. Every group-member should

come prepared to each meeting.

I. Presence and commitment

We will try to avoid distractions as much as possible, but if too many occur, the leader will intervene. We will put in scheduled breaks in our schedule to avoid them as well.

J. How to support each other

Be honest to each other. Give constructive criticism not destructive.

K. Disagreement, breach of contract

If breach of contract occurs:

1. A discussion in the group will be held to try and fix the problem.
2. Written warning if the work contract has been breached several times if the majority of the group agrees.
3. Conversation with supervisor - after two written warnings.
4. Exclusion from group after majority vote by majority group members and supervisor, after several conversations with supervisor.

L. Failure to complete a task

If the person is not able to complete a task, the person must explain why he was not able to complete the task by describing what he has tried and what went wrong. If the reason is inadequate we move into point D.

4 Planning, Follow Up and Reporting

4.1 Main Division of the Project

4.1.1 Choice of Software Development Model/Process Framework with Argument

For the software development model we've chosen Agile, mainly because of the "agile" nature of the framework. Being able to adapt to change will be very important if the project scope grows, in case we accomplish the initial tasks. This is why we favor agile over something like Waterfall, so that we don't have to design everything from the start and to keep ourselves open for deviation. Agile will also the group deliver faster, keep the clients in the loop and our project on course with their feedback. It will also help eliminate the risk of wasting time due to delays or setbacks.[5]

Within the agile framework, we've chosen to use Kanban to help our work visualized and to keep everyone on the same page, by easily keeping track of what has been done and what needs to be done. Kanban has also been favored over something like Scrum, to keep our work routines a little more flexible, by not thinking about sprints or strict deadlines. Most importantly, Kanban is a fairly easy framework which will keep us focused on the actual work, instead of getting slowed down by investing time into learning and using a framework correctly. [6]

4.1.2 Description of how the group will follow the model

Group will follow the model by creating a Kanban board on Github to keep track of our progress and work backlog. The board will be divided up in a To do, To do - Priority, In progress, Testing and Done.

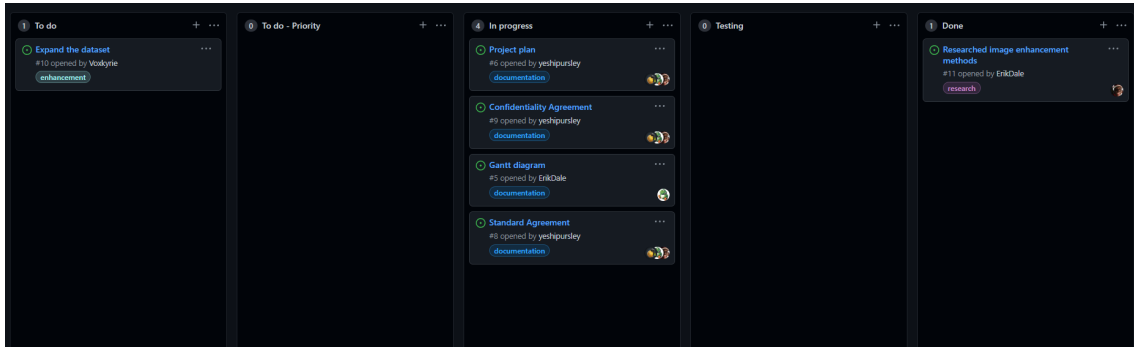


Figure 3: Shows our kanban board on Github.

4.2 Plan for status meetings and decision moments in the period

Meetings: Group meetings are held every Monday at 12:00, where we discuss our work from the previous week and keep each other informed on the progress. We also discuss potential problems and questions that we would like to take up in the following meetings that week. The meetings do not have a set time limit, but are usually between 1-2 hours. Group meetings are also held after every other type of meeting. Meetings with the client is held every Thursday between 12:00 and 13:00, where we discuss our current project and ask questions.

Meetings with the advisor is held every Thursday/Friday somewhere between 10:00 and 15:00.

4.3 Plan if some our main goals are not met

- If our pre-processing goals are not met for the damaged scrolls we will have to work with well conditioned scrolls only. Damaged and fragmented scrolls may be harder to image pre-process, so denoising and enhancing them might be hard.
- If we are not able to segment the letters/words automatically, we will then focus on manually segmenting the letters. We can either work on manually fixing the letters/words that were not automatically segmented correctly or simply segment all the letters/words manually. We will then focus more on the other parts of the project like Image Enhancement and letter classification.
- If we are not able to classify the letters with high enough precision, we will focus more on Image Enhancement, Image Segmentation, and the literary research part of the project.

4.4 Plan if resource needs are not met

If we are not able to get at least 120 crop images of all the 22 letters we might have to consider data augmentation.[7]

5 Organizing of Quality Assurance

5.1 Documentation, Standards and Configuration

Håvard is responsible for keeping an overview of the documentation. We make sure that it is easy for the group to find out what we are working on, who is responsible for what, and that we write good documentation in our code.

