A Handwriting Recognition and Classification System for Dead Sea Scrolls

December 28, 2020

Abstract

Ancient scrolls are primitive records that contain important pieces of information related to the history of humans and other significant historical events, due to deterioration by time these scrolls must be decoded to extract the useful information robustly before they are lost entirely. In this paper, we develop an automatic end to end system to recognize and extract the text from the Dead Sea Scrolls document and also classify the document into one of three periods namely Archaic, Hasmonean, Herodian. The main system is divided into four parts line segmentation, character segmentation, character recognition and transcribing, and style classification. The system takes in the image of the document as the input and gives two text files as output, the first output containing the transcribed text from the document and the second output containing the period name.

1 Introduction

Historical documents give us very valuable information from the past. The deterioration of these documents due to the effect of time, improper storage and biological agents there is a need to extract and digitize this information robustly. The degradation of these documents is the major challenge to the system. Our project focuses on building an automatic end to end system to recognize and extract the text from the Dead Sea Scrolls document and also classify the document into one of three periods namely Archaic, Hasmonean, Herodian. The Dead Sea Scrolls [1] (also the Qumran Caves Scrolls) are ancient Jewish religious manuscripts.

The first part of the line segmentation has two subprocesses, the geometric rotation and A* path planning. Farooq et.al [4] uses the slope correction technique to correct lines that are not straight in the Arabic documents, we have implemented a similar slope correction method by computing the horizontal projection profiles on different angles and selecting the best angle with maximum line peaks. Similar to the work of surnita et. al [12] using the novel A* path planning algorithm on the historical handwritten documents the line segmentation is performed.

Initially, for the second part, a similar A* method was experimented on for the character segmentation as well but the idea was dropped favouring an even better approach of based on Connected Filters [3] and Template matching [7].

For the third part of the system, we look at CNN based methods for the feature extraction and classification. Ajay et.al [8] used Alexnet for feature extraction with Support Vector Machine (SVM) for classification of Malayalam characters with an accuracy of 98%. Muhaafidz Md Saufi et.al [10] used CNN to classify Roman Handwritten character recognition with AlexNet and GoogLeNet with 94.47% and 94.23% accuracy respectively. Moreover, they mentioned GoogLeNet produced the best result with a longer time to achieve and Alexnet produced less accurate result but with a faster rate. But these mentioned papers use a very high amount of clean training which is not available to us in our case, Thus, with little training data for Hebrew characters, our goal is to achieve good accuracy.

The final task of style classification is divided into two parts of feature extraction and classification. Similar work from [5] using CNN to extract higher, middle and lower level features and determine the most useful features, [14] using Vgg-16 to visualise the extracted features from different layers and [15] to extract CNN features for classification, even though this paper does not directly relate to the current

task it provides good expertise on CNN and feature extraction process. The architecture we used here specifically is Vgg-16. For the classifier part, we do similar work from [6], this paper shows very good results by stacking multiple classifiers. Further research shows that boosting algorithms works well for low data and unbalanced dataset. Hence the final classifier will be a result of stacking multiple boosting classifiers.

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.

