Handwriting Recognition

Avinash Pathapati(S3754715)

June 30, 2019

Abstract
There are lot of things that are unique for each human like fingerprint, DNA and handwriting. As digital processing power is developing and data is easily accessible, lot of researchers have attended to make automatic handwriting recognition.
In this paper, we discuss the techniques we used in detail to build the automatic handwriting recognition pipeline.

1 Introduction

Automatic handwriting recognition is one of the most complex tasks in the field of Artificial Intelligence. Many of the advanced algorithms we have right now are still basic in the sense that they work really well in constrained space but performs poorly when applied to a broader domain. Particularly, many of the current techniques work well for machine printed text but perform poorly in case of handwritten text. The reason is that handwritten text does not have specific format unlike machine printed text. We experimented with many of these techniques on a specific handwritten text "Dead Sea Scrolls" and chose the techniques that gave the best result.

As computers have become the most common way of storing and sharing information, digitization of historical handwritten documents have become increasingly popular. The handwritten historical documents pose many problems like having unclear text, torn pages and so on. So extracting the required information with minimal loss is a highly challenging task.

Dead sea scrolls are ancient Jewish religious manuscripts. More information on dead sea scrolls is given in 2.1. We built a digital pipeline on these scrolls that can perform handwriting recognition. The first step in this pipeline is to extract the region of interest(ROI) which is the parchment from each of the scroll images. Then "Pre-processing" step is performed to extract the text present in the scrolls as a binary image with minimal information loss. The next step is "Line Segmentation" which extracts the lines containing the text from the pre-processed images. Then these lines are sent to the "Character Segmentation" part where the words are first identified and the characters are extracted later from these words.The final step is "Character Recognition" task which takes the characters as input, identifies these characters and reconstructs the text back. The detailed description of these methods are provided in the section below. As we are not provided an additional dataset to test our pipeline, we could only provide the performance of our pipeline on the same dataset.

2 Method

2.1 Dataset [1]

The dataset we used to test our system on, was provided to us confidentially and it is comprised of twenty gray-scale images from the Dead Sea Scrolls which are ancient Jewish religious manuscripts. These images were fused from multiple images of different spectral bands that were made out of each physical fragment and they were given to us rotated into the appropriate position for reading. We can

[1] Section written by Andreas Pentaliotis

see an example in Figure 1. In general, the images have components that are not relevant to the task at hand, and the scrolls themselves are degraded and damaged, some more than others. The quality of the text on the scrolls varies from almost completely faded to fairly clear. The text on the sample image for example is fairly clear.
Moreover, we were given images of handwritten ancient Hebrew characters which we used to pre-train our character classifier on, before including it into our final system (Figure 2). Because five letters of the Hebrew alphabet have two different forms based on their position in a word, a decision was made to include the different forms as a different class in the character images. Thus, we were given 27 different classes of character images instead of 22. This made it easier for our system to recognize the character classes.

Figure 1: Sample image from the Dead Sea Scrolls dataset.

(a) Alef (b) Shin (c) Yod (d) Qof

Figure 2: Sample characters from the handwritten character data.

2.2 Pre-processing

2.2.1 Thresholding method [2]

Some well-known binarization algorithms are Otsu [1] and Sauvola [2]. We used Otsu's method because it performed better for our images with clean background. Sauvola is better for images with more complex background (e.g. for images with glare). Otsu's method performs better when we are thresholding the histogram with multimodal or bimodal distribution.

Otsus's method finds the threshold value based on maximum and minimum of two classes. Also, the processing time of Otsu's method is very fast as it uses only one threshold value.

2.2.2 Relevant Text Extraction [3]

As shown in Figure 1, each image consist of multiple components that are not relevant to our task. This means the first task at hand was to extract the relevant text from the given scrolls. Morphological filtering, connected component filtering and masking techniques have been applied for extracting the central component.