5.1.1 Tools

Name	Type	Area of use
Discord	Communication Platform	Communication and meetings for the group.
Microsoft Teams	Communication Platform	Meetings with advisors and client.
Google Docs	Online word processor	Collaboration agreement, standard agreement and meeting log.
Google Sheet	Spreadsheet program	Used to log time and create Gantt-diagram.
Overleaf	LaTeX editor	Writing the Project Plan and report.
Google Slides	Presentation program	For making presentations.
PyCharm	Python IDE	Writing and running python code.
Visual Studio Code	IDE	Writing and running python code.
Jupyter Notebook	Python notebook	Writing and running python code
Microsoft PowerToys	Image editing tool	Mostly for resizing images
Lucidchart	Online chart drawer	Creating diagrams and figures

5.2 Plan for Inspections and Testing

5.2.1 Testing Image Segmentation Results

For testing the Image Segmentation we draw rectangles over each of the letters/words on the image of the scroll provided by the code. After filtering the image segmentation results, we draw red rectangles over results that the filter filtered out and blue ones over the results we want. This allows us to get an oversight of how well the images segmentation worked. We can zoom in on the document, analyze and compare with different results.

To test the individual letters we can create a ground truth of a the shape of a letter and then compare it with the prediction using the "Intersection over Union" metric.[2]. We can also use this metric to calculate the precision of the amount of letters/words it segmented in the document.

5.2.2 Testing Machine Learning Results

While training the model, we will use 20% of the dataset to test the accuracy of the model, as well as the average loss.

The finished model will also be used to predict the labels of a known dataset, which allows us to easily see and visualize how the model performs on both standard and edge cases.

5.2.3 Testing Image Denoising

Testing and inspection of image denoising methods are in general pretty easy to evaluate just by looking at the differences in the original and the denoised image. There exists however mathematical formulas to test the different kinds of denoising methods. PSNR is one that can be used to evaluate performance metrics of such methods.[8] PSNR is used to estimate the efficiency of compressors, filters, etc. A larger PSNR value means a more efficient compression or filter method.[3] This is what we will use to evaluate the performance of the different methods we have used to try reducing noise in our scroll-images.

5.3 Risk Analysis

5.3.1 Risk Assessment

Description	Likelihood	Consequence	Risk	Measures
Conflict within the group.	Unlikely	Problematic	Less Severe	Yes
Somebody is sick for a significant duration.	Unlikely	Severe	Considerable	Yes
Project is not finish in time.	Unlikely	Problematic	Less Severe	Yes
Somebody leaves the group.	Unlikely	Severe	Considerable	Yes
Loss of documents and/or source code.	Unlikely	Severe	Considerable	Yes
Unable to implement specific functionality.	Likely	Problematic	Considerable	Yes
Client wants to change the requirements specification.	Very Likely	Problematic	Serious	Yes

5.3.2 Measures

Number	Measure
1	If there is a conflict within the group we will have to resolve this problem quickly through meetings with the group and with supervisors if necessary. It is important to solve it quickly to minimize the consequences.
2 and 4	To reduce the consequences of somebody being sick for a long duration, we make sure that everybody in the group understands what each of us are working on in the project. Good documentation is also important. If somebody is not able to work or they have to leave the group, we would be able to continue working on what they were working on.
3	We will have a meeting with the client if we are not able to finish the project in time. We will have to discuss with them about reducing the scope of the project. We make sure that we are aware of the progress of our work so that if we notice that something is behind schedule we address it early on.
5	All of our documents are stored on Google docs/Sheet and overleaf. We upload our source code to our GitHub repository and locally on our computers.
6	If we are unable to implement certain functionalities to the project we will have to have a meeting with our client and explain why we were unable to do that. We will then discuss alternative solutions to the problem.
7	If the client wishes to change the requirements specification we will have to evaluate if we have enough time and knowledge of the thing they wish us to change and/or implement. We have a meeting with them at least once a week on Thursday which makes it easier for us to evaluate the change and discuss with the client about it.