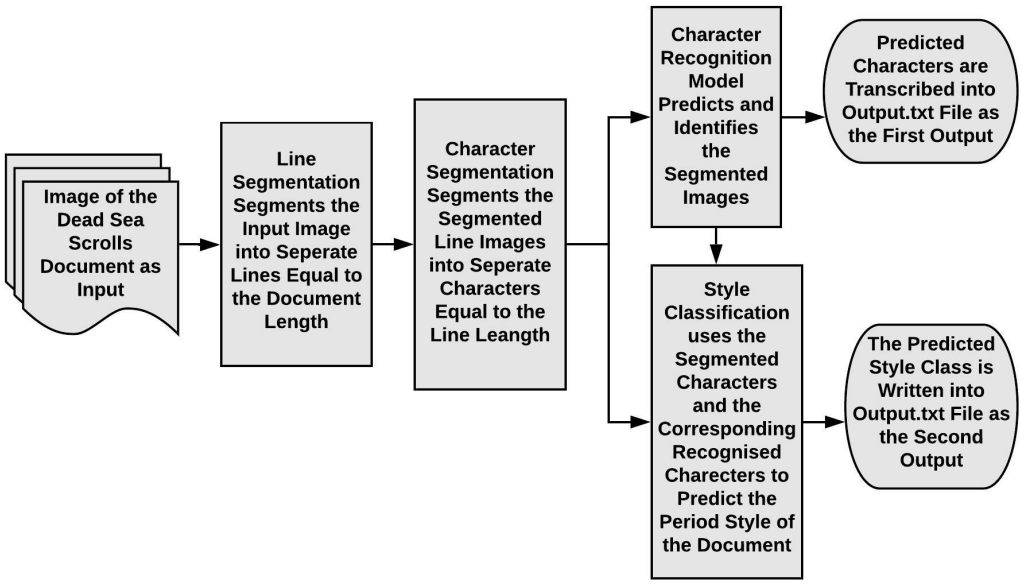

Figure 1: System Architecture

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.

2.2 Line Segmentation

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.

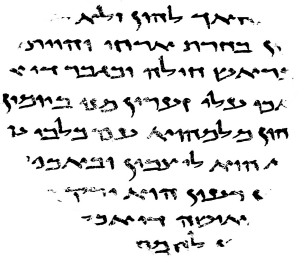

Figure 2: Input document

2.2.1 Rotation

Initially, before segmenting the lines from the document we rotated the document to make all the lines to align in a straight line 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 centre of the document of mass (similar to the slope correction method proposed [4] 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 its 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.

Horizontal projection profiles (HPP) are calculated to find the height of each line in the document, the lines which have 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 its horizontal projection profile and peak hills. Figure 3 shows the output of geometric rotation for the given input document.

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

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 [12]. This is 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 a hyper-parameter, which divides the high peak regions and low peak regions in the horizontal profile. To separate the lines which have 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 size of the window is stored as separate segments, by grouping the blocks found in that particular line. Then we applied an A* path planning algorithm [12] with the start and goal stage as the beginning and end of the lines found in horizontal projection profile clusters.

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

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 its 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 its 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 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 roadblock 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 [11]. 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 kernel 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 Figure 5.

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

Figure 5: Images extracted by Connected Filtering

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 labelled 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.

It can be observed that the 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 can classify this accurately. Thus as a whole, the pipeline remains robust to individual errors.

[1]The implementation of A* path planning and roadblock is 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/`)

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

Figure 6: Images extracted by Template Matching

A point for consideration is that not all extraction is perfect in this stage. Hence, in our code, we do 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 the 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 a powerful tool for extracting features from handwritten characters and classifying using softmax making effective technique for Character Recognition Process [2]. Thus, after Processing the Images, Alexnet Architecture [9] 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 overfitting and vanishing gradient [13].

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 greater than its height. So, to resolve this we set two thresholds. Which are, if the height to width ratio is greater than 0.23 and if the predicted character by the model has 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.

3.2 Character Segmentation

Figure 11a shows the original line which was used as the input for character segmentation. 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.