[2] Section written by Andreas Yelasis
[3] Section written by Rohit Malhotra

We start by loading the grayscale image and perform area closing by a large square kernel. The purpose of this step is to merge all the text with the background, so that relevant shapes are filled. It is possible that the relevant component might get connected to the irrelevant components and therefore an area opening using the same kernel is used in order to disconnect them. The best results were observed with a kernel width of 20. Next, we needed to binarize the results in order to perform connected component filtering. Because some images contained pieces of tape, we couldn't set one global threshold to binarize every image. Hence we used Otsu's method [1], which automatically detects the threshold value.

An observation which we made was that in all of the images the central component had the biggest area of all components. Therefore, in order to extract relevant text we calculated the connected components of the binarized image and filtered out the component with the largest area. We used this component as a mask to extract the relevant component from the original image.

2.2.3 Binarization [4]

Until now we have only extracted the relevant portion from the grayscale image. In order to move forward we need to convert this grayscale image to a binarized image, with text coloured black and the background set as white.

First we start by smoothing the image to remove noise. As edges are important parts of a characters, we use a Median filter [3] to smooth the image because it is an edge preserving noise removal technique. Next, we binarize our result using Otsu's method. As the colour of text was always of black shade and non text was always of grey shade, only single threshold binarization was sufficient, hence we used Otsu. After this we performed edge detection on the binarized image and extracted only non edge parts from the previous result. This step helped us increase the gaps between the two characters, which can significantly help in character segmentation. Although it resulted in slight thinning of the characters, the negative effect was insignificant.

Now we have a result where not only characters are black but also some portions of image are black, like holes in scrolls. So we used an eroded version of the mask which we obtained in relevant text extraction and converted all those pixels of our result to white which are black in the mask. We used an eroded version of the mask to remove borders of the scrolls.

2.3 Line Segmentation [5]

We implemented many different techniques for line segmentation. We started with the A^* path planning algorithm [4]. In our case there were many images where lines and characters were broken and we observed many paths that were crossing each other. This resulted in bad performance and large computational costs.

Furthermore, we implemented a Hough transform based technique as described in [5]. We observed many small fractions of lines instead of complete lines and couldn't find a way to merge these smaller lines into a proper segmentation and decided to also drop this method.

As we observed in most of the provided scrolls, lines were straight with almost no skew. The main problem was that they were broken, with many gaps in each line. Keeping this in mind we made our own method for line segmentation which can be categorized as a connected component based bottom up approach. It is a combination of Linear Waterflow method [6], with horizontal projection profiling and it assigns each connected component to a line.

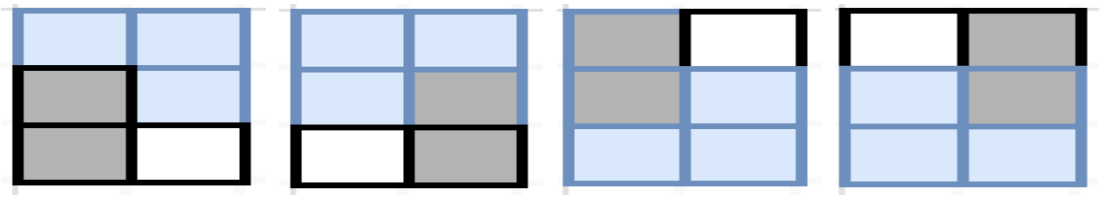
Step 1: We start by calculating all the connected components (CCs) and we represent each CC as an object with the following properties: its centroid, area, height, width, top pixel which is top height and left column pixel and gravity centers of connected component. We define gravity centers of a CC by dividing each CC into blocks of average width of all CCs and height of the current CC. Next, we calculate the position of the gravity centers of each block of CC and store them. We only use those CCs that have an area greater than a certain threshold. We call this parameter the "area of no impact" and we tuned by trial and error to a final value of 100. This filtering is necessary because in our images we found many small CCs that can be classified as noise.

[4] Section written by Rohit Malhotra

[5] Section written by Rohit Malhotra

Step 2: Now we apply the linear waterflow algorithm as described in [6]. The basic idea of the water flow algorithm is to imagine water flowing through the image and characters acting as barricades. When we imagine water flowing from left to right, then barricades protect region on their right from getting wet and when we imagine water flowing from right to left, the barricades protect the region on their left. These unwetted regions are used to extract portions of line segments.