6 Plan for implementation

6.1 Gantt-diagram

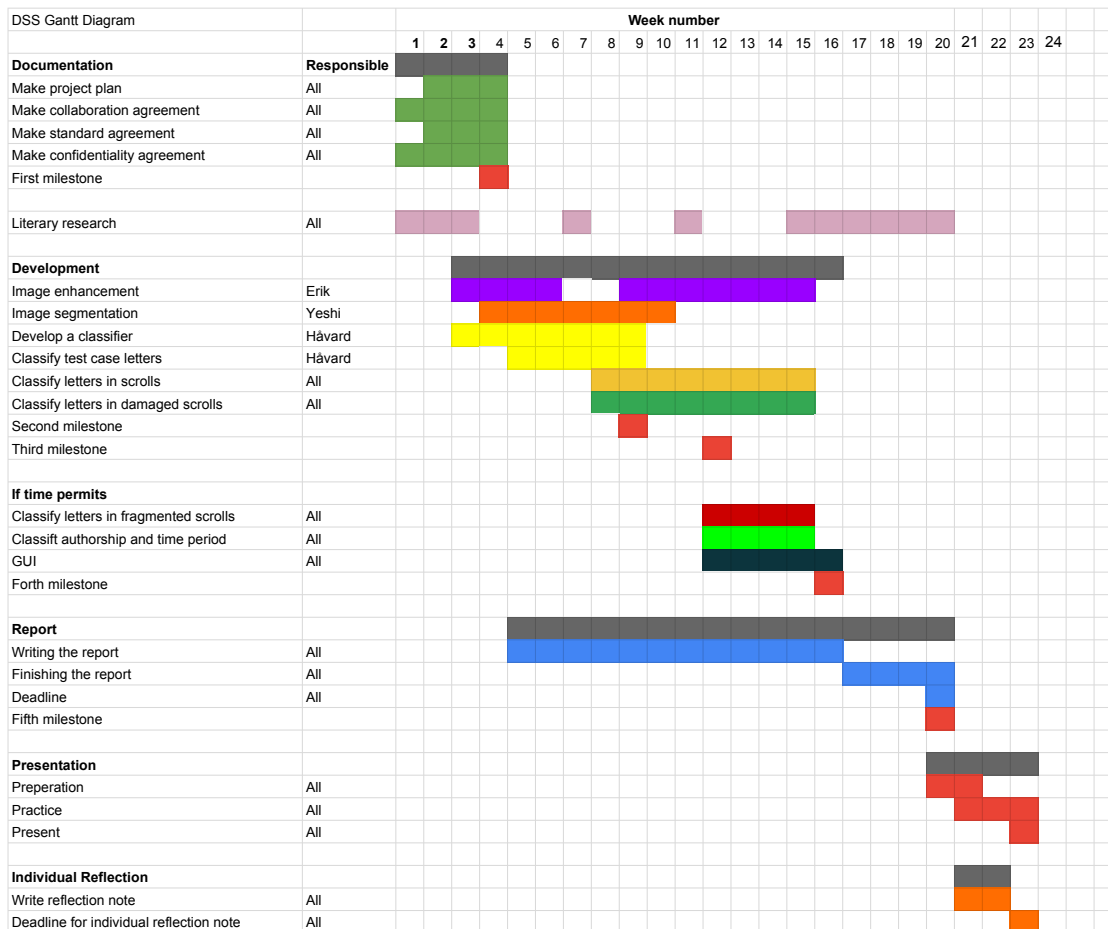


Figure 4: Our Gantt diagram. [Link to it.](#)

6.2 Activities, Milestones and Decision Moments

First milestone:

- Finish the project plan.
- Finish the collaboration agreement.
- Finish the standard agreement.

Second milestone:

- Finish code for Image Enhancement.
- Finish code for Image segmentation.
- Develop a classifier.
- Finish code that is able to classify test case letters.

- These tasks will provide the core functionality that we will need to work on the scrolls. We will work more on Image Enhancement and Image Segmentation later to improve results but when we finish this milestone those functionalities will provide good enough results for the next milestone.

Third milestone:

- Create code that can classify letters in the clean scrolls provided by Tabita.
- Create code that works for scrolls that need image enhancement.
- In this milestone we expand on our work from the previous milestone.

Forth milestone:

- Create code that works for scrolls that are fragmented.
- Create code for recognizing handwriting.
- Create a GUI.
- This milestone contains optional tasks that we can work on if we need more work.

Fifth milestone:

- Finish the report. We will be working on the report during the entire project.

Bibliography

1. Halevi S. The Leon Levy Dead Sea Scrolls Digital Library. Available from: <https://www.deadseascrolls.org.il/explore-the-archive/image/B-508181> [Accessed on: 2022 Jan 28]
2. Jordan J. Evaluating image segmentation models. Available from: <https://www.jeremyjordan.me/evaluating-image-segmentation-models/> [Accessed on: 2022 Jan 23]
3. Saha A. Python — Peak Signal-to-Noise Ratio (PSNR). Available from: <https://www.geeksforgeeks.org/python-peak-signal-to-noise-ratio-psnr/> [Accessed on: 2022 Jan 21]
4. Tobing T. Handwritten Text Recognition on the Dead Sea Scrolls. 2022
5. Radigan D. Agile vs. Waterfall project management. Available from: <https://www.atlassian.com/agile/project-management/project-management-intro> [Accessed on: 2022 Jan 23]
6. Rehkopf M. What is a kanban board? Available from: <https://www.atlassian.com/agile/kanban/boards> [Accessed on: 2022 Jan 23]
7. Wikipedia. Data augmentation. Available from: https://en.wikipedia.org/wiki/Data_augmentation [Accessed on: 2022 Jan 29]
8. Fan L, Zhang F, Fan H and Zhang C. Brief review of image denoising techniques. Available from: <https://vciba.springeropen.com/articles/10.1186/s42492-019-0016-7> [Accessed on: 2022 Jan 21]

Appendix

A Standard Agreement



Norwegian University of Science and Technology

STANDARD AGREEMENT

on student works carried out in cooperation with an external organization

The agreement is mandatory for student works such as master's thesis, bachelor's thesis or project assignment (hereinafter referred to as works) at NTNU that are carried out in cooperation with an external organization.

