Hand Writing Recognition and Classification on Dead sea scroll Data (Group 5)

Hari Vidharth
s4031180

Krishnakumar Santhakumar
s4035992

Dhawal Salvi
s4107624

Ashwin Vaidya
s3911888

University of Groningen

Abstract

1 Introduction

2 Methods

Methods section has three major distinct processes on the dead sea scrolls data. These process includes segmentation to segment characters from the input Hebrew documents, the recogniser to identify the characters in the document and finally the style classifier to identify the period of the document. The initial segmentation part is further sub-divided into two major categories the line segmentation and character segmentation.

2.1 End to End System Pipeline

The end to end system pipeline of the Dead Sea Scrolls handwriting recognition and classification system is shown by Figure 1. The image of the document is given as the input to the system. Each image is passed one at a time to the system and two outputs are generated for each image.

First, line segmentation is performed on the input image, the image is further segmented into (l) images, where l is the number of lines identified in the main image, that is each new image represents a line from the main image.

Next, character segmentation is performed on the segmented line images, the image is further segmented into (c) images, where c is the number of characters identified in the line image, that is each new image represents a character from the line images. So the total number of images generated will be l x c in the format of Root Folder $->$ Line Folders $->$ Line Images $->$ Character Images.

After segmentation, using the segmented characters the character recognition system predicts and identifies the characters and transcribes them into the Output.txt file which is the first output of the system.

Finally, the style classification part of the system uses the segmented characters and the recognised characters to predict and classify the period style of the document and writes to the Output.txt file the style of the document which is the second output of the system.

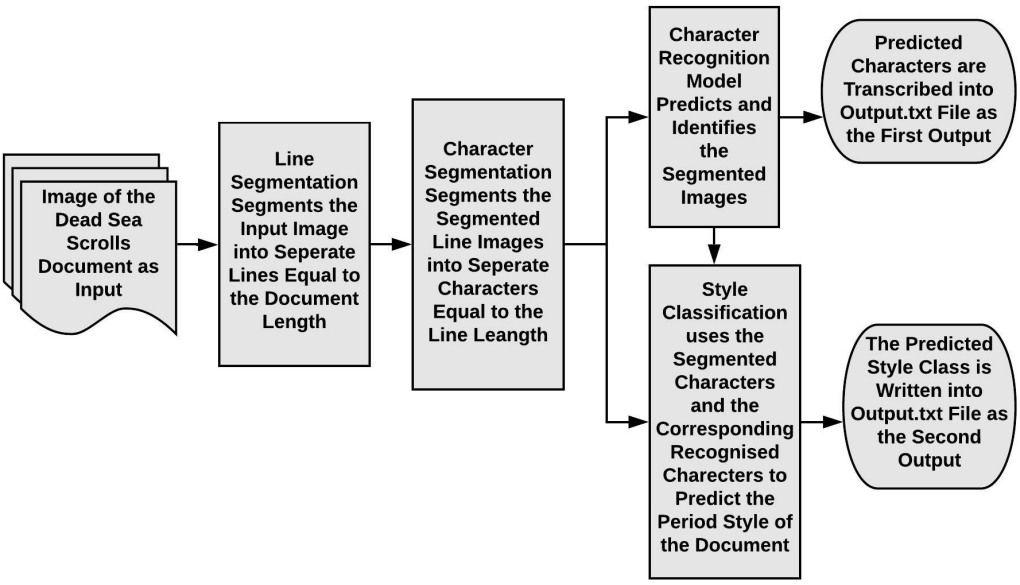


Figure 1: System Architecture

2.2 Line Segmentation

In order to segment the characters from the document, we first segment the lines from the document. The input documents (figure 2) for the segmenting the lines and characters are in binarised format. The segmentation of document into lines involves two steps rotation and line separation. Segmented lines will then used as an input for the character segmentation task to segment the characters from these lines, which is explained in section 2.3.