We start by extracting the bounding box(BB) of each CC from the image and in each BB we only keep the main component and remove the portions of other CCs. We define the BB as a padded rectangle enclosing CC. Now we calculate the boundary pixels in each BB. In addition to definition of boundary pixels in [6], we define two more types of boundary pixels, shown in Figure 3b and Figure 3d. As we observed there were far too few pixels that could be categorized as boundary pixels according to original criteria, so we removed 1 pixel (shown as white pixels in Figure 3). Using these pixels as seed pixels we ran the water flow algorithm. Results are shown in Figure 4.



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

Figure 3: We also changed definition of boundary pixels by making pixels in white having no impact.

(a) Binarized text image.

(b) Unwetted areas of waterflow algorithm are shown as white pixels.

Figure 4: An example of execution of waterflow step.

Step3: Now we do horizontal projection profiling of the unwetted region graph obtained from the water flow algorithm. We define a parameter for minimum distance between two peaks to be considered as separate lines as 50. This gives us an approximate idea of the number of lines in the image.

Step 4: In this step we categorize each connected component to a line. As we mentioned in step 1, each connected component is represented by its many gravity centers. So categorizing a CC to a line is reduced to a problem of which line will fit these gravity centers best using a loss function. The mean squared error was used as the loss function.

After this step we need some post processing, due to the skew in the lines it might be possible that a few of the connected components might get associated with the line above or line below. Thus we calculate the average height position of each line and reclassify each component again using these as position of lines. As mentioned in step 1 we don't consider the components with area less than the area of no impact.

2.4 Character Segmentation [6]

The character segmentation task in the pipeline receives the lines extracted by the ”line segmentation” method one at a time as input. The character segmentation task contains two sub-tasks which are explained in detail below.

2.4.1 Word Segmentation

The first step in character segmentation is to identify and extract the words. After exploring different possible methods for word segmentation, the initial approach which we implemented for word segmentation was clustering technique as mentioned in [7]. The paper compares the clustering technique with other techniques on handwritten Arabic text for word segmentation and showed that clustering based techniques performed well compared to the probability based techniques. Among the different clustering techniques, the paper showed that Fuzzy C-means clustering technique yielded the best results. So we started with this technique to find the word boundaries in the extracted Hebrew line images.

To implement this technique, we had to find the valleys initially in these extracted line segments using Vertical Projection Profile(VPP). VPP assigns each column the number of white pixels in that column of the image. Since the valleys found could be a possible gap between the words or characters, we segregated these valleys into two groups namely ”within” word and ”between word” thresholds using Fuzzy C-means clustering. The results we obtained were bad as the technique classified many of the ”within” word gaps as ”between word” gaps for multiple images. One of the lines with the word boundaries found using this technique are presented in the figure 5a.

(a) Incorrect word segmentation

(b) word segmentation

Figure 5: Word Segmentation results

We can clearly see that boundary marked in ”red” shouldn’t be a word boundary but still the technique identifies it as one. The reason could be that clustering techniques need considerably large number of samples to be able to correctly cluster them into groups. As we had only very few samples of the word gaps, the technique tried to group together most of the character gaps with the word gaps. So we then switched to directly using the gaps found using ’VPP’ in the lines to find the word boundaries. If the gap length computed using ’VPP’ is greater than a threshold value of 10 pixels, it corresponds to word boundary. We arrived at this threshold value after experimenting with different values on all the dead sea scrolls. One of the word segmented images with this technique is shown in the figure 5b.

2.4.2 Character Segmentation

The next step is to extract the characters from the word boundaries identified using 2.4.1. We have gone through techniques like sliding window, thinning, VPP but these are simple techniques which may not give the best results in segmenting the characters in our case. The reason is that the images of the handwritten documents are noisy, contains characters which are of varying sizes and stroke thickness. The documents also had touched or overlapped characters at many places. In our chosen technique which can address all of these cases, the first step is to over-segment the image using VPP to find all the possible character segmentation points. The main objective of over-segmentation is to reduce the probability of extracting segments that contain more than one character. After experimenting with multiple values on the different scrolls, we chose a threshold value of 50 percent of the mean for the VPP value. So all the columns having histogram value less than 50 percent of the mean in a line image are identified as segmentation points. A sample line image from the dead sea scrolls with the identified segmentation points is shown in the figure 6a. The white lines in the image indicate the segmentation points.