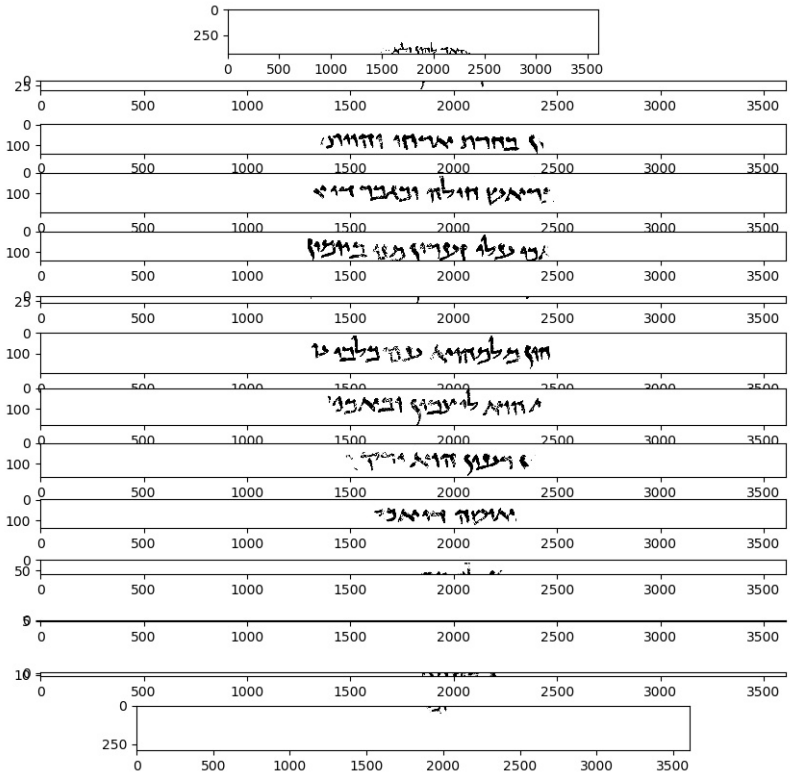

Figure 10: Final line segmented results after the geometric rotation and A* path planning on the input document (figure 2).

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

Figure 11: Comparison of extracted characters from the line

3.4 Style Classification

Table 2 shows the final results of the stacking classifier used for the classification of the document and the document mentioned in the above example has been successfully classified into the *Hasmonean* class.

4 Discussion

4.1 Line Segmentation

From figure 10, the output of the line segmentation, 8 out of 9 lines from the main document is segmented out correctly, the one line not segmented is further fragmented into multiple segments. We believe this is caused by the segmenting threshold value parameter set during the binary map generation. This value is document specific and the current value is achieved by trial and error on testing with different documents. Also, one thing to note in the A* result is that in the first line of the docu-

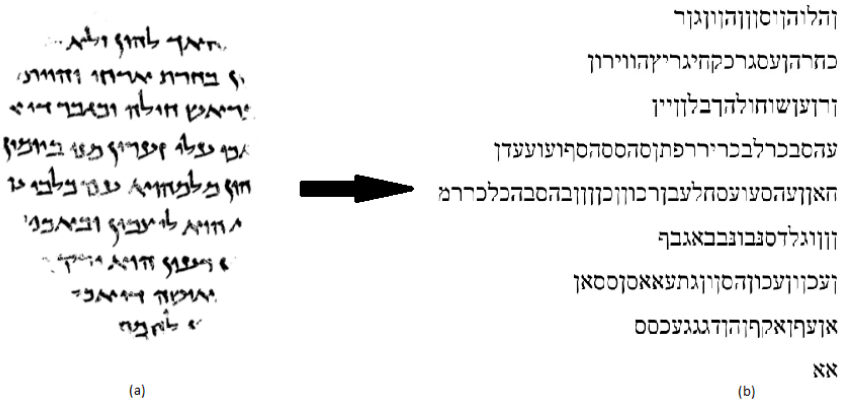

Figure 12: Result of Character Recognition Model on the input Image.

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

ment the algorithm draws two-line boundaries instead of one and also in the final line the characters are fragmented by two separate paths. These errors occur due to the characters improper ink full, and an example for that is in the first line the character ”Nun”. In some of the results during experimentation, two lines are segmented into the single-segmented blocks, this is also due to the segment thresholding parameter. This problem could be solved by using gaussian filters but in our case, it breaks the important features of the character.

To counter these problems before applying the geometric transformations we can attempt to remove the noise for example by using SIFT to find the features associated with characters to filter out the noise, we also need to remove the unwanted block segments as in the final blocks of segmented lines to eliminate the noise in the character segmentation part, and regarding the segmentation problem mentioned earlier, the segmentation threshold parameter can be made as a trainable value to learn from the input document. Also using the statistical approach like bi-variate Gaussian distribution may improve the segmentation accuracy.