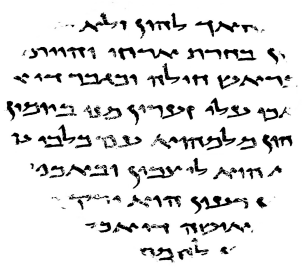


Figure 2: Input document

2.2.1 Rotation

Initially, before segmenting the lines from the document we rotated the document to make the all the lines to align in straight line in-order to achieve better segmentation. Most of the documents we tested are non skewed. Even though those documents are non skewed, some document has the lines which are not straight. So we did the 2D geometric rotation on the images based on the documents center of mass (similar to the slope correction method proposed [2] for Arabic documents) for the angle ranges

from -10 degree to 10 degree with the threshold of 0.1 increment. These angle values are determined based on different experiments on different test images, angles with more than these degrees show poor while computing projection profiles. Based on the angle of rotation, the character objects in the document get shifted to it's nearby pixels by producing 2x3 transformation matrix. Then the image is reconstructed using affine transformation on the constructed transformation matrix. These geometric rotation on the input document is achieved using *opencv* python library functions *getRotationMatrix 2D* and *warpAffine*. For each angle of rotation corresponding horizontal projection profile will be calculated.

Figure 3: The result of geometric rotation on the input image and it's best suited angle orientation.

Horizontal projection profiles (HPP) are calculated to find the each lines height in the document, the lines which has more text shows high projection profile peaks compared to the lines with white spaces. Before finding the horizontal projection profiles the character edges are found using the sobel edge detector, which help to compute HPP by making every pixel other than characters to zero (black). Figure 13 shows (in appendix A.1) the computed horizontal projection profile on the input document for the angle of 0.10 degree. After computing HPP for all the angles, select the appropriate angle by choosing the maximum horizontal projection peaks found among the results obtained for various angels. Then finally rotate the input document according to the identified angle of rotation and compute it's horizontal projection profile and peak hills. Figure 3 shows the output of geometric rotation for the given input document.

2.2.2 A* path planning

After finding the suitable orientation angle, the lines associated with the new rotated document is identified using A* path planning algorithm [5]. This done by generating the binary map cluster for the lines identified using the horizontal projection profiles peak regions (as shown in figure 13) with the segmenting threshold value of 5. This segmenting threshold value is an hyper-parameter, which divides the high peak regions and low peak regions in the horizontal profile. In order to separate the lines which has the merged characters, the regions with merged characters are found using the sliding window with the window size of 25 across the lines regions. If there are no characters found in the intermediate regions the path is retained else the block with the windows size is stored as an separate segments, by grouping the blocks found in that particular line. Then we applied an A* path planning algorithm [5] with the start and goal stage as beginning and ending of the lines found in horizontal projection profile clusters.

We used the squared Euclidean distance to find the distance between the start and goal of the line. The closest distance between the start to end pixels are found using the heap queue priority algorithm (a binary tree algorithm, using *heapq* python library) which stores the least value in the root node compared to it's child nodes. The least distance is found by taking the least distance scores in the heap queue list along with the distance value computed between the starting point of the line and it's neighbouring pixels. The neighbours include the squared pixel values of $(0, 1)$, $(1, 0)$, $(0, -1)$, $(-1, 0)$, $(1, 1)$, $(-1, -1)$, $(1, -1)$ and $(-1, 1)$, the starting point of the line get update until the final goal pixel value in the image is reached. The neighbour with the least distance is taken as the starting point of the next iteration. This process is carried out for all the lines found in the document, figure 4 shows the result of A* algorithm on the input document (figure 2). Once the shortest line paths are found for all

Figure 4: Result of A* path planning algorithm on the input document.

the lines, the lines are segmented out based on the upper and lower boundary path between the lines. To segment out the final line we added an empty line at the end of the document after A* segmentation. We used the article [1] for the implementation of A* star path planning and road block technique.

2.3 Character Segmentation

The character segmentation follows line segmentation. In this step we extract all the characters from each line image. This extraction is done in two steps. The first step involves doing crude segmentation by using connected filters. And in the following step we use template matching to refine the output of the previous step.

2.3.1 Connected Filters

Initially, the line image is thresholded using Sauvola Thresholding [4]. It is a local thresholding algorithm which uses a specific formula to do image binarization. Usually, Sauvola Thresholding is used to binarize an image when the background is uneven. Given that the line images are already binarized, this step seems superfluous. But from our experiments we found that using this with a very small kernal of 3x3 improved the accuracy when finding the characters.

Next, we find the contours, which are lines joining the boundaries of all the pixels which have the same intensity. These contours help in extracting the bounding boxes. We also do a check on the area of the bounding boxes to ensure that it does not select an artifact on the paper or that it does not end up putting a bounding box around the entire line. The results of this step are shown in the Figure 5.

