Handwriting Recognition on the Dead Sea Scrolls and IAM Dataset

Hemran Akhtari (s5424801)[1] Kyriakos Antoniou (s5715881)[1] Robert Power (s5332419)[1]
Group [09]
Handwriting Recognition (WMAI019-05) 2023-2024.2B
[1] Artificial Intelligence Department, University of Groningen
{h.akhtari, k.antoniou, r.power}@student.rug.nl

Abstract

This report presents two handwriting recognition pipelines for the Dead Sea Scrolls (DSS) and the IAM dataset. The DSS pipeline is a three stage process that first involves text line segmentation, followed by character segmentation, and finally character recognition. This pipeline leverages a mixture of traditional computer vision techniques and modern machine learning methods. Specifically, text line segmentation relied on a Hough-based method, character segmentation involved vertical projection, a ResNet variant and a hybrid method combing ResNet with Transformers was used for character recognition. These recognizers were pre-trained on a collection of available Hebrew fonts, and fine-tuned on the Monkbrill dataset. The data collection procedure involved various augmentation techniques, i.e. traditional image augmentation techniques as well as using a general adversarial network and a conditional variational autoencoder to generate large amounts of training required for our data-hungry recognizers. The results of the DSS pipeline are somewhat lackluster due to the high error in the segmentation process.

The link to our GitHub Repository: https://github.com/RobertPower7/Handwriting-Recognition

I. Introduction

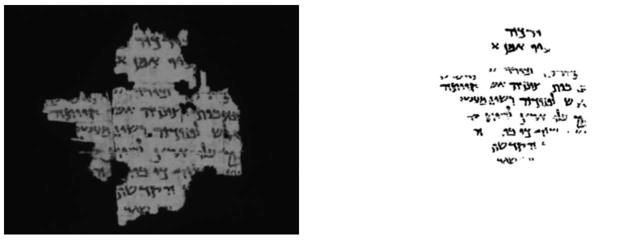
Handwriting recognition, or optical character recognition (OCR), is a pattern recognition task concerned with recognizing handwritten characters. OCR has played an important role in the preservation and analysis of historically relevant handwritten documents. The task of OCR, when performed by humans, typically requires domain experts and is highly labour-intensive. However, thanks to advancements in the broad field of artificial intelligence (AI), e.g., machine learning (ML), computer vision (CV), and natural language processing (NLP), it is now possible to digitize manuscripts in an automatic fashion. Despite these advances, OCR for historical documents poses a number of challenges compared to modern documents such as material degradation and script complexity (1). This report explores the task of OCR on a dataset of historical documents, the Dead Sea Scrolls (DSS) (2), as well as more modern documents, the IAM dataset (3).

II. Background

This section provides the background and relevance of the datasets that OCR is performed on. The DSS are introduced in subsection II-A and the IAM dataset is described in subsection II-B.

A. Dead Sea Scrolls

The first pipeline concerns the OCR of the DSS. The DSS are a set historical documents, dating back from 300BCE to 70BCE, that were discovered primarily in the Qumran Caves close to the Dead Sea. The DSS are historically relevant as they include religious, and some non-religious, material. This pipeline only concerns the OCR of a set of Hebrew scrolls from the DSS. An example of a DSS is depicted in Figure 1. In Figure 1a, one can observe a grey-scale version of the scroll. However, the DSS pipeline, takes binarized image as input, as depicted in Figure 1b.



(a) Grey-scale representation (b) Binarized representation

Fig. 1: Example of a Dead Sea Scroll

B. IAM Dataset

The IAM Handwriting Database contains forms of handwritten English text. The database was first published in (3) at the ICDAR 1999. The IAM-database as of October 2002 can be found in detail at (4). The database contains line segment images of unconstrained length which were scanned at a resolution of 300dpi and saved as PNG images with 256 gray levels. Noting that the actual dataset used was a fraction of the original dataset with the redacted images being used for marking this report. The original dataset consists of 9862 line images from 500 different writers, out of which it is split down to a train, 2 validation and a test set where one writer did not contribute to more than 1 set. An example of a line can be seen here 11

III. Methods

A. Text Line Extraction

Text line extraction simplifies the OCR process, particularly when dealing with historical documents like the DSS. The DSS have an irregular format with respect to the lines that a piece of Hebrew text belongs to. Therefore, segmenting each text line simplifies the character segmentation process. To achieve this, a Hough-based method for extracting text lines in handwritten documents (5) is employed, which has shown to have promising results on Hebrew scrolls (6).

The Hough-based method involves generating hypotheses of text lines in the Hough domain and validating these hypotheses in the image domain. This approach does not use any prior information for the orientation of text lines, allowing for multiple possible orientations within the scroll. The Hough transform (7) is used to detect lines in an image by converting points in Cartesian space, i.e. (x, y) coordinates in the image domain, to the polar space, i.e. (ρ, θ) coordinates in the Hough domain, according to:

$$\rho = x \cos\theta + y \sin\theta \qquad (1)$$

where ρ is the distance from the origin to the line, and θ is the angle between the x-axis and the normal to the line.