We can clearly see from the figure that over-segmentation can split the individual character as well. To resolve this, all the segmentation points which are less than 40 pixels apart are merged together.

[6] Section written by Avinash Pathapati

(a) Over segmented image

(b) Segmented image

Figure 6: Character Segmentation

Again, this threshold value was chosen after experimenting with different values and examining which one would produce the most accurate result in our case. The resulting images with the optimal segmentation points are shown in the figure 6b. It can be clearly seen from the image that some of the characters are still connected together. To further segment the connected characters, we implemented the technique in [8]. The technique performs non-linear character segmentation using multi-stage graph search algorithm. The detailed explanation of the algorithm is provided below.

2.4.3 Multi-stage graph search algorithm

In an image containing touching characters, the accumulated intensity of the path along a character boundary will be less than through the actual character stroke. So the problem of finding the character boundary is equivalent to finding the path with the least accumulated intensity. To perform this technique, the image is modelled as a multi-stage graph where each row of the image corresponds to a stage in multistage graph, each pixel is a vertex and intensity of the pixel is the distance between vertex in the current stage and the vertex in the next stage. The shortest path is found using the search algorithm described in [8]. We observed in the dead sea scrolls that the average width of the character is around 5 pixels. So this technique is triggered in our case only if the width of the boundary is greater than 5 pixels because there is a higher chance that it contains multiple characters.

2.5 Character recognition [7]

2.5.1 Classifier Architecture

Our final architecture for character recognition is a convolutional neural network (CNN), which we decided to use because it can automatically extract the relevant features that distinguish the character classes from each other and because it is invariant to any translation of the images (see for example [9, c. 9]. This effectively means that our feature extraction step is combined with the character recognition step. Our network takes as input the character images, which are resized to 64x64 pixels. Then the input is passed to one convolutional layer with 128 filters of size 5x5 which perform a convolutional operation with a stride of 1 on the input to extract feature maps. In this layer we applied padding with zeros around the image before the convolutional operation to avoid any shrinkage in image height and image width. The resulting feature maps are passed through a rectified linear unit activation function which is given by

$$R(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{1}$$

This surrogate activation function helps the network learn because it has a derivative that allows for gradient descent. Then we apply a max pooling layer with a filter of size 2x2 and a stride of 2. The max pooling layer performs a pooling operation that takes the maximum in each filtered area of the feature maps in order to downsample them. The previously mentioned steps are repeated one more time, but the second convolutional layer consists of 64 filters instead of 128, because the feature maps are now downsampled and our network tries to identify higher-level features which should be less than their constituent parts. The rest layers are exactly the same as the layers mentioned above. Consequently, the flattened output is fed to a fully connected layer of 1024 units that is followed by a rectified linear unit activation function. We applied dropout with probability 0.2 to this final layer because we have a large number of parameters and dropout has been shown to help neural networks generalize better