Explanation of terms

Copyright

Is the right of the creator of a literary, scientific or artistic work to produce copies of the work and make it available to the public. A student thesis or paper is such a work.

Ownership of results

Means that whoever owns the results decides on these. The basic principle is that the student owns the results from their own student work. Students can also transfer their ownership to the external organization.

Right to use results

The owner of the results can give others a right to use the results – for example, the student gives NTNU and the external organization the right to use the results from the student work in their activities.

Project background

What the parties to the agreement bring with them into the project, that is what each party already owns or has rights to and which is used in the further development of the student's work. This may also be material to which third parties (who are not parties to the agreement) have rights.

Delayed publication (embargo)

Means that a work will not be available to the public until a certain period has passed; for example, publication will be delayed for three years. In this case, only the supervisor at NTNU, the examiners and the external organization will have access to the student work for the first three years after the student work has been submitted.

1. Contracting parties

The Norwegian University of Science and Technology (NTNU) Department: Institutt for datateknologi og informatikk
Supervisor at NTNU: Aditya Suneel Sole Marius Pedersen email / telephone:

<p>aditya.sole@ntnu.no / 94165542</p> <p>marius.pedersen@ntnu.no / 93634385</p>
<p>External organization:</p> <p>Contact person, email address and telephone number of the external organization:</p> <p>Sule Yildirim Yayilgan, sule.yildirim@ntnu.no, 46623172</p> <p>Tabita Anggraini Meilita, tabita.tobing@ntnu.no, +62 852 176 976 52</p>
<p>Students:</p> <p>Erik Dale</p> <p>Yeshi Jampel Pursley</p> <p>Håvard Østli Fjørkenstad</p> <p>Date of birth:</p> <p>06.01.2000 (Yeshi)</p> <p>07.01.2000 (Erik)</p> <p>01.02.2000 (Håvard)</p>

The parties are responsible for clearing any intellectual property rights that the student, NTNU, the external organization or third party (which is not a party to the agreement) has to project background before use in connection with completion of the work. Ownership of project background must be set out in a separate annex to the agreement where this may be significant for the completion of the student work.

2. Execution of the work

The student is to complete: (Place an X)

A master's thesis	
A bachelor's thesis	X
A project assignment	
Another student work	

Start date: 01.01.2022
Completion date: 20.05.2022

The working title of the work is:

Image Content\Hand Writing Analysis of the Dead Sea Scrolls for Provenance

The responsible supervisor at NTNU has the overarching academic responsibility for the design and approval of the project description and the student's learning.

3. Duties of the external organization

The external organization must provide a contact person who has the necessary expertise to provide the student with adequate guidance in collaboration with the supervisor at NTNU. The external contact person is specified in Section 1.

The purpose of the work is to carry out a student assignment. The work is performed as part of the programme of study. The student must not receive a salary or similar remuneration from the external organization for the student work. Expenses related to carrying out the work must be covered by the external organization. Examples of relevant expenses include travel, materials for building prototypes, purchasing of samples, tests in a laboratory, chemicals. The student must obtain clearance for coverage of expenses with the external organization in advance.

The external organization must cover the following expenses for carrying out the work:

No

Coverage of expenses for purposes other than those listed here is to be decided by the external organization during the work process.

4. The student's rights

Students hold the copyright to their works [2]. All results of the work, created by the student alone through their own efforts, is owned by the student with the limitations that follow from sections 5, 6 and 7 below. The right of ownership to the results is to be transferred to the external organization if Section 5 b is checked or in cases as specified in Section 6 (transfer in connection with patentable inventions).

In accordance with the Copyright Act, students always retain the moral rights to their own literary, scientific or artistic work, that is, the right to claim authorship (the right of attribution) and the right to object to any distortion or modification of a work (the right of integrity).

A student has the right to enter into a separate agreement with NTNU on publication of their work in NTNU's institutional repository on the Internet (NTNU Open). The student also has the right to publish the work or parts of it in other connections if no restrictions on the right to publish have been agreed on in this agreement; see Section 8.

5. Rights of the external organization

Where the work is based on or further develops materials and/or methods (project background) owned by the external organization, the project background is still owned by the external organization. If the student is to use results that include the external organization's project background, a prerequisite for this is that a separate agreement on this has been entered into between the student and the external organization.

Alternative a) (Place an X) General rule

	The external organization is to have the right to use the results of the work
--	---

This means that the external organization must have the right to use the results of the work in its own activities. The right is non-exclusive.

Alternative B) (Place an X) Exception

X	The external organization is to have the right of ownership to the results of the task and the student's contribution to the external organization's project
---	--

Justification of the external organization's need to have ownership of the results transferred to it:

Because this work is carried out within the scope of the lying pen of scribes project founded by the research council of Norway.

6. Remuneration for patentable inventions

If the student, in connection with carrying out the work, has achieved a patentable invention, either alone or together with others, the external organization can claim transfer of the right to the invention to itself. A prerequisite for this is that exploitation of the invention falls within the external organization's sphere of activity. If so, the student is entitled to reasonable remuneration. The remuneration is to be determined in accordance with Section 7 of the Employees' Inventions Act. The provisions on deadlines in Section 7 apply correspondingly.