First, the connected components in the image are extracted with the method presented in subsection III-B. The the gravity centers (centroids) of each connected component in the image are calculated. These components represent a segment of text, which can correspond to an individual character, a part of a character, or a group of connected characters. Let $c_i = (x_i, y_i)$ denote the gravity center of the i-th connected component, where $i = 1, 2, ..., N$. The Hough transform is applied to these centroids to detect collinear alignments. For each gravity center, the set of lines intersecting that point for different discrete values of ρ and θ corresponds to a set of cells in the Hough domain. These cells are initialized to zero and incremented each time a centroid intersects a line. Cells with large values, i.e. high peaked cells of the Hough domain, indicate strong alignments.

To account for fluctuations in the text lines, two hypotheses are considered for each alignment. The first hypothesis is generated for the cell with the largest value, denoted as (ρ_0, θ_0). A cluster around this cell is created to search for the second hypothesis. With the resolution set to 1° along the θ direction, and $R = 0.2\bar{H}$ in the ρ direction, where $\bar{H}$ is the

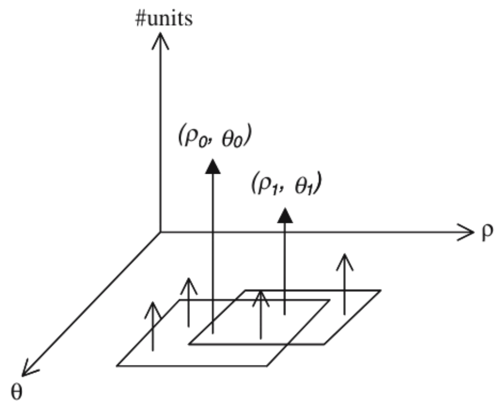


Fig. 2: Hypothesized cells in Hough space. Each peak represents aligned connected components. An alignment consists of connected components that belong to a cluster of cells surrounding a hypothesized cell (6).

average height of the connected components, the clustering factor f_{clus} is defined as:

$$f_{clus} = \begin{cases} \frac{\bar{H}}{R} & \text{if } 85^\circ \leq \theta \leq 95^\circ \\ 0.5\frac{\bar{H}}{R} & \text{if } 0^\circ \leq \theta < 85^\circ \text{ or } 95^\circ < \theta \leq 180^\circ \end{cases} \tag{2}$$

The cluster for searching for the secondary hypothesis (ρ_1, θ_1) is defined as a rectangular sub-domain, i.e.

$$\begin{aligned} x_0 = \rho_0 - f_{clus}, \quad x_1 = \rho_0 + f_{clus}, \\ z_0 = \theta_0 - \Delta\theta, \quad z_1 = \theta_0 + \Delta\theta \end{aligned} \tag{3}$$

where $\Delta\theta = 3^\circ$. This clustering approach allows for slight fluctuations in the line direction. The hypothesis with the strongest is alignment is chosen, and is considered the primary cell. The cell structure in Hough space for the hypotheses are illustrated in Figure 2.

Actual text lines do not always correspond to strong alignments. Hebrew is written in a horizontal fashion over many lines on a scroll. Therefore, the strongest alignments will be in the vertical direction and not the horizontal direction. The validation process ensures that the hypothesized text lines correspond to actual text lines in the scroll. A hypothesis is validated if it comprises fewer external neighbours than internal neighbours which ensures perceptual relevance within the image domain. Internal neighbors of a component C_i are the adjacent components on the hypothesized line, i.e. C_{i-1} and C_{i+1}, ordered by their x coordinate, whereas external neighbours are non-adjacent components within a certain radial distance δ_i, i.e.

$$\delta_i = \frac{d_l(c_{i-1}, c_i) + d_l(c_i, c_{i+1})}{2} \tag{4}$$

where d_l is the inter-component distance for adjacent components. The inter-component distance is based on the position of the component's bounding box relative to the hypothesized line. The inter-component distance for internal neighbours is the edge to edge distance (of the components' bounding boxes) along the hypothesized line. The inter-component distance for potential external neighbours, is computed as the edge to edge distance along a line joining the components' centroids. If this distance is less than δ_i for component C_i, then the component is considered an external neighbour for that hypothesized line. All components undergo an evaluation of whether or not they are an external neighbour of C_i for all i. If the line is validated, then the components of that alignment are removed from the Hough space and cannot be associated with future alignments, i.e. each component belongs to only one alignment. This procedure is performed iteratively until a stopping condition is met. A natural stopping condition is when every component belongs to an alignment.

B. Character Segmentation IAM and DSS

Following the text line extraction, the next step of OCR is segmenting each character from the text lines. Two methods have been used to extract the characters from the lines.

1) Vertical Projection: Firstly on line level we have used vertical projection (or vertical histogram) while further on at word level, we have used a further fine tuned vertical projection as well as the connected components method (8).

The vertical projection method is a simple but effective method where for each column the black pixels are counted. There are two main ways where this information are being used.

- Detection of white spaces between characters (We have used this as our word segmentation method)
- Indication of vertical strokes, where this indication along with logical approaches could indicate where a character begins and ends.

Lastly in order to make the most use out of the information a threshold needs to be set where we define the minimum magnitude of a vertical stroke that would be considered a segmentation point for words and characters.

