An Automated Handwriting Recognition System for the Dead Sea Scrolls

Group 6
Charlie Albietz, S3058735
Jan-Henner Roberg, S3228851
Péter Varga, S4291093
Panagiotis Ritas, S3152529

***Abstract*— This paper investigates the performance of several handwriting recognition techniques used to be able to automatically extract information from and classify the style of pages of the Dead Sea Scrolls. Characters were obtained from the pages by means of locating the lines using horizontal projection and then extracting the connected components, furthermore, some characters were split using erosion and dilation as well as a sliding window approach using a character recognizer. A selection of different character recognizers were tested including AlexNet, a handcrafted CNN and an SVM. Finally, the style of a page was determined using either hinge feature extraction or an SVN using Histogram of Oriented Gradients (HOG). The results show that using horizontal projections, as well as connected components are good methods for extracting characters from the Dead Sea Scrolls, however, better post-processing steps are required to separate connected characters. Using pre-trained CNNs seems to be a very good method for character recognition as both the CNNs outperformed the SVM. Finally, using an SVM with HOG was shown to perform better style classification than Hinge feature vector extraction, this result may have occurred to some internal errors as better results were reported in other studies using a similar dataset.**

I. Introduction

Handwriting recognition is a field of computer vision and artificial intelligence that is one has proven rather difficult to solve. Unlike recognising isolated characters, which has become a staple in showing the power of neural networks, the task of recognising handwriting is not as straightforward. For one, compared to single character recognition, where the training dataset is often extremely large and rather clean, handwritten texts are written by real people, who all have different handwriting styles and tendencies. Furthermore, many of the texts that would be of interest to apply handwriting recognition to are ancient texts that have reduced in quality over the decades. It thus becomes clear that this task is not trivial and why this field of research has the large history it possesses.

Usually, when attempting such a task, two components must work together to be able to understand the handwritten words. The model must be able to recognize words or characters, as well as extract the desired elements from an image containing multiple lines of words. There are an array of methods that can be used for such tasks. In the early days of handwriting recognition, the main method for recognising characters was using Optical Character Recognition systems, in which the visual features of typed characters are measured and classified (Garris & Dimmick, 1996). Since then, many technological advancements have paved the way for approaches that allow for the characters that are being recognised to be freely written and of lesser quality. Nowadays, one of the most common strategies for recognizing images of characters and digits is to use neural networks. More specifically, Convolutional Neural Networks (CNNs) are specialized to recognize and classify images of whatever the network has been trained on (Albawi, Mohammed, & Al-Zawi, 2017; Krizhevsky, Sutskever, & Hinton, 2012). As previously mentioned, however, when working with handwritten characters of texts, it is not always the case that a training dataset exists for the type of characters that one wishes to recognize. If the training dataset is small, there are a few methods that can be used to overcome this issue. For one, augmenting the data that one has can lead to a dataset that is large enough for the model to perform well (Ding, Chen, Liu, & Huang, 2016). A further method is to apply transfer learning where the knowledge obtained from training a model on a similar dataset can be applied to learn the new, limited dataset (Tang, Peng, Xu, Wang, & Furuhata, 2016). Thus, solving the issue of being able to recognise characters is highly possible with today's technology.

When it comes to identifying the locations of characters of texts, there are again multiple steps involved. First, the model must be able to recognize where the lines are in the image. One method for doing so is to use a Hough Transform. This method can find the angles and directions of lines that connect elements in an image that are inline (Duda & Hart, 1972; Likforman-Sulem, Hanimyan, & Faure, 1995). A simpler yet powerful method is to take the horizontal projections of an image and locate at which y-locations a high number of elements are found (Dos Santos, Clemente, Ren, & Calvalcanti, 2009; Marti & Bunke, 2001; ?, ?). After the lines have been located and appropriately segmented, the model must now work on segmenting the individual words and characters.

Whereas splitting up words can be as simple as calculating the average distance between the elements in the line (Seni & Cohen, 1994), more involved methods such as using neural networks that incorporate wordhood information as well as n-grams (Tian, Song, Xia, Zhang, & Wang, 2020) have been

employed. Depending on which recognizer one wishes to use, further image processing may be necessary or not. In the case that one wishes to recognize characters, methods such as obtaining characters using connected components algorithms, or splitting up connected characters using drop-falling algorithms or neural networks have been shown to be useful, albeit susceptible to artefacts (Casey & Lecolinet, 1996; Sambyal & Abrol, 2016; Cao, Huang, & Wang, 2010; Qaroush, Awad, Modallal, & Ziq, 2020). On the other hand, some papers have worked with whole word recognition. This saves the need for such methods that tend to be erroneous. Malakar et al. (2020) for instance trained a CNN using the lottery ticket regularization technique, where the model methodically layers of the CNN, speeding up the model and reducing overfitting. Although some of these word-based recognizers outperform the character-based counterparts, they are only able to do so if the vocabulary that is to be predicted is limited in size (Kim & Govindaraju, 1997), on the other hand, if the set of predictable words is only limited by the number of words in the dictionary, there would be too many output units for the method to be viable.