[7] Section written by Andreas Pentaliotis

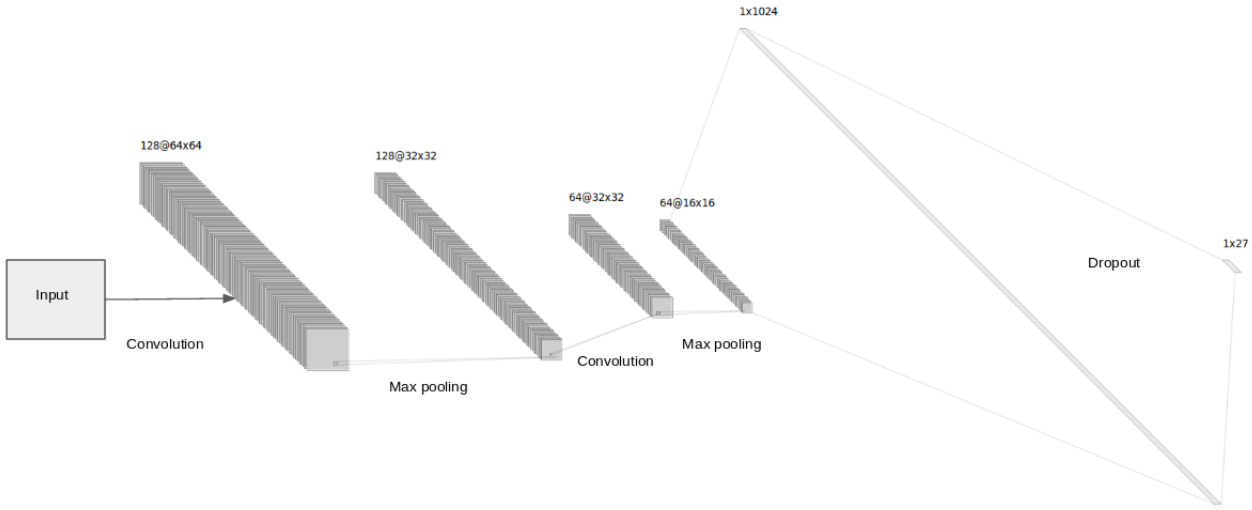


Figure 7: Architecture of our convolutional neural network classifier. The input image is passed through a convolutional layer that produces 128 feature maps of size 64x64. Then a max pooling layer reduces the size of the feature maps to 32x32. We repeat a similar procedure and we end up with 64 feature maps of size 16x16. The information is passed to the output layer through a fully connected layer of 1024 units with dropout. Part of the figure constructed with the software in [10].

on unseen data [9, p. 265]. Finally, the last layer is a fully connected layer with 27 units that use the softmax activation function to output the probability of the input belonging to each of the 27 classes. The softmax activation function is given by

$$\sigma(x)_i = \frac{\exp(x_i)}{\sum_{k=1}^{N} \exp(x_k)} \tag{2}$$

where i is the index of the i-th class and in our case $N = 27$. We use the categorical cross-entropy loss function which is given by

$$L(p) = -\sum_{k=1}^{N} 1_{o,k} \ \log p_{o,k} \tag{3}$$

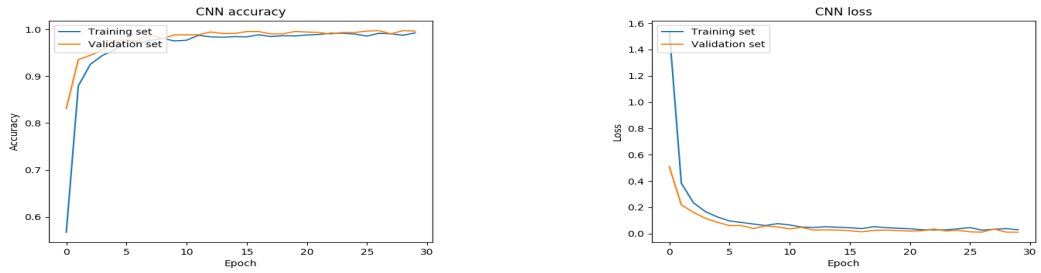
where 1 is an indicator function that takes the value 1 if observation o belongs to class k or 0 otherwise, and p is the probability given by the model that observation o belongs to class k. The sum is taken over all N classes. This formulation makes the categorical cross entropy loss appropriate for problems with multiple classes. For optimization, we use the Adam optimizer which has been shown to be very efficient in many deep learning tasks and very robust with respect to the choice of hyperparameters [9, p. 309]. Our architecture is depicted graphically in Figure 7.

2.5.2 Training

During all the experiments and the final training described in this section, we load the character images into memory, we resize them to 64x64 pixels and we perform binarization on them using the Otsu method [1], to more closely resemble the segmented characters from the Dead Sea Scrolls. We apply data augmentation during training, either with a validation split or without, but also during testing as it has been shown to provide better accuracy rates [9, p.453][11]. The data augmentation techniques, which have also been used in similar tasks [12], involve zooming in and out by a factor of 0.1, shifting the image horizontally or vertically by a factor of 0.1 and rotating the image slightly by 5 degrees either clockwise or counter-clockwise. These parameters were kept the same every time we performed data augmentation.