2) Connected components: The connected components method is in short clustering connected pixels together. There are two connected neighborhoods (CN) used, the 4 pixel and 8 pixel connected neighborhood (CN). The 4 pixel CN and the 8 pixel CN where when one can imagine a 3x3 graph where our starting pixel is the one in the middle (position 2,2) the 4 pixel CN is assuming that the pixel above, below, left and right are part of the same neighborhood (positions 1,2 , 2,1 , 2,3 , 3,1) while on the 8 pixel CN all the pixels on the edge of 3x3 graph are neighbors of the middle pixel (position 1,1 1,2 , 1,3 , 2,1 , 2,3 , 3,1 , 3,2 , 3,3). We will be using the 8 CN method of connected components due to the nature of the curved letters in the line.

The method then uses the Spaghetti algorithm (9) for its 8-way CN that we will be using. A brief overview of the algorithm is as follows:

1) The algorithm goes through the image in a raster scan fashion, assigning temporary labels to our information carrying pixels and recording the equivalences between labels using the 8 way CN.

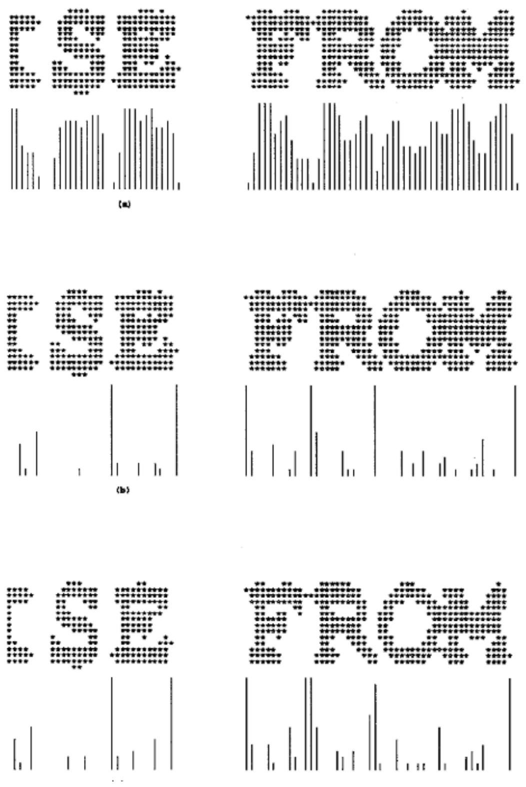


Fig. 3: Dissection based on projection: (a) Vertical projection of an image. It is an easy matter to detect white columns between characters, and regions of light contact. The function fails, however, to make clear the 0-M separation. (b) Differencing measure for column splitting. The function from [l] is based on a second difference of the projection. This gives a clear peak at most separation columns, but may still fail for the 0-M case. (c) Differencing measure after column ANDing. The image transformed by an AND of adjacent columns, with the difference function of (b) applied to the transformed image. The AND operation tends to increase separation, leading to a better defined peak at the 0-M boundary (8).

2) Next the algorithm goes for a second pass and uses the equivalence labels and a Union-Find Data Structure in order to assign a final label to all the temporary labels

The method then cuts and orders the separated labeled components in a raster scan order which was later adjusted to be ordered from left to right.

C. Character Recognition

Following the segmentation of lines and characters, this stage is dedicated to classifying each segmented character for further processing.

In this chapter, the methodologies and paradigms used for this character recognition component is being presented. It will give an outline on the datasets and data augmentation techniques used, as well as the chosen model architecture and training procedures.

1) Datasets: For the character recognition model, a two-stage training process was implemented: pre-training and fine-tuning. In the pre-training phase, a synthetic dataset was created using the provided Habakkuk font as the base. Similar fonts were manually searched and selected from the Google Fonts database to augment this dataset, as seen in Figure 4. This approach provided a diverse range of Hebrew characters, which builds a robust base model for further fine-tuning.

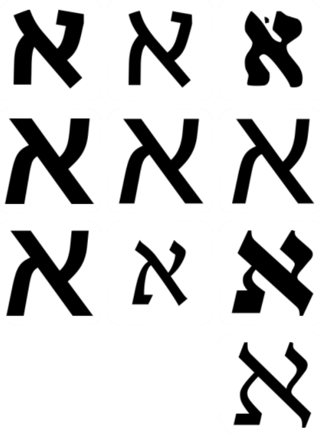


Fig. 4: Pre-Training: Synthetic Dataset using Fonts

In the fine-tuning phase, the Monkbrill dataset was used, which consists of 27 classes and a total of 5.537 samples, depicted in Figure 5. These classes contain handwritten Hebrew characters which were already segmented.

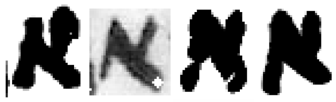


Fig. 5: Fine-Tuning: Monkbrill Dataset

2) Data Augmentation: To improve the performance and generalisation of the character recognition model, data augmentation was used to expand the Monkbrill dataset. This data augmentation process was carried out in two stages: Conditional Variational Autoencoder (cVAE) (10) and Generative Adversarial Network (GAN) (11).