7. NTNU's rights

The submitted files of the work, together with appendices, which are necessary for assessment and archival at NTNU belong to NTNU. NTNU receives a right, free of charge, to use the results of the work, including appendices to this, and can use them for teaching and research purposes with any restrictions as set out in Section 8.

8. Delayed publication (embargo)

The general rule is that student works must be available to the public.

Place an X

X	The work is to be available to the public.
---	--

In special cases, the parties may agree that all or part of the work will be subject to delayed publication for a maximum of three years. If the work is exempted from publication, it will only be available to the student, external organization and supervisor during this period. The assessment committee will have access to the work in connection with assessment. The student, supervisor and examiners have a duty of confidentiality regarding content that is exempt from publication.

The work is to be subject to delayed publication for (place an X if this applies):

Place an X

Specify date

	one year	
	two years	
	three years	

The need for delayed publication is justified on the following basis:

If, after the work is complete, the parties agree that delayed publication is not necessary, this can be changed. If so, this must be agreed in writing.

Appendices to the student work can be exempted for more than three years at the request of the external organization. NTNU (through the department) and the student must accept this if the external organization has objective grounds for requesting that one or more appendices be exempted. The external organization must send the request before the work is delivered.

The parts of the work that are not subject to delayed publication can be published in NTNU's institutional repository – see the last paragraph of Section 4. Even if the work is subject to delayed publication, the external organization must establish a basis for the student to use all or part of the work in connection with job applications as well as continuation in a master's or doctoral thesis.

9. General provisions

This agreement takes precedence over any other agreement(s) that have been or will be entered into by two of the parties mentioned above. If the student and the external organization are to enter into a confidentiality agreement regarding information of which the student becomes aware through the external organization, NTNU's standard template for confidentiality agreements can be used.



The external organization's own confidentiality agreement, or any confidentiality agreement that the external party has entered into in collaborative projects, can also be used provided that it does not include points in conflict with this agreement (on rights, publication, etc). However, if it emerges that




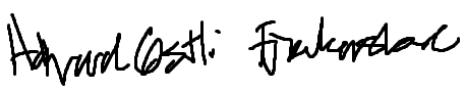
there is a conflict, NTNU's standard contract on carrying out a student work must take precedence. Any agreement on confidentiality must be attached to this agreement.

Should there be any dispute relating to this agreement, efforts must be made to resolve this by negotiations. If this does not lead to a solution, the parties agree to resolution of the dispute by arbitration in accordance with Norwegian law. Any such dispute is to be decided by the chief judge (sorenskriver) at the Sør-Trøndelag District Court or whoever he/she appoints.

This agreement is signed in four copies, where each party to this agreement is to keep one copy. The agreement comes into effect when it has been signed by NTNU, represented by the Head of Department.

Signatures:

Head of Department: Marius Pedersen  Date: 31/01/22
Supervisor at NTNU: Aditya Sole  Date: 31-01-2022
External organization (Employer at NTNU): Sule Yildirim Yayilgan

 Date: 19.01.2022		
Student:   		
<hr/> Erik Dale	<hr/> Yeshi Jampel Pursley	<hr/> Håvard Østli Fjørkenstad
Date: 26.01.2022		

[1] If several students co-author a work, they can all be listed here. The students then have joint rights to the work. If an external organization instead wants a separate agreement to be concluded with each student, this is done.

[2] See Section 1 of the Norwegian Copyright Act of 15 June 2018 [Lov om opphavsrett til åndsverk]

B Confidentiality Agreement



Norwegian University of Science and Technology

Approved by the Pro-Rector for Education 10 December 2020

STANDARD template between a student and an external organization for student work such as master's thesis or another student work in cooperation with an external organization, cf. Clause 9 in the standard agreement on student work carried out in cooperation with an external organization.

Student at NTNU:

Erik Dale

Yeshe Jampel Pursley

Håvard Østli Fjørkenstad

Date of birth:

07.01.2000 - Erik Dale

06.01.2000 - Yeshe Jampel Pursley

01.02.2000 - Håvard Østli Fjørkenstad

External organization:

NTNU

1. The student is to carry out work in cooperation with an external organization that is part of his/her course of study at NTNU.

2. The student undertakes to maintain secrecy regarding what he/she learns about technical equipment, procedures as well as operational and business matters that for competitive reasons have importance for the external organization. It is the responsibility of the external organization to make it absolutely clear what this information includes.

3. The student is obliged to maintain secrecy about this for 5 years from the date he/she completed work at the organization.

4. The confidentiality requirement does not apply to information that:

a) was in the public domain when it was received

b) was lawfully received from a third party without any agreement concerning secrecy

c) was developed by the student independently of information received

d) the parties are obliged to provide in accordance with law or regulations or by order of a public authority.