(a) (b) (c) (d)

Figure 5: Images extracted by Connected Filtering

[1]The implementation of A* path planning and road block are done with the help of the article *"segmenting lines in handwritten documents"* by muthu krishnan (`https://muthu.co/segmenting-lines-in-handwritten-documents-using-a-path-planning-algorithm/`)

2.3.2 Template Matching

As can be seen from Figure 5 not every character is extracted perfectly during this stage and another step is needed to ensure that all the characters are segmented out correctly. To achieve that, we then use template matching.

At this point we are unsure that the extracted image contains only a single character or a combination of them. We use the labeled characters given in the dataset and match each character with the segmented image. To do the match we first scale the template image to match the height of the extracted image. The best match from all the templates is isolated and the segmented image is split around this region. The unmatched region is iteratively split and matched to the character images. From our experiments we found that template matching on words and single characters gives significantly accurate results. This way we get a finer segmented image containing only the extracted characters. Figure 6 shows a sample of images extracted using template matching. These images are then sent to the next stage for identification.

(a) (b) (c) (d) (e)

Figure 6: Images extracted by Template Matching

It can be observed that template matching stage preserves individual characters and refines aggregates. The characters in Figure 5d are segmented into two as shown in Figure 6d and Figure 6e. And while in Figure 6e the character has been truncated our experiments found that the character recognition neural network is able to classify this accurately. Thus as a whole the pipeline remains robust to individual errors.

A point for consideration is that not all extraction is perfect in this stage. Hence, in our code we do a checks such as threshold on the ratio of the area of segmented characters to the original aggregate. Further, a check is done on the ratio of width to height of the images obtained from the first stage. This is based on the observation that images with height to width ratio greater than a 1.2 contain single characters and are not processed further.

2.4 Character Recognition

2.4.1 Preprocessing Images

The characters can be rectangular (Zayin) or square (Alef) in shape and if we choose 50x50 as an input size for the Neural Network, then the image can become distorted. To resolve this, we first resized each character class with its mean resolution. This lead to each character class having its common resolution. Now to make one final resolution for all character class, the mean resolution was taken per character class and standard deviation was subtracted to its width and added to its height. Hence, the final resolution was 32x49 for all character class and became input size for Feed Forward Neural Network. Figure 7 shows the result after processing images.

(a) (b) (c) (d)

Figure 7: Preprocessing result, (7a) original Alef, (7b) processed Alef, (7c) original Zayin and (7d) processed Zayin

2.4.2 Feature Extraction

Convolutional Neural Network is powerful tool for extracting features from handwritten characters and classifying using softmax making effective technique for Character Recognition Process [1]. Thus, after Processing the Images, Alexnet Architecture [3] was used for feature extraction containing 8 layers (5 Convolutional Layers and 3 Fully Connected Layers). Further, we choose Alexnet because it is not too deep nor too shallow architecture as or training contains fewer examples. Hence, if the architecture is too deep then it can lead to overfit and vanishing gradient [6].

2.4.3 Traning Alexnet

To complile the model, we have used *Adam* optimizer, *categorical crossentropy* loss, *relu* activation function. Moreover, to fit the model we have use early stopping with validation split as 10%. However, with this configuration (Model 1) we achieved 32.24% validation loss & 94.58% as validation accuracy as seen in Table 1. Therefore, to decrease the validation loss, first we introduce dropout with 0.1 after second dense layer and decreased the learning rate from 0.001 to 0.0005 (Table 1 Model 2). To further decrease the validation loss, we added two convolution layer with relu activation and maxpooling (Model 4) and achieved 14.56% validation loss & 96.39% as seen in Table 1. In addition Table 1 Model 3 shows the result of adding one convolution layer with relu activation and maxpooling.

Models	Validation Accuracy	Validation Loss
1	94.58%	32.24%
2	95.67%	22.17%
3	95.13%	18.09%
4	***96.39%***	***14.56%***

Table 1: Result of Different Models of Alexnet Architecture

2.4.4 Character Classification

After training the model, the classifier takes the segmented character and output the unicode of the character class to print in the text file. However, the segmented character may contain noise, such as, segmented characters containing partial features and sometimes the width being 30 times more greater than its height. So, to resolve this we set two threshold. Which are, if the height to width ratio is greater than 0.23 and if the predicted character by the model have confidence smaller than 25% then skip the character. Figure 8 shows the result of the same.