Conditional Variational Autoencoder (cVAE) The cVAE was designed to generate synthetic images of Hebrew characters, which were used to augment the existing dataset. The architecture of the cVAE consists of an encoder and a decoder network, each playing a role in learning and generating the data distribution.

The encoder takes an input image of size 112x112 and its corresponding label, as seen in Figure 6. The label is embedded and then reshaped to match the input image dimensions. Next, the concatenated input goes through a series of convolutional layers with down-sampling followed by batch normalisation and activation function. Furthermore, a transformer block is incorporated to improve feature extraction, followed by dense layers to produce the latent space representation with a mean and a variance.

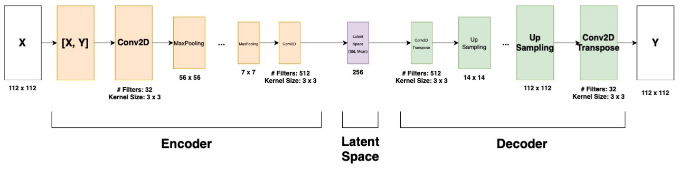


Fig. 6: VAE Architecture

The decoder on the other hand, receives the latent vector and the label as inputs, which is concatenated. The input is processed through dense-, convolutional and up-sampling layers to exactly mirror the encoder, before the final layer produces the output image.

The loss function of the cVAE model combines the reconstruction loss and the KL divergence loss:

1) Reconstruction Loss: The reconstruction loss $\mathcal{L}_R$ is calculated using binary cross-entropy between the original input image x and the reconstructed image $\hat{x}$:

$$\mathcal{L}_R = -\sum_{i=1}^{N} \left(x_i \log \hat{x}_i + (1 - x_i) \log(1 - \hat{x}_i) \right) \tag{5}$$

where x_i is the pixel value of the original image at position i, $\hat{x}_i$ is the pixel value of the reconstructed image at position i, and N is the total number of pixels in the image. This loss measures how well the reconstructed image matches the original input image, with the binary cross-entropy function evaluating the difference for each pixel.

2) KL Divergence Loss: The KL Divergence Loss $\mathcal{L}_K L$ measures how much the learned latent distribution deviates from the standard normal distribution. It is formulated as follows:

$$\mathcal{L}_{KL} = -\frac{1}{2} \sum_{j=1}^{L} \left(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2 \right)$$

where μ_j and σ_j^2 are the mean and variance of the latent variables at dimension j, and L is the dimensionality of the latent space. This loss ensures that the distribution of the latent variables z approximates a standard normal distribution, encouraging regularisation in the latent space.

The total loss $\mathcal{L}$ for the cVAE is the sum of the reconstruction loss and the KL Divergence Loss, scaled by the beta parameter β:

$$\mathcal{L} = \mathcal{L}_R + \beta \times \mathcal{L}_{KL}$$

where β is a weighting factor that balances the contribution of the KL Divergence Loss relative to the Reconstruction Loss.

To train the model, certain hyperparameters have been set:

- Image Shape: (112, 112, 1)
- Latent Dimension: 256
- Number of Classes: 27
- Batch Size: 32
- Learning Rate: 0.0001
- Beta (KL Divergence Term): 0.8

The Monkbrill dataset with some additional augmentation techniques was used to train the model, with a total training size of 13.500 data points.

Furthermore, to introduce more variability in the output of the data, noise has been added to the final latent space with a factor of 0.1. This ensured that the resulting samples per class were generated in different shapes and forms,

Generative Adversarial Network (GAN) The Generative Adversarial Network (GAN) was implemented to further augment the dataset by generating noisy images of Hebrew characters. The GAN architecture consists of a generator and a discriminator. These are both designed to compete with each other to improve the quality of the generated images. The generator takes a noise vector and a label as inputs. The label is then embedded and flattened to match the dimensions of the noise vector, which is then processed through several dense layers, batch normalisation and dropout layers. The output layer of the generator reshapes the data into the image dimensions with a tanh activation function to produce the synthetic image. The discriminator conversely, takes takes the image and label as input to further process it.

The loss function of the GAN consists of two parts:

1) Discriminator Loss: The discriminator loss evaluates how well it can distinguish between real and fake images. It is defined as follows:

$$A_i = y_i \log D(x_i) + (1 - y_i) \log(1 - D(G(z_i)))$$

$$\text{Discriminator Loss} = -\left(\frac{1}{m} \sum_{i=1}^{m} A_i \right)$$

where $D(x_i)$ is the output of the discriminator for real image x_i, $D(G(z_i))$ is the output of the discriminator for the generated image $G(z_i)$, y_i is the label (1 for real, 0 for fake), and m is the batch size.

2) The generator loss measures how well the generator can produce images that "fool" the discriminator. It is defined as follows:

$$\text{Generator Loss} = -\left(\frac{1}{m} \sum_{i=1}^{m} \log(D(G(z_i))) \right)$$

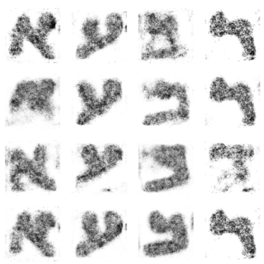
where $D(G(z_i))$ is the output of the discriminator for the generated image $G(z_i)$, and m is the batch size.