Signatures

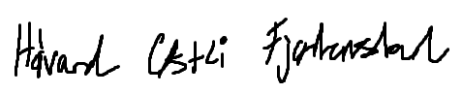
Student:



Erik Dale



Yeshi Jampel Pursley



Håvard Østli Fjørkenstad

Date:

19.01.2022

External organization: Sule Yildirim Yayilgan, NTNU



Date: 19.01.2022

Appendix E

How We Created a Custom TRAINEDDATA File

We followed all of the steps on the website [53] to create our custom TRAINED-DATA file. We used the article [54] to get a better understanding of the steps in [53].

We used Tesseract version 4.0.0. Other versions might not work.

The filename to the images we wish to train with Tesseract needs to have the form: [language name].[font name].exp[number].[file extension]. The steps we are going to list, which are the steps we followed to create our custom TRAINED-DATA file, include the steps from the website [53], how we used QT Box editor, and examples of each step.

1. Open CMD.
2. Create a box file.
 - Syntax: `tesseract [langname].[fontname].[expN].[file-extension] [langname].[fontname].[expN] batch.nochop makebox`
 - Example: `tesseract heb.Dssfont-Regular.exp0.png heb.Dssfont-Regular.exp0 batch.nochop makebox`
3. Use QT Box Editor to fix boxes that are incorrect. More information about QT Box Editor can be read in Section 3.3.5. Select the image in the application and not the box file.
4. Create a .tr file.
 - Syntax: `tesseract [langname].[fontname].[expN].[file-extension] [langname].[fontname].[expN] box.train`
 - Example: `tesseract heb.Dssfont-Regular.exp0.png heb.Dssfont-Regular.exp0 box.train`
5. Extract the charset from the box file.
 - Syntax: `unicharset_extractor [langname].[fontname].[expN].box`
 - Example: `unicharset_extractor heb.Dssfont-Regular.exp0.box`

6. Create font properties file

- Syntax: `echo "[fontname] [italic (0 or 1)] [bold (0 or 1)] [monospace (0 or 1)] [serif (0 or 1)] [fraktur (0 or 1)]" > font_properties`
- Example: `echo "Dssfont-Regular 0 0 0 0" > font_properties`

7. Train the data

- Syntax: `mfttraining -F font_properties -U unicharset -O [langname].unicharset [langname].[fontname].[expN].tr`
- Example: `mfttraining -F font_properties -U unicharset -O heb.unicharset heb.Dssfont-Regular.exp0.tr`
- Syntax: `cntraining [langname].[fontname].[expN].tr`
- Example: `cntraining heb.Dssfont-Regular.exp0.tr`

8. Rename the files that were created after training the data to include the language.

- Syntax: `rename first_filename second_filename`
- Example: `rename shapetable heb.shapetable`

9. Combine the files we renamed into a TRAINEDDATA file

- Syntax: `combine_tessdata [langname].`
- Example: `combine_tessdata heb.`

Appendix F

User Interface Repository Link and README

Following is a link to the user interface GitHub repository and its README.

F.1 Link

https://github.com/ErikDale/DSS_userinterface

F.2 README



DSS_userinterface

This application is made mainly for Windows and the following steps are for Windows OS primarily.

Run User Interface

To run this program you can either use our .exe file or clone this repo and run our python file.

Use Our Executable File

Here: https://www.dropbox.com/s/pp1g0mtchky6gz/dss_userinterface.zip?dl=0 you can find a zip file containing our executable file. Simply download that, unzip it and run the dss_userinterface.exe file that is inside it. You unzip a zip-file by right clicking it and selecting "extract all":



If you move the .exe file from the folder it will not work, so keep it in the folder.

Use Repository To Run User Interface

Install Python and Anaconda

To run our user interface using our repository you need to have python: <https://www.python.org/downloads/> installed on your Windows computer. I would also recommend you download Anaconda: <https://www.anaconda.com/products/individual> to make it easier to run the application.

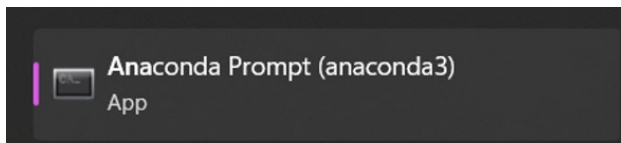
Clone Repository

Clone repository in a wanted location on your computer using:

```
git clone https://github.com/ErikDale/DSS_userinterface.git
```

Open Anaconda Prompt

Once you have installed Anaconda you should be able to press the Windows button and search for Anaconda Prompt:



Once you have opened that you should navigate to the repo. You can do this by using:

```
cd <full_path to the cloned repo>
```

Use Pip Install

When you have navigated to the cloned repo you should use:

```
pip install -r requirements.txt
```

to install all the dependencies needed to run the application.

🔗 Run User Interface

When that is done you should be able to run the user interface using:

```
python ./dss_userinterface.py
```

🔗 Test image

To test the user interface we have added a test image called *test.jpg* in the repo. The image is a paragraph from The Great Isaiah Scroll column 35, gotten from: <https://archive.org/details/qumran>



Appendix G

Repository Link and README

Following is a link to the main GitHub repository and its README.