To determine the optimal hyperparameters of the model, we used trial and error. We first split the character data into training set and test set. The test set consisted of 25% of the original dataset and was kept as a final test for our model after the optimal hyperparameters were determined. Consequently, we

extracted 25% of the training set data and used them for validation purposes in each epoch. Eventually, we found that training the model for 30 epochs with a batch size of 32 is sufficient to achieve an accuracy of 99.6% on the validation set. The metrics we extracted for the training set and validation set during a validation split experiment with the mentioned settings are shown in Figure 8. We can see that the model does not overfit as its validation accuracy and loss are closely following their training equivalents. We suspect that this was due to the fact that we applied data augmentation on the validation set, but given that we are also performing test-time data augmentation this decision was reasonable.
After determining the optimal hyperparameters we used them to re-train our model on the union of the training set and the validation set, and used the test set to determine the final accuracy of 97% on unseen data. The data augmentation on the test set, involved augmenting a given character image twenty times. After that, we took the mean vector of those twenty prediction vectors as the actual probabilities for the character belonging to the 27 different classes. We then used the maximum of those 27 probabilities to determine the label for the given image.
Finally we re-trained our optimal model one time with all the available character data (i.e. the union of the training set, validation set and test set), again using data augmentation, and we included this final version in our system.



(a) Training and validation accuracy

(b) Training and validation loss

Figure 8: Training and validation accuracy / loss of our convolutional neural network classifier during an experiment with validation split. We trained for 30 epochs with a batch size of 32 and we applied data augmentation during both training and validation.

2.5.3 Classification

When our trained classifier is given a segmented character for prediction, we apply the same preprocessing steps that were described in the training section, resizing the character image to 64x64 pixels and binarizing it with the Otsu method. After the preprocessing steps, we apply the same data augmentation procedure that we used during the testing on unseen data and which is described in the training section. The data augmentation parameters are kept the same as during training. We then provide the determined label for the image to consequent parts of our system.

2.6 Transcription [8]

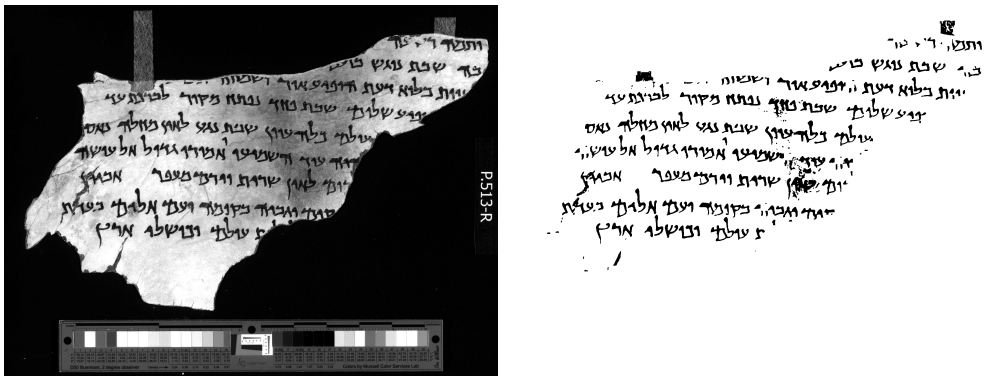
The purpose of the final module in our system, is to translate the predicted labels from the character recognition module to ancient Hebrew character symbols and then write them in a document file. Specifically, for every line in the original image, a new line is generated in the document file, which contains as many characters as our system was able to recognize from the original line. Internally, we map each label to its respective Unicode form and then use that Unicode form to produce the output symbol. We create one document file for every given image, and the output of this module is also the final output of our system.