The total loss for the GAN is a combination of these two losses, where the discriminator and generator are optimised alternatively.

For training the GAN, the same augmented Monkbrill dataset that has been used for the cVAE was used here.

By running inference on the two models to generate data, the resulting dataset size was increased from 5.537 to 24.948 samples, which can be seen in Figure 7.

Other Augmentation Techniques Several non-generative augmentation techniques have been used as well to further introduce variety into the dataset:


(a) GAN

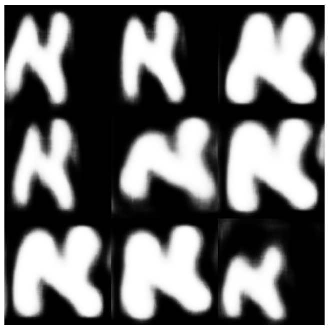

(b) VAE

Fig. 7: Generated Characters for Monkbrill Dataset

- Handcrafted Noise: Adds random visual elements such as dots, lines, or ellipses to the image, simulating various types of visual noise that can occur in real-world images.
- Gaussian Blur: Applies a Gaussian blur to the image, which smooths out details and reduces sharpness.
- Rotation: Rotates the image by a random angle within a specified range, introducing variation in the orientation of the image content to enhance model robustness to rotational variations.
- Shear: Applies a horizontal shear transformation to the image, distorting the shape of the content by slanting it, which helps the model become invariant to minor distortions in the training data.

3) Model Architecture: **CNN** The CNN model uses the ResNet50 model (12) as a base without pre-trained weights. Since the input of that specific architecture requires three channels instead of the current one channel, the model expands the input from one to three channels. After the forward-passes through the ResNet50, a global average pooling, batch normalisation and dropout have been added to reduce dimensionality and reduce overfitting. The final layer is a dense layer with a softmax activation function to classify the characters into 27 classes.

The model is trained using the categorical cross-entropy loss:

$$\text{Loss} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(\hat{y}_{ic})$$

where N is the number of samples, C is the number of classes, y_{ic} is a binary indicator (0 or 1) if class label c is the correct classification for sample i, and $\hat{y}_{ic}$ is the predicted probability that sample i belongs to class c.

CNN + Transformer For the hybrid approach, 4 transformer blocks have been added after the input has been reshaped with some additional normalisation and global average pooling layers subsequently.

4) Image Pre-Processing: To ensure consistent image shapes and features, a pre-processing pipeline has been created:

- Grayscale Conversion
- Inversion
- Upscaling to 224x224
- Normalisation [0, 1]
- Channel Expansion

5) Pre-Training: The pre-training phase uses the previously mentioned synthetic dataset, which was created using Hebrew fonts, consisting of 13.500 samples for training after augmentation. Additionally, the labels were adapted to the categorical cross-entropy loss by one-hot encoding them. The hyperparameters for the pre-training were defined as follows:

- Learning Rate: 0.001
- Epochs: 10
- Batch Size: 64
- Train-/Val Split: 0.2

6) Fine-Tuning: Fine-tuning is applied to refine the pre-trained model by adapting the weights as initial weights for the new training. Furthermore, the augmented Monkbrill dataset has been used for this stage which consists of 40.500 samples in total for training. Here, the labels were one-hot encoded as well. Since this is the fine-tuning stage, the learning rate has been reduces and the epochs increased:

- Learning Rate: 0.0001
- Epochs: 30
- Batch Size: 64
- Train-/Val Split: 0.2

Additionally, learning rate scheduling and early stopping were implemented to prevent overfitting and balance exploration vs. exploitation.

IV. Experimental Setup

A. Text Line Extraction

The text line extraction procedure was designed with a focus on generalization and robustness. This is due to the diverse distribution of scrolls in the DSS, with varying component sizes and text line orientations. The method presented in subsection III-A does not use any prior information to aid the text line extraction process, except during the generation of secondary hypotheses. In this step, the clustering factor (f_{clus}) computation is based on the angle θ of the primary hypothesis. The width of the sub-domain to search for the secondary hypothesis is halved if $0° \leq \theta < 85°$ or $95° < \theta \leq 180°$. Additionally, deviations of the primary hypothesis angle is explored, i.e. $\Delta\theta = 3°$.

Component size and quality also impact the quality of extracted text lines. Noisy components can lead to overestimated text lines and they affect the total external neighbours count which impacts validation. In an attempt to devise a robust solution, the extracted components were filtered based on their size relative to the average size of components. Components that did not meet the following criteria were not included in the Hough-based method:

(i) $C_{i,width} = 0.9 \times \bar{C}_{width}$ or $C_{i,height} = 0.9 \times \bar{C}_{height}$;
(ii) $C_{i,width} = 0.8 \times \bar{C}_{width}$ and $C_{i,height} = 0.8 \times \bar{C}_{height}$
$\forall i = 1, 2, ..., N$

where $\bar{C}_{width}$ and $\bar{C}_{height}$ are the average widths and heights of all extracted components, respectfully. Additionally, to further mitigate the issue of noisy components, the validation criterion was adjusted as follows: a line is validated if:

$$|C_{internal}| < \frac{|C_{external}|}{1.5} \text{ and } |C_{internal}| > 0$$

Another issue regarding the component size arose when determining which components (bounding boxes of the components) are actually intersected by the line, i.e. belong to the primary hypothesis alignment. Small components that perceptually belonged to the line were not intersected. To account for this, a tolerance factor ϵ was introduced to search for components within $\Delta\epsilon$ of the hypothesis line. This hyperparameter also greatly impacts the quality of extracted text lines. If ϵ is too small, then the small components that perceptually belong to the text line are ignored. If ϵ is too large, then components that perceptually belong to other lines are wrongfully included in the alignment. For generalization purposes this tolerance was set to:

$$\epsilon = \frac{\bar{C}_{height}}{4}$$

With the above settings, the extracted text lines are somewhat valid. However, we found that more directional bias, specifically bias in the horizontal direction, aided in the quality and efficiency of the text line extraction process. The lines in the scrolls are mostly straight, except for some rare exceptions where there are what seem like notes written in very diagonal directions. Therefore, the set of generated hypotheses could only be considered a primary hypothesis if $80° \leq \theta < 100°$ or $265° < \theta \leq 285°$. This limited the amount of hypotheses generated which accelerated the text line extraction process.

A final note on implementation is the case where a text line is invalidated. At the beginning of the process, a number of hypotheses are generated, if a line is validated, then a new set of hypotheses are to be generated. If the line is invalidated, and we generated a new set of hypotheses, it is likely to lead to a similar alignment being generated and selected as the previous iteration and therefore being invalidated again, and becoming stuck in a loop with no termination. Instead, if a line is invalidated, we select the next best line in the latest set of generated hypotheses and perform the Hough-based method with that alignment. If we run out of text lines that are all invalidated, then we need to consider weaker alignments and the threshold (φ) controlling the Hough line extraction is halved, i.e.

$$\varphi \leftarrow \frac{\varphi}{2}$$

The set of extracted text lines (with their associated components) is ordered by the bottom left coordinate of the bounding box of the leftmost component on the text line. The text lines are ordered along the y-axis. The output is an image with the same width as the original scroll image, with characters located at their respective positions based on the bottom left coordinate of their bounding box. This ensures that the text lines and connected components are correctly sequenced and spatially accurate for further processing in the character segmentation model.

B. Character Segmentation for IAM and DSS

The character segmentation pipeline was split into 5 distinct tasks.

1) Pre-processing the initial image line
2) Separation of words and white space
3) Weak threshold vertical projection cut
4) Connected components separation
5) Recursively increased threshold for a vertical projection cut

Before any character segmentation is done the line image of the image is pre-processed in such a way that the maximum height of the image is at 300 pixels for the DSS dataset and 284 for the IAM, the aspect ratio for both dataset line images was kept the same. The image was further processed with the closing, opening and eroding morphological operations with a kernel of a 3x3 square. This morphological operations were done to fix characters that had thin cuts, to remove small noise from the image. The erosion was done on the background of the characters which led to a similar effect to dilation on the characters but with the characters being able to keep a better shape.

We then began by defining what a white space is for both tasks. We have defined that a white space is sixty (60) vertical projection lines in a row that have magnitude smaller than a five hundredth (0.05) of average vertical projection line. When a white space is found it is passed forward to the model as a flag for the ”space” character. This allowed us to split the line of text into images of words. In addition a less strong threshold of two tenths (0.2) was used to separate white spaces between characters in a word, this threshold was weak enough to not cut any lines but strong enough to have no issue with noise.

Next, we now have increased the threshold to 1/3, this threshold now begins cutting vertical projection lines that begin to resemble connected characters. This is meant to cut characters where the pen has began lifting from the paper but still makes a small conduct, accidentally connecting characters together albeit easily distinguishable characters. Due to the increased threshold a line was selected if there has not been a selected line in the last 60 vertical projections for both the IAM and DSS

Subsequently, we used the connected components method to separate characters which are not connected but are impossible to separate using the vertical projection method due to the large amount of pixels crossing from one character to another but are still separable by curly white space lines.

Lastly, as all the easily separable characters have been separated we are left with characters that have been drawn using a movement that did not lift the pen from the paper making them hard to separate. This characters are mostly found in cursive writing which there is a last exploit that can be used albeit not always working. While cursive characters are written at the begging and ending of the character the pen usually goes up and down the paper just like the accident discussed in step 3. Unlike the mistake which is easily separable due to the quick pull up on the pen this connected characters are connected by smooth lines that increase and decrease in thickness as the writer is moving from character to character in order to make them easily distinguishable in even first time readers. Thus we begin our approach by slowly increasing the threshold at which vertical projection lines can be deemed as the ones to cut, the recursive threshold increasing stops when the image is smaller than the largest character found in training which was 210 pixels long for the IAM dataset and 130 pixels for the DSS dataset. Finally the minimum 60 line vertical projection threshold to select a vertical projection line to cut was lowered to 20 for the IAM dataset due to the smaller connected characters but it was kept the same for the DSS scrolls as characters were the same size when they were connected or not.