G.1 Link

<https://github.com/yeshipursley/image-analysis-DSS>

G.2 README



Image Analysis DSS

The files in this project is made mainly for Windows and the following steps are for Windows OS primarily. The files can however be run on other operative systems as well.

Run Files

Install Python and Anaconda

To run our files using our repository you need to have python: <https://www.python.org/downloads/> installed on your Windows computer. I would also recommend you download Anaconda: <https://www.anaconda.com/products/individual> to make it easier to run the files.

Clone Repository

Clone repository in a wanted location on your computer using:

```
git clone https://github.com/yeshipursley/image-analysis-DSS.git
```

Open Anaconda Prompt

Once you have installed Anaconda you should be able to press the Windows button and search for Anaconda Prompt:

Once you have opened that you should navigate to the repo. You can do this by using:

```
cd <full_path to the cloned repo>
```

Use Pip Install

Before installing the pip packages, create a virtual enviroment to make managing the pacakges easier with:

```
python3 -m venv dss
```

Then, to activate the virtual enviroment do:

```
dss\Scripts\activate
```

Terminal should now show (dss) before the prompt.

When that is done you should use:

```
pip install -r requirements.txt
```

to install all the dependencies needed to run the files.

Run Image Enhancement Files

The image enhancement part of the project has been divided into three folders: Histogram, MorphologicalTransformations and NoiseReduction. The way you run the python files in these three folders is the same for every folder. You should run them in the Anaconda Prompt after you have installed the necessary dependencies using the earlier step. This is how you run them:

```
cd <folder_name>
```

Use cd to move to wanted folder.

```
python .\filename.py -input .\inputImage.jpg -output .\outputImage.jpg
```

The `.filename.py` is the name of the python file you want to run. The `.inputImage.jpg` is the path of the image to want to do image enhancement on. The `.outputImage.jpg` is the path where you want to store the now newly created image that has been enhanced. Remember to include the filename in this path. The paths can be absolute paths or relative paths to the cloned repo. Use backslashes ("`\`") in the paths and if you use relative paths use a dot before the path (`.\path\imageName.jpg`).

IMPORTANT: For the `adaptive_histogram.py` file the image should be rgb or gray-scale. For all the other files the image should be binarized.

🔗 Image Segmentation Files Structure

Our image segmentation work has been divided into 5 folders:

- `binarization_comparison`: this file was used to test different binarization methods.
- `custom_traineddata_file`: contains our custom traineddata file
- `pytesseract_image_to_boxes_comparison`: contains a file that was used to test how different binarization methods affect pytesseract's segmentation.
- `segmentation_to_classifier`: contains files needed to run our image segmentation.
- `yolo_to_img`: a file we created to manually convert letters extracted from Labellmg in the YOLO format to images.

🔗 Run Image Segmentation Files

Steps to test our image segmentation:

- Copy the `segmentation_to_classifier` folder into another folder.
- Download tesseract - <https://github.com/UB-Mannheim/tesseract/wiki>
- Copy the `heb.traineddata` file from the "custom_traineddata_file" folder in this GitHub repository and put it into your Tesseract-OCR/tessdata folder.
- Copy the `default.model` from "image-analysis-DSS/machine_learning/neural_network/models/default/" folder in this GitHub repository and paste it in your `segmentation_to_classifier` folder.
- In `segmentation_to_classifier.py` make sure the path, located at line 322, goes to your `tesseract.exe` file.
- In `segmentation_to_classifier.py` add these lines at the bottom of the code:

```
img = cv2.imread('path to your image')  
  
Segmentor().segmentClearBackground(img)  
# or  
Segmentor().segmentVariedBackground(img)
```

- Run the `segmentation_to_classifier.py` file.

🔗 Run Machine Learning Files

🔗 Training

```
python MachineLearning\NeuralNetwork\train.py
```

Add `--gpu` to run the training on an available GPU

Add `-e :number:` to run the training for a set number of epochs, where `:number:` is your desired epoch (default is 20)

Add `-d :name:` or `--dataset :name:` to run a dataset other than the default "default", datasets must be located in the `MachineLearning/NeuralNetwork/datasets` folder

Add `-m :name:` or `--model :name:` to give the trained model a name other than "default", trained models will be saved in the `MachineLearning/NeuralNetwork/models` folder

Add `--earlystop :loss:` to use the callback function and stop the training at a certain loss value, or until all epochs are completed

🔗 Extraction

```
python MachineLearning\extract.py
```

Add `-d :path:` to specify what `:path:` folder the dataset should be extracted from, folder needs to have subfolders with all the letters with their respective name

Add `-n :name:` or `--name :name:` to specify what `:name:` the dataset should be called, datasets are saved in the `MachineLearning/NeuralNetwork/datasets` folder

🔗 Predict

```
python MachineLearning\NeuralNetwork\predict.py
```

You will need to change the `model_name` variable at line 114 to the desired model located in the `MachineLearning/NeuralNetwork/models` folder

🔗 Test image

To test our code we have added a test image called `test.png` in the repo. The image is a paragraph from The Great Isaiah Scroll column 35, gotten from: <https://archive.org/details/qumran>