(a) (b) (c) (d)

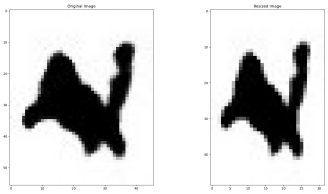
Figure 8: Classification Result after character segmentation. (8a) Hand-written Lamed, (8b) Predicted Lamed, (8c) Hand-written Tet and (8d) Predicted Tet

2.5 Style Classification

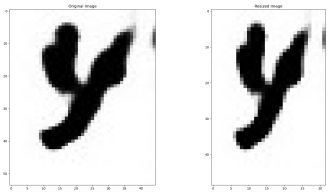
2.5.1 Pre-Processing

The input images for the training data is available in different sizes. To provide a more robust extraction of features, before the feature extraction process we resize the image to a constant size so that all input images are of a fixed size. The size we choose for reshaping is 32x49 the reason as mentioned in the Character Recognition Preprocessing step (2.4.1). To resize the image bicubic interpolation was used, this method considers 16 pixels (4x4) into account during sampling, even though the processing time is a bit longer, the interpolated surface of the re-scaled images are much smoother and have fewer

interpolation artifacts. Figure 9 shows two examples for the *Alef* and *Ayin* characters, the original image before preprocessing and the resized image after the preprocessing.



(a) Alef
Left: Original Image
Right: Resized Image



(b) Ayin
Left: Original Image
Right: Resized Image

Figure 9: Examples for Style Classification Pre-Processing

2.5.2 Feature Extraction

The next step after preprocessing the images is extracting the features from the images. For this feature extraction process, we use the VGG-16 architecture, this model was implemented from scratch in Keras and then trained on the preprocessed images. If N represents the number of layers in the model architecture then N-3 layers were used for the feature extraction process. This means that from the VGG-16 model architecture all the dense layers at the end were not used, all the previous convolution and pooling layers were retained. Finally after flattening the outputs from the previous layers the model is ready for the feature extraction process. Figure 16 shows the examples of the high level and low level features extracted from by model.

2.5.3 Training

The style classification training dataset is a highly unbalanced dataset with a low amount of training data available. Directly using a neural network or deep learning model to train, will not work as there will be high chances of overfitting. Hence to counter this we used a deep learning model for the feature extraction and coupled that with a machine learning model to provide the final classification. First, the data is split into a training set and testing set, 90% as train data and 10% as test data, as cross-validation is done here there is no separate data reserved for validation. As mentioned earlier the dataset is highly unbalanced, to counter this imbalance we use a stratified train test split method, this method ensures there are an equal proportion of the classes in the training set and the testing set so that the model does not overfit on any particular class due to the imbalanced dataset during the train test split. Again to counter this class imbalance and less amount of training data boosting algorithms are known to work best for classification. We used a stacking combination of gradient boosting algorithm and histogram gradient boosting algorithm, and once again the gradient boosting algorithm as the final classifier. For this, we used the pass-through method, which means that in the stacked model the next model will not only learn from the previously learnt representations but also the training data simultaneously. Finally, this model was trained with a stratified cross-validation technique with a CV of 2, because many classes have fewer data and when training in a stratified fashion, any higher number exceeds the max splits done by cross-validation.

2.5.4 Classifier

The above-mentioned model was trained to predict one of the three classes *Archaic*, *Hasmonean*, *Herodian* on each of the characters individually and the learnt representations saved as model files resulting in 27 .pkl files, one for each character. Using the segmented images and the character recognition model from the previous sections, the segmented images are first recognized and the corresponding character model is called and using that model the final style classification is performed on the segmented image. After all the segmented images are classified final voting takes place to classify the document into the respective period style class.

3 Results

3.1 Line Segmentation

Figure 10, shows the final segmented characters of the input document, with individual lines segmented into individual blocks.