C. Character Recognition

Levenshtein Distance, also known as Edit Distance, is a measure of the minimum number of edits of single characters (insertions, deletions, or substitutions) required to change one string into another. It is a useful metric for evaluating the output of our character recognition model, as it quantifies the difference between the recognized text and the ground truth. The Levenshtein Distance d between two strings s and t can be defined as:

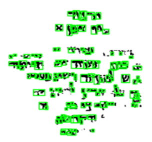
$$d(s,t) = \begin{cases} |s| & \text{if } |t| = 0 \\ |t| & \text{if } |s| = 0 \\ \min\{d(s_{1\dots|s|-1}, t) + 1, & \\ d(s, t_{1\dots|t|-1}) + 1, & \\ d(s_{1\dots|s|-1}, t_{1\dots|t|-1}) + 1_{[s_{|s|} \neq t_{|t|}]}\} & \text{otherwise} \end{cases} \tag{6}$$

In OCR applications, a lower Levenshtein Distance indicates higher accuracy, as fewer edits are needed to match the OCR output to the ground truth text.

V. Results

A. Text Line Extraction

The extracted components from Figure 1b are illustrated in Figure 8a. This example shows very broken characters. In total, 137 components were extracted. The ground truth label is 138 characters. Numerically, the extraction is close to the ground truth, but perceptually, many characters are undetected and there are many noisy components.


(a) Extracted components

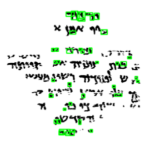

(b) Undetected components

Fig. 8: Extracted components and text lines from the Hough-Based method

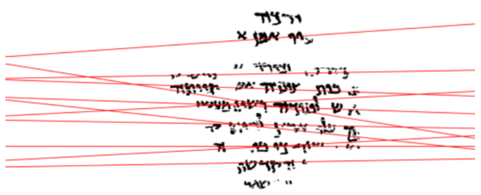


Fig. 9: Extracted lines

The extracted lines are depicted in Figure 9. The quality of extracted text lines depends on the quality of the extracted components. For this image, the text line extractor outputted 10 text lines, while the ground truth label is 9 lines. Numerically, the result is close to the ground truth. However, perceptually, some true lines were extracted, but the top and bottom lines were undetected, and many lines were very diagonal and intersected, leading to inaccuracies in the text line extraction. The remaining components that were not extracted are shown in Figure 8b, further demonstrating the perceptual discrepancies in the pipeline thus far. Examples from the output are displayed in Figure 10. The text line extraction trial ran for 3 minutes 10 seconds before it terminated due to reaching the user-specified maximum number of iterations; which was set to 300. This trial was performed on a Windows device with an Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz processor and 8GB of RAM.

(a) The output of the extracted line that intersects the second line in Figure 9. We can observe that the first text line has not been detected.

(b) The output of the extracted line that intersects, what perceptually appears to be, the third and the fourth text lines in Figure 9.

Fig. 10: Example text lines from Figure 9 that have been extracted.

B. Character Segmenation for IAM and DSS

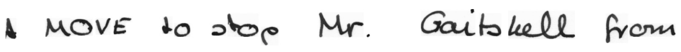


Fig. 11: Original Line Image from IAM

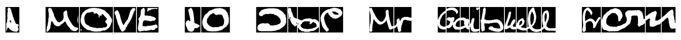


Fig. 12: Concatenated Segmented Characters Image of IAM

Looking at the original IAM line image 11 and comparing it to the concatenated image 12 we can see that both large characters like the M from MOVE and small characters like

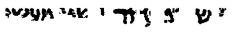


Fig. 13: Extracted Line Image from DSS

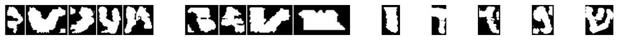


Fig. 14: Concatenated Segmented Characters Image of DSS

the t from to we can see that they have been successfully cut and kept all of there information. Furthermore we can see that connect characters such as the to from stop and the Mr from Mr we can see that they were also successfully cut. Although that is not the case for the name Gaitsbell where we can see that the G and a have been cut in half as the model was unable to find the correct place to cut the connected characters due to the slightly tilted and overlapping charters when seen in the vertical projection, while the connected components also failed due to the two characters touching each other. In addition we can also see very good cuts on the bell part of the Gaitsbell name showing that vertical projection works even when the characters are connected and have a thin line connecting them which is unfortunately not the case for the om part of from at the end where part of the m was cut. Lastly but not least we can also see that the trade-offs of having large characters not being cut like the M, we can see the ts from the name Gaitsbell as well as the fr part of from being kept together as there length was smaller than that of the largest character M of this line.

Next looking at the extracted line image from DSS scrolls 13 and comparing it to the Concatenated Segmented Characters of the same Image 14 we can see that the character segmentation is easily able to segment the characters even when they are not sharp but we can see that there are still issues with long characters as we can see the fourth character from the end has been cut in half with little resemblance to the original character.