4.2 Character Segmentation

From figure 11a the final segmented images from the line are seen, the character segmentation is decent, but this method depends on the quality of the lines segmented from the previous step, the characters within each line and the resolution of the original image. Further, it uses image ratio to discard image, area threshold to retain match and matching threshold to eliminate noise. While there were not many issues on the training data the performance on the test data might be different. In that case, this algorithm might not work too well. Additionally, template matching is also sensitive to scale and the pixels in the search image, that is If the image contains only black pixels, then the character with the most strokes will be matched by the template matching algorithm.

To counter these issues we believe that using neural networks will make the segmentation process more robust and resistant to the changes in resolution and noise in the document. First is to train a neural network to recognize individual characters to segment them. Second, a more traditional method of optical character recognition can be employed which involves directly giving the segmented word image as input to a neural network and have it predict the entire word which has been successful for English character recognition and number plate recognition. Finally, we want to experiment with neural networks in a loop, that is after segmenting the character, we can have a neural network output it's probabilities and if the predicted class has a probability below some threshold, then we can assume that the character image is an aggregate and can recursively split and send to the neural network. We believe that these strategies might help improve the performance of the character segmentation part.

4.3 Character Recognition

From figure 12 the character recognition classifier sometimes predicts extra characters for example like in line 4 and 5. This can because by the character segmentation in the previous step might split one character into two and the classifier might try to predict two characters. Further, if partial characters are segmented then the classifier might predict is the confidence is above the threshold. Moreover, if the character is segmented perfectly but has incomplete features then the character might be predicted with confidence below the threshold.

To counter this problem more training data is needed either in the form of real data or data augmentation. Also, Residual Network (ResNet) can be used to solve for vanishing gradient problem as discussed in section Character Recognition. The use of ResNet over Alexnet-with more depth is more effective because when we try to add more layers to AlexNet the results we received 0.046 as validation accuracy and 3.08 as validation loss. This way we can increase the complexity of the model without increasing the depth to give better predictions.

4.4 Style Classification

Looking at the training dataset for style classification, the overall amount of data is very low. The data averages from a minimum of 94 to a maximum of 150 images per character class, considering the complexity the data is low. Also adding to the previous point the class Hasmonean and Herodian both contain 50 images each per class for each character, except for TSADI-FINAL, which has 44 and 50 images respectively. The problem comes in the Archaic class, which contains fewer data compared to the other two classes averaging from a minimum of 2 to a maximum of only 36, with an exception of Yod which has 50 in all the three classes. This leads to a highly unbalanced dataset. Looking at the results, Ayin, Kaf, Nun-Final, Pe have lower testing accuracies compared to the training accuracies because the archaic class of the characters has only 12, 8, 4, 2 images respectively, and the algorithm was not able to learn the representations for all the cases and characters like Lamed, Taw, Qof, have

lower Training accuracies compared to the testing accuracies because the archaic class of the characters has only 16, 14, and 8 images respectively. Other characters like Samekh, Gimel, Dalet, and He, even though having image data in single-digit numbers, the classes were different enough for the algorithm to learn the representations. Finally, the rest of the results show that the overall learning representations of the classier were decent.

The above-mentioned problems can be solved using data augmentation methods, but the task of handwriting recognition is very sensitive to changes in the data. Even a small change could result in an entirely different meaning. The regular data augmentation methods like rotation, skewing, sheer and zoom did not work well as expected because it results in an incorrect meaning of the character. Other methods like adding noise, for example, gaussian noise, salt and pepper noise, Poisson noise and speckle noise also were tried but did not give the improvement we expected, the raw data without any transformations gave the best results. This raises the need for a more robust data augmentation method for the specific task of the handwriting recognition system as a part of our future work.

Apart from the above-mentioned hurdles, the final style classification part, heavily relies on the previous parts of line segmentation, character segmentation and the character recognition, as explained in the methods section, and any impact in the performance in the previous parts will also impact the final performance of this part.

4.5 Conclusion

In this project, we successfully built an automatic end to end handwriting recognition system for the Dead Sea Scrolls [1], which takes in a document image as an input and successfully segments the lines and characters, and then recognizes and transcribes the text to an output file. Also as an additional step, the style classifier predicts and classifies the document into one of three periods namely Archaic, Hasmonean, Herodian and writes it into another output file. We were able to achieve this with decent results, but with the above-mentioned counter to the problems faced and the suggested corrections and future work all mentioned in the previous section, we can aim to improve upon our current results to achieve a robust system.