[8] Section written by Andreas Pentaliotis

3 Results [9]

3.1 Preprocessing

An exemplar result of the preprocessing step can be found in Figure 9. This example was selected because of the pieces of tape that are present on the edges of the scrap of paper in order to illustrate the difficulties encountered during this phase. While the extraction of the relevant part of the image was always successful, the removal of the pieces of tape was problematic. Figure 9 shows that pieces of the tape are still visible. However, in general, preprocessing was done with success. As can also be seen in Figure 9, the characters in the image have been separated from their background without significantly damaging them in any way.



(a) An image from the Dead Sea Scrolls dataset containing tape.

(b) The resulting image after preprocessing

Figure 9: An example of an image and its preprocessed version.

3.2 Line segmentation

Figure 10: Line segmentation illustrated using colours.

A result of the line segmentation module can be found in Figure 10. To illustrate how the document has been separated into lines, each line is marked in a different colour. The horizontal lines indicate the centres of the projection profiles of each line. It is clear that line segmentation of this document has been quite successful. Characters belonging to the same line are marked with equal colours and there are no parts of characters that have been cut off. What should be mentioned however is that the characters marked in red have been classified as a single line while this is clearly not the case. Another issue noticeable in the image is a stain that has been marked as being part of the third line. Would this stain have been removed during preprocessing however, this issue would not have occurred.

[9] Section written by Lennart Faber

3.3 Character Segmentation

The image and the characters extracted from the ”Character Segmentation” method are shown in the figure 16. We can see that our technique was successful in extracting the characters from the word shown in figure 11. However, this method does not produce the best results in all the scenarios. Sometimes, it will split the individual character into two parts whereas in some other cases multiple characters are still connected together. This is shown in the figure 20.

Figure 11
(a) Word containing touching characters

Figure 12
(b) character-1

Figure 13
(c) character-2

Figure 14
(d) character-3

Figure 15
(e)character-4

Figure 16: Segmented characters

Figure 17
(a) Connected characters

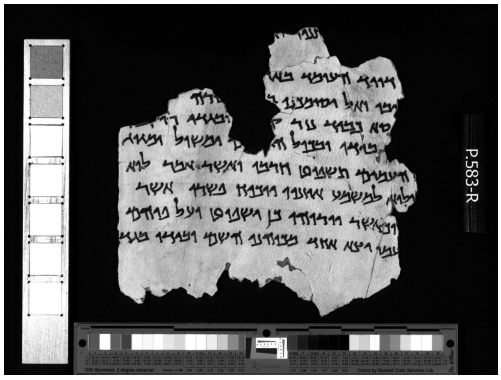
Figure 18
(b) character segmentation region

Figure 19
(c) Possible split points

Figure 20: Incorrectly segmented characters

Our technique took the point(highlighted in green) for segmenting the first two characters in the image instead of the ideal point(highlighted in red) as shown in figure 19c.

3.4 Classification & transcription



(a) An example image of the Dead Sea Scrolls dataset.

רהור העוב בן
תט ואאל הכם ר הן
סא ם תל ר ע וא דר
קסתר וטלן ה ק טשול טוג
העגום תשמט לגסעסנמאמסעשבהאקץסח
אנוגבסמצנמרט ותטוגש ועס פון ם
פוותצא טר מורור ורשם דסתהדן רסע

(b) The transcribed text produced by the system.

Figure 21: An example of an image and its preprocessed version.

An image and its transcribed text produced by the system can be found in Figure 21. The first fact to be noted is that the line segmentation module has not performed as desired for this image: while eight lines can be seen in the raw image, the final transcription shows only seven. Further inspection reveals

that the 6th line in the raw image was not seen as a separate line and one half of it was appended to the line above it and the other half to the line below.

The actual classification of characters however appears successful for at least some letters. While most characters do not seem to have been classified correctly, the system seems to recognise the Hebrew characters 'Alef' and 'Tsadi-medial' correctly in most cases.