Appendix H

YOLO to Images Code

```
1 # Taken from https://stackoverflow.com/questions/64096953/how-to-convert-yolo-
  # format-bounding-box-coordinates-into-opencv-format
2 # Date: 25.02.2022
3 # Have done a few changes
4 import cv2
5 import matplotlib.pyplot as plt
6
7 # For changing the directory so that we can save images in a different folder
8 import os
9
10 # Reading the DSS-image
11 img = cv2.imread('<DSS-image>')
12
13 # Converting it to gray-scale
14 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
15
16 # Getting the shape of the gray-scale image
17 dh, dw = gray.shape
18
19 # Opening and reading the lines in the yolo-txt file and storing them in data
20 fl = open('<yolo-txt-file>', 'r')
21 data = fl.readlines()
22 fl.close()
23
24 # Changing the folder to the folder you want to store the crops in. May need full
  # folder-path
25 os.chdir('<Folder you want to store the crops in>')
26
27 # Counter that is to be in the file name of the crops
28 counter = 1
29
30 # Going through all the crops and saving them as individual images.
31 # It also creates an image with rectangles over the crops that will be done.
32 for dt in data:
33
34     # Split string to float
35     _, x, y, w, h = map(float, dt.split(' '))
36
```

```
37 # Taken from https://github.com/pjreddie/darknet/blob/810
    d7f797bdb2f021dbe65d2524c2ff6b8ab5c8b/src/image.c#L283-L291
38 # via https://stackoverflow.com/questions/44544471/how-to-get-the-coordinates-
    of-the-bounding-box-in-yolo-object-detection#comment102178409_44592380
39 # Date: 25.02.2022
40 l = int((x - w / 2) * dw)
41 r = int((x + w / 2) * dw)
42 t = int((y - h / 2) * dh)
43 b = int((y + h / 2) * dh)
44
45
46 # Cropping every letter I have marked in labelImg
47 crop = gray[int(t):int(b), int(l):int(r)]
48
49 # The letter to use in the file-name
50 letter = '<letter-name>'
51
52 # the number of the column if any
53 column = '<column-number>'
54
55 # What the crops will be saved as in the folder you chose.
56 # Should name the crops the letter that it is a crop of and some kind of
    counter.
57 cv2.imwrite(letter + '0' + column + str(counter) + '.png', crop)
58
59 # Increment the counter
60 counter += 1
```

Code listing H.1: How we convert the labels and coordinates from the YOLO format to images.

Appendix I

Time Tracking

Here is a link to our time tracking log: <https://docs.google.com/spreadsheets/d/1yBRsi8tCnhGBsxudDbr1g2ywAfwAbc1PULBqGnk-si0/edit?usp=sharing>

Time Tracking Table				
Month	Erik	Yeshi	Håvard	Total
Jan	66:15	71:38	47:45	185:38
Feb	67:47	41:37	48:14	157:38
Mar	91:23	59:21	33:04	183:48
Apr	39:53	54:19	15:58	110:10
May	55:11	84:08	40:13	179:32
Total	320:29	311:03	185:14	816:46

Figure I.1: Time tracking table. These numbers represent how many hours we have worked.

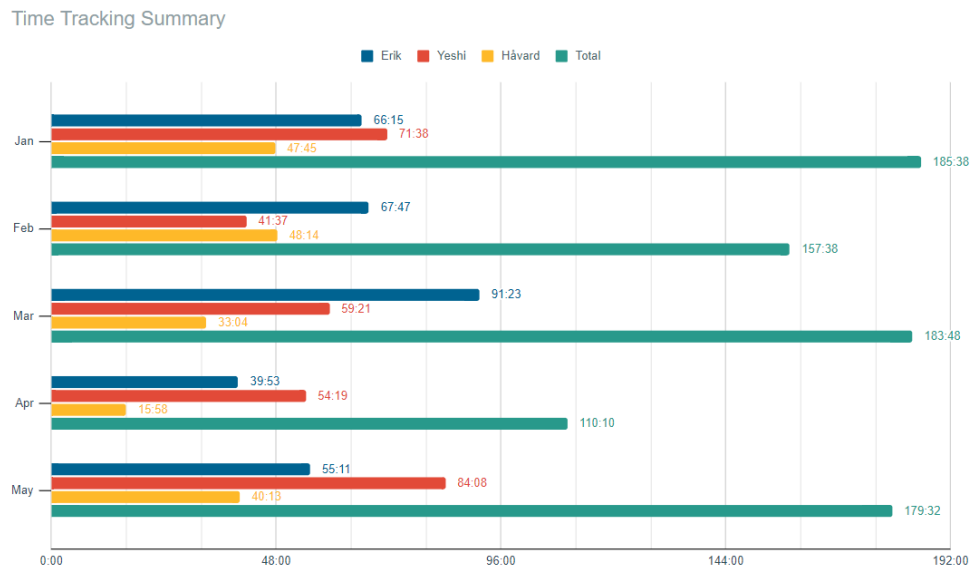


Figure I.2: Time tracking graph. These numbers represent how many hours we have worked.