References

[1] Isreal Antiques Authority. Dead sea scrolls. `https://www.deadseascrolls.org.il/learn-about-the-scrolls/introduction?locale=en_US`.

[2] 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.

[3] Tim Chin. Using image processing to detect text. `https://medium.com/@theclassytim/using-image-processing-to-detect-text-8be34c677c11`.

[4] 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.

[5] Dario Garcia-Gasulla, Ferran Parés, Armand Vilalta, Jonatan Moreno, Eduard Ayguadé, Jesús Labarta, Ulises Cortés, and Toyotaro Suzumura. On the behavior of convolutional nets for feature extraction, 2017.

[6] Nima Hatami and Reza Ebrahimpour. Combining multiple classifiers: Diversify with boosting and combining by stacking. 7, 01 2007.

[7] Md. Anwar Hossain and Sadia Afrin. Optical character recognition based on template matching. *Global Journal of Computer Science and Technology*, 19:31–35, 05 2019.

[8] Ajay James, Saravanan Chandran, and Manjusha J. Malayalam handwritten character recognition using alexnet based architecture. *International Journal on Electrical Engineering and Informatics*, 6:393–400, 12 2018.

[9] 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.

[10] Muhaafidz Saufi, Mohd Zamanhuri, Norasiah Mohammad, and Zaidah Ibrahim. Deep learning for roman handwritten character recognition. *Indonesian Journal of Electrical Engineering and Computer Science*, 12:455–460, 11 2018.

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

[12] 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.

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

[14] Wei Yu, Kuiyuan Yang, Yalong Bai, Hongxun Yao, and Yong Rui. Visualizing and comparing convolutional neural networks. *CoRR*, abs/1412.6631, 2014.

[15] Musa ÇIBUK, Ümit Budak, Yanhui Guo, M. Ince, and Abdulkadir Sengur. Efficient deep features selections and classification for flower species recognition. *Measurement*, 137, 04 2019.

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.

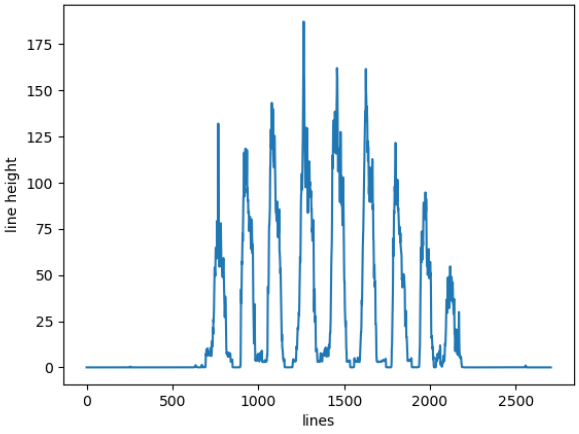

Figure 13: Horizontal projection profile

A.2 Character Segmentation

Inspired by Surinta et al., [12], 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 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

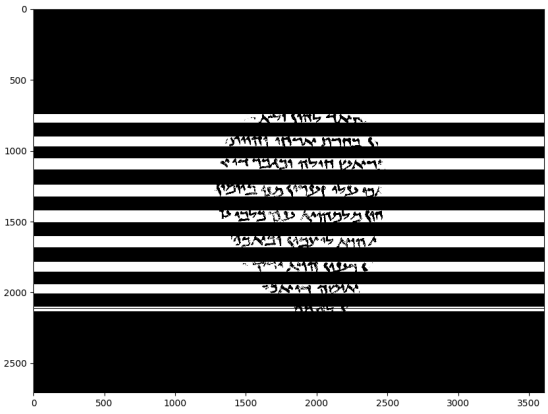

Figure 14: Line regions found as a result of horizontal projection profile

step is needed to make finer segmentation. When experimenting with A* we found the results poor. Unlike lines which have mostly 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