4 Discussion

In this paper, we presented an automatic handwriting recognition system which takes the "Dead Sea Scrolls" as input and constructs a "doc" file with the identified text from the scroll. Our pre-processing part in the pipeline is robust as we were able to extract the text from the images with minimal loss. Line segmentation technique works well except in few cases where it identifies two separate lines as one single line. This happened because Horizontal Projection Profile(HPP) in our method identifies it as one line instead of two because the lines are slightly skewed and close to each other. We tried to put extra constraints to identify these lines but this caused the algorithm to split a single line into two in other scenarios. Performing skew correction techniques before applying our line segmentation method could have yielded better results which we haven't tried. In case of character segmentation, we employed a lot of heuristics to get descent results on the "Dead Sea Scrolls" as described in 2.4.2. Our technique can extract the characters which are separated, touching and slanted as well. It is also quite fast as we have used dynamic programming technique to implement the non-linear segmentation task. In some cases, the method still fails to segment the touching characters and also splits the individual character. The character segmentation method is also affected by the output of line segmentation when it sends two separate lines as one single line. Similar results using this method on completely different datasets cannot be expected as we have employed some heuristics suited to the text in Dead Sea Scrolls. Alternative methods like "Watershed" in [13] and "Neural network" in [14] might provide better results for us which we would like to try in the future. "Neural network" technique to find the character boundary is an interesting approach but it requires large number of labelled samples as input which we don't have in this case. The final task "Character Recognition" uses the Convolution Neural Network(CNN) to classify the images. The model produced descent results in identifying the characters. To further improve, we would like to experiment with other techniques like 'LSTM', 'HMM' with Viterbi algorithm in the future for character recognition where the probability of predicting a character in the image also depends on the context around it. This could have improved our model's prediction accuracy.

To conclude, our pipeline produced good results in extracting the text given that the scrolls provided are not so easy to work with. Clearly, there is a possibility to improve our pipeline by trying out the other approaches as mentioned. Given the limited amount of time, these are the best results we have achieved.

References

[1] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[2] J. Sauvola and M. Pietikäinen, "Adaptive document image binarization," *Pattern recognition*, vol. 33, no. 2, pp. 225–236, 2000.

[3] Wikipedia contributors, "Median filter — Wikipedia, the free encyclopedia," 2019, [Online; accessed 29-June-2019]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Median_filter&oldid=889568500

[4] O. Surinta, M. Holtkamp, F. Karabaa, J.-P. Van Oosten, L. Schomaker, and M. Wiering, "A path planning for line segmentation of handwritten documents," in *2014 14th International Conference on Frontiers in Handwriting Recognition*. IEEE, 2014, pp. 175–180.

[5] G. Louloudis, B. Gatos, I. Pratikakis, and C. Halatsis, "Line and word segmentation of handwritten documents," in *1st International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2008, pp. 247–252.

[6] D. Brodić, "Text line segmentation with water flow algorithm based on power function," *Journal of Electrical Engineering*, vol. 66, no. 3, pp. 132–141, 2015.

[7] A. Al-Dmour and F. Fraij, "Segmenting arabic handwritten documents into text lines and words," *International journal of advancements in Computing technology*, vol. 6, no. 3, p. 109, 2014.

[8] S.-W. Lee, D.-J. Lee, and H.-S. Park, "A new methodology for gray-scale character segmentation and recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 18, no. 10, pp. 1045–1050, 1996.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[10] A. LeNail, "Nn-svg: Publication-ready neural network architecture schematics." *The Journal of Open Source Software*, vol. 4, p. 747, 2019.

[11] F. Perez, C. Vasconcelos, S. Avila, and E. Valle, "Data augmentation for skin lesion analysis." *In OR 2.0 Context-Aware Operating Theaters, Computer Assisted Robotic Endoscopy, Clinical Image-Based Procedures, and Skin Image Analysis*, pp. 303–311, 2018.

[12] M. A. Radwan, M. I. Khalil, and H. M. Abbas, "Neural networks pipeline for offline machine printed arabic ocr," *Neural Processing Letters*, vol. 48, no. 2, pp. 769–787, 2018.

[13] P. Mathivanan, B. Ganesamoorthy, and P. Maran, "Watershed algorithm based segmentation for handwritten text identification," *ICTACT Journal on Image and Video processing*, vol. 4, no. 03, pp. 767–772, 2014.

[14] M. Blumenstein, "Cursive character segmentation using neural network techniques," in *Machine Learning in Document Analysis and Recognition*. Springer, 2008, pp. 259–275.