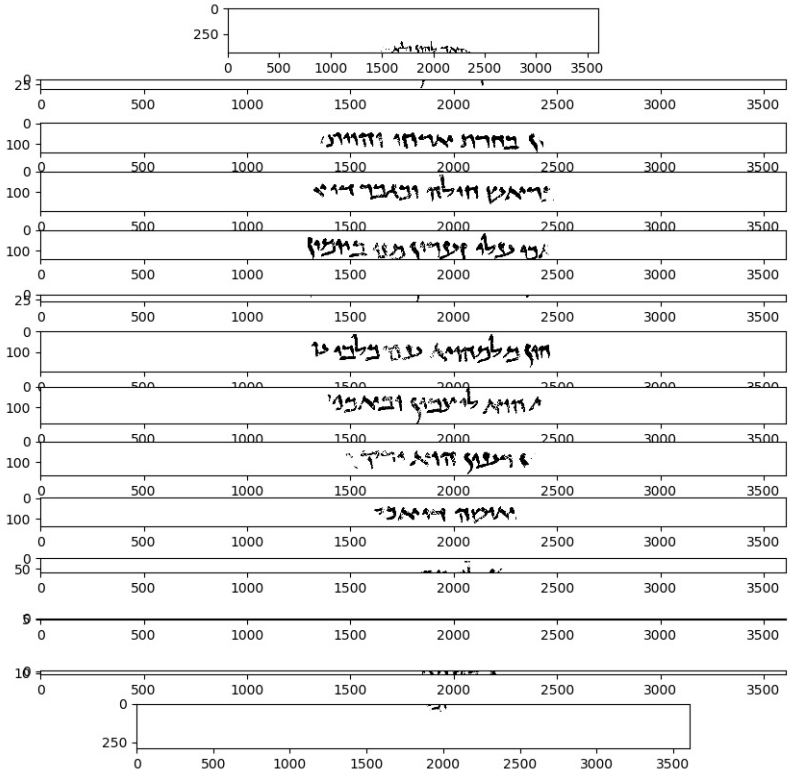


Figure 10: Final line segmented results after the geometric rotation and A* path planning on the input document (figure 2).

3.2 Character Segmentation

(a) (b) (c) (d) (e) (f)

Figure 11: Comparison of extracted characters from the line

Figure 11a shows the original line which was used as the input for character segmenter. On the right, we can see the output of the segmentation stage.

3.3 Character Recognition

Figure 12 shows the result of the input document (Fig. 12 (a)) and the output (Fig. 12 (b)) after Character Segmentation.

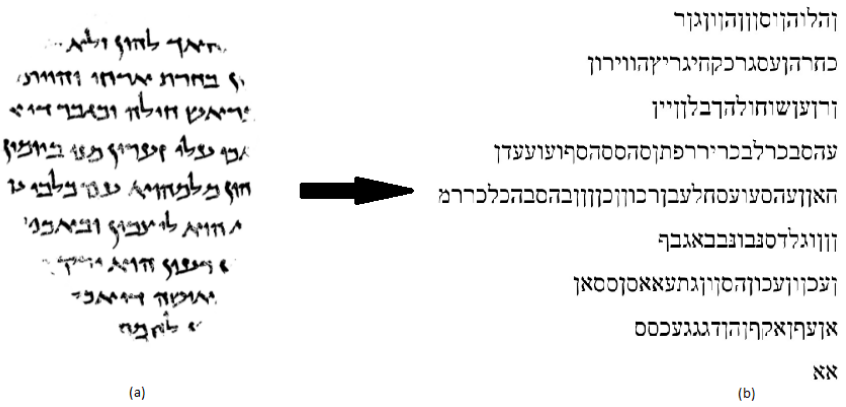


Figure 12: Result of Character Recognition Model on the input Image.

3.4 Style Classification

Table 2 shows the final results of the stacking classifier used for the classification of the document.

Character / Stacking Classifier	Train Accuracy	Test Accuracy	Train Loss	Test Loss
Alef	100%	100%	0.000203	0.000688
Ayin	99%	91.6%	0.00898	0.0743
Bet	91%	85.7%	0.00001709	0.5775
Dalet	100%	100%	0.004082	0.01799
Gimel	100%	100%	0.00001977	0.0001572
He	100%	100%	0.00004129	0.0003827
Het	100%	100%	0.00000169	0.00000340
Kaf	91.44%	83.83%	0.4318	0.5547
Kaf-final	100%	100%	0.001049	0.00004361
Lamed	98.07%	100%	0.07472	0.01104
Mem	98.19%	84.61%	0.03002	0.2203
Mem-medial	100%	80%	0.000021901	0.1458
Nun-final	96.77%	90.90%	0.3106	0.3451
Nun-medial	100%	100%	0.0000219	0.0000219
Pe	100%	90%	0.000016963	0.08827
Pe-final	97.77%	100%	0.1226	0.00017859
Qof	97.93%	100%	0.2044	0.00013736
Resh	98.19%	100%	0.01704	0.00003122
Samekh	100%	100%	0.0002609	0.002121
Shin	100%	100%	0.000007894	0.000053674
Taw	97.05%	100%	0.04353	0.00017343
Tet	96.66%	90%	0.1463	0.9745
Tsadi-final	100%	100%	0.000042938	0.000088895
Tsadi-medial	100%	100%	0.000035019	0.000065468
Waw	100%	100%	0.000023321	0.0001026
Yod	100%	86.66%	0.000091521	0.3375
Zayin	100%	100%	0.000024528	0.000028214