In this paper, a selection of the above-mentioned methods will be used on the Dead Sea Scrolls. The Dead Sea Scrolls are among the oldest and most historically significant pieces of scripture that were found in fragments all around the Dead Sea. They consist of mainly Hebrew texts were written on parchment and data back to 80BC (Atwill & Braunheim, 2004). Based on the content of the texts, as well as the styles of the characters, the texts can also be attributed to different periods, such as the Archaic (650 BCE to 480 BCE), Hasmonean (140 BCE to 37 BCE) and Herodian (37 BCE to 73 CE) period. For this reason, this paper will not only attempt to segment and recognize the words in these texts but also perform methods for style classification to be able to sort the pages into the aforementioned periods.

To do so, we intend to make use of the AlexNet CNN to recognize characters (Krizhevsky et al., 2012) as well as a handcrafted CNN and an SVM and see which performs the best. We thus require segmentation steps to obtain characters rather than just words. A horizontal projection will be used along with the A algorithm for line segmentation (Surinta et al., 2014). As a majority of the pages of the Dead Sea Scrolls contain characters that are not connected to other characters, the connected component method will be used. Furthermore, the style of a scroll will be classified by obtaining the hinge features of the individual characters using an SVM (Bulacu & Schomaker, 2007; Boser, Guyon, & Vapnik, 1992).

II. Methods

The pipeline was distributed into and developed in three separate segments: The first one is the text segmentation task, in which first sentences, and then their respective characters are segmented. The second task is Character Recognition, in which each character is classified. Lastly, Style Classification is done based on the labeled characters to provide the dominant style period in which the input scroll was written: Archaic, Hasmonean, or Herodian.

A. Datasets

1) Text Segmentation: To construct the components needed to perform text segmentation, 22 images were used as a reference. These images were originally colorized pictures of pages from the Dead Sea Scrolls that were then turned into grayscale images and subsequently binarized. Two of these images were used to test the performance of the text segmentation so to avoid overfitting the methods on specific images.

2) Character Recognition: For model pre-training on the character recognition task, we used a benchmark dataset, the Handwritten Hebrew Dataset (HHD) developed by Rabaev, Barakat, Churkin, and El-Sana (2020). It was used for pre-training on the CNNs used in this report. Subsequently, the models were fine-tuned on the actual dataset. The HHD dataset consists of 5,054 characters in total, separated into 28 different classes. We removed one class from our characters to ensure consistency in the number of classes. Afterwards, we used elastic distortions (described below) on every character. Finally after preprocessing, elastic morphing and removal of one class, the pretraining dataset consisted of 9,560 characters.

The actual dataset used for training and testing the models on the character recognition task is the brill characters provided by the Dead Sea Scrolls Electronic Library (Tov, 2016). The brill collection dataset consists of 27 classes with each class being a different character. Overall, the brill dataset contains 5522 images of different characters, these were first split into a training and testing set. The split was performed using an 80% to 20% ratio and was done at the character level, ensuring a proportional amount of each character in the training and testing set. For an example overview of different data classes for both datasets in the task, see Figure 2. The dataset has several challenges, there are a lot of artifacts in the images and the classes are unbalanced. To combat these issues preprocessing and elastic-morphing was applied, which is described in more detail below.

3) Style Classification: For individual character style classification, we used a training dataset split into three labels: Archaic, Hasmonean and Herodian. Each dataset has 27 labels, according to each character label. The images were extracted from different manuscripts from the dead sea scrolls. We performed data augmentation on the dataset to balance the ratio between classes, as data on some labels were scarce with sometimes down to only two instances per label inside of a style period. Data morphing was also used to balance the number of characters of the three periods. Figure 1 gives an example of characters from different style periods.

4) Preprocessing: For the preprocessing of all characters, we applied multiple steps to ensure suitability for the subsequent character recognition task. The following preprocessing steps were applied on all datasets to ensure consistency.