C. Character Recognition

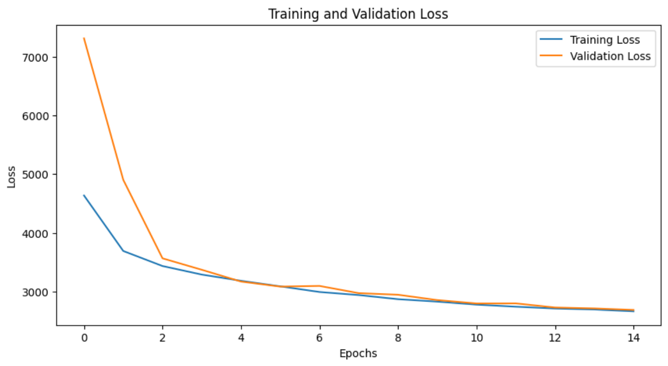
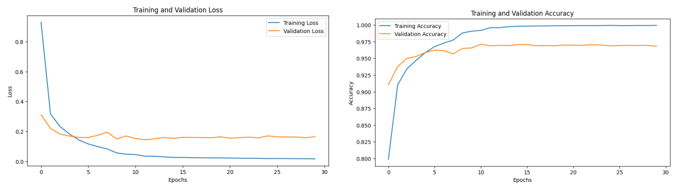


Fig. 15: Conditional Variational Autoencoder Loss

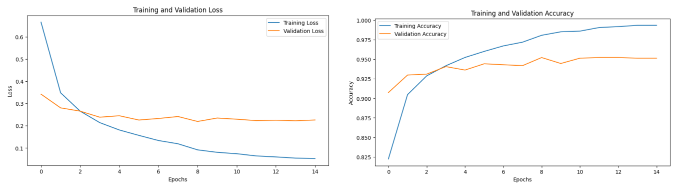
D. Entire Pipeline

Given the scroll depicted in Figure 18, the pipeline achieved a Levenshtein distance of 324.



(a) Loss (b) Accuracy

Fig. 16: Character Recognition Model Evaluation (CNN)



(a) Loss (b) Accuracy

Fig. 17: Character Recognition Model Evaluation (CNN + Transformer)

VI. Contributions

A. Hemran Akhtari

Worked on the DSS and IAM pipelines:

- Performed research on Character Recognition and data augmentation tasks
- Implemented the Charachter Recognition component
- Implemented the GAN and cVAE data generators
- Implemented the overall pipeline with assistance
- Report:subsection III-C,subsection V-C

B. Kyriakos Antoniou

Worked on the DSS and IAM pipelines:

- Performed reasearch on Character Segmentation methods as well as End to End Solutions for the IAM that were not used due to time limitations
- Fully implemented the character segmentation pipeline.
- Assisted with design choices and ideas on the overall pipeline
- Report:,subsection III-B,subsubsection III-B1, subsubsection III-B2,subsection IV-B, subsection V-B

C. Robert Power

Worked on the DSS pipeline:

- Performed research and took part in design decisions for the overall pipeline.
- Fully implemented the text line extraction model.
- Report: Abstract (DSS), section I, subsection II-A, subsection III-A, , subsection IV-A, subsection V-A.

References

[1] T. L. Tobing, S. Y. Yayilgan, S. George, and T. Elgvin, "Isolated handwritten character recognition of ancient hebrew manuscripts," in *Archiving Conference*, vol. 19.

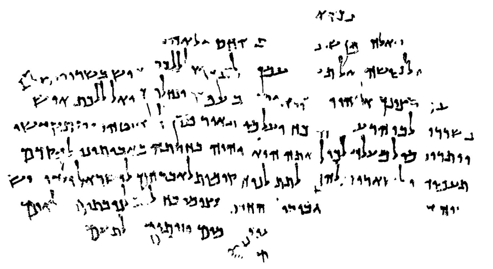


Fig. 18: Sample of a DSS Scroll

Society for Imaging Science and Technology, 2022, pp. 35–39.

[2] Israel Antiquities Authority. (2024) The leon levy dead sea scrolls digital library. Https://www.deadseascrolls.org.il/.

[3] U.-V. Marti and H. Bunke, "A full english sentence database for off-line handwriting recognition," in *Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR'99 (Cat. No. PR00318)*. IEEE, 1999, pp. 705–708.

[4] ——, "The iam-database: an english sentence database for offline handwriting recognition," *International Journal on Document Analysis and Recognition*, vol. 5, pp. 39–46, 2002.

[5] L. Likforman-Sulem, A. Hanimyan, and C. Faure, "A hough based algorithm for extracting text lines in handwritten documents," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 2. IEEE, 1995, pp. 774–777.

[6] L. Likforman-Sulem, A. Zahour, and B. Taconet, "Text line segmentation of historical documents: a survey," *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 9, pp. 123–138, 2007.

[7] P. V. Hough, "Method and means for recognizing complex patterns," Dec. 18 1962, uS Patent 3,069,654.

[8] R. Casey and E. Lecolinet, "A survey of methods and strategies in character segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 7, pp. 690–706, 1996.

[9] F. Bolelli, S. Allegretti, L. Baraldi, and C. Grana, "Spaghetti labeling: Directed acyclic graphs for block-based connected components labeling," *IEEE Transactions on Image Processing*, vol. PP, pp. 1–1, 10 2019.

[10] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2022.

[11] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.