Table 2: Style Classification Stacking Classifier Final Results

4 Discussion

5 Contributions

S.No	Student number	Name	Contribution
1	s4035992	Krishnakumar Santhakumar	Line Segmentation
2	s3911888	Ashwin Vaidya	Character Segmentation
3	s4107624	Dhawal Salvi	Character Recognition
4	s4031180	Hari Vidharth	Style Classification

Table 3: Individual contributions

References

[1] Mayur Bora, Dinthisrang Daimary, kh Amitab, and Debdatta Kandar. Handwritten character recognition from images using cnn-ecoc. *Procedia Computer Science*, 167:2403–2409, 01 2020.

[2] Faisal Farooq, Venu Govindaraju, and Michael Perrone. Pre-processing methods for handwritten arabic documents. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 267–271. IEEE, 2005.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[4] Jaakko Sauvola and Matti Pietikäinen. Adaptive document image binarization. *Pattern recognition*, 33(2):225–236, 2000.

[5] Olarik Surinta, Michiel Holtkamp, Mahir Karaaba, Jean-Paul Oosten, Lambert Schomaker, and Marco Wiering. A* path planning for line segmentation of handwritten documents. volume 2014, 05 2014.

[6] Chi-Feng Wang. The vanishing gradient problem. `https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484`.

Appendix A Additional results

A.1 Line segmentation

The horizontal projection profile generated for the input document (figure 2) is shown in the figure 13 and it's corresponding line regions are shown in the figure 14.

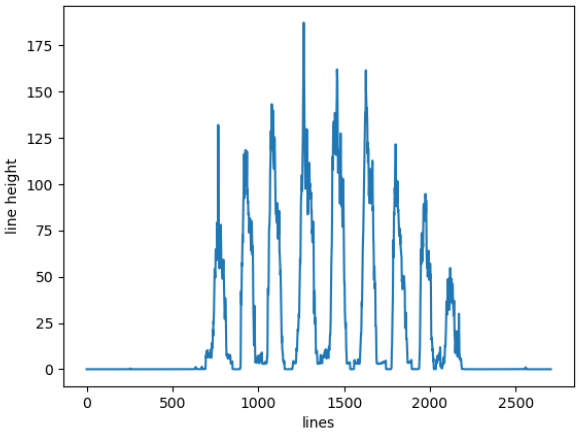


Figure 13: Horizontal projection profile

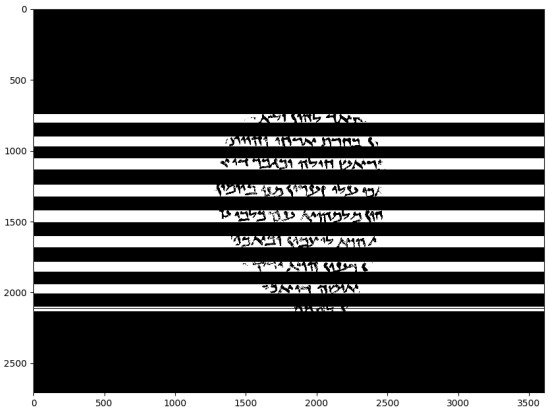


Figure 14: Line regions found as a result of horizontal projection profile

A.2 Character Segmentation

Inspired from Surinta et al., [5], we decided to try a similar approach for segmenting our the characters. For this we first generated a histogram from the pixels. This is shown in Figure 15a. We then used a window in which we noted the peak and a threshold to select histograms which demarcate the character boundary. The result of this is shown in Figure 15b. As it can be seen this method provides a good result. However, it still ends up missing a few characters appearing very close together. This means another step is needed to make finer segmentation. When experimenting with A* we found the results poor. Unlike lines which have mostly empty space except for a few stray strokes, characters usually are connected as a single stroke. Even after factoring in the connected stroke, our results remained the same. As a result, we decided to move onto another method. Since another step was required, we dropped the histogram based method and favoured connected components due to faster computation.

(a) (b)

Figure 15: Histogram based approach for character segmentation

A.3 Style Classification

Figure 16 shows the examples of the high level and low level features extracted from by model.

(a) Alef
Higher Level Features Extracted

(b) Alef
Lower Level Features Extracted

Figure 16: Examples for Style Classification Feature Extraction