First, we applied image-binarization to all characters. Then a bounding-box technique for segmenting and cropping the

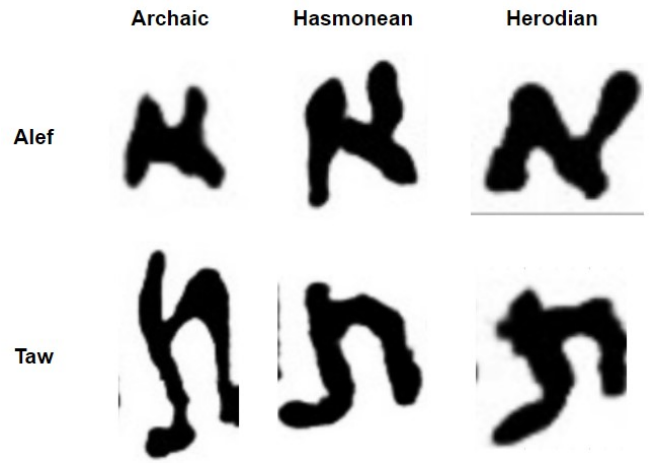


Fig. 1: Example of images from the training dataset. Here, the characters Alef and Taw are portrayed across the three different periods to be classified. Textural information extracted from such images is key to dominant style classification.

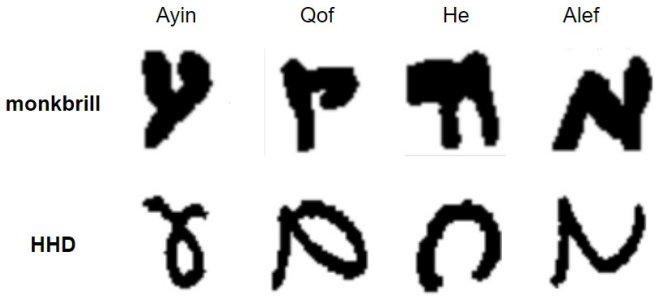


Fig. 2: Comparison of datasets used in the character recognition tasks. top: example classes of the Monkbrill dataset used for training. Bottom: same classes in modern cursive Hebrew contained in the HHD dataset.

characters from overlaps that occurred in the pictures was applied. We did this by first finding all contours in the image. Then, bounding boxes were formed around each image and subsequently the image was cropped in the coordinates with the biggest bounding box, which is the character. For the next step, we resized the images to 40x40.

To completely ensure the removal of unneeded noise, all contours other than the characters were again found on the new image. These contours were removed from the image by masking them on the character image. In the case of grayscale datasets, morphological operations, specifically closing was performed to remove the remaining image noise. Figure 3 shows an example of a preprocessed image along with the preprocessing steps.

a) Character morphing: As stated previously the classes in the character recognition dataset is unbalanced. There are several character sets with less than 40 different images and also some with 300 different images. To balance the data, a data augmentation tool from Bulacu, Brink, Zant, and Schomaker (2009) was used. This tool applies

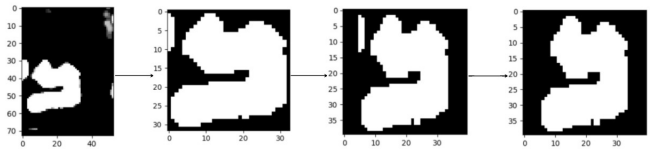


Fig. 3: Example of the preprocessing steps taken for the Hebrew character "bet", as seen from left to right. First, the character is cropped, resized and padded, and lastly, any "noise" from overlapping characters is removed.

random elastic rubber-sheet transformations and this way creates a new image based on a character image in the dataset. Two parameters that need to be set for the system to perform character morphing are the amplitude and standard deviation of the random elastic rubber-sheet transformations. Amplitude was always sampled from a range of [1.5, 2.5] and sigma from a range of [5.5, 8.5]. For an example of this morphing see Figure 4, for a more detailed description please see (Bulacu et al., 2009). To balance out the classes the image morphing tool was used to morph the characters so that every class contained at least half as many images as the class with the most images. For the training dataset, this meant that after this step every class contained at least 150 instances.

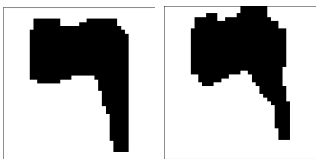


Fig. 4: Left original character; Right character after applying elastic distortions. For more information please see the text

Since the brill dataset contains many unclear characters that can therefore be hard to classify for a model, clean font characters were also included in the training set. Using the Habbakuk font, provided in the project instructions, for each character one clear image was obtained. Since only adding one clear character per class would not be enough, the image morphing tool was used to morph each character 150 times (see Figure 5).

5) Final datasets: The sizes of the different datasets after preprocessing and image morphing can be seen in Table I. The training character recognition dataset is not class imbalanced anymore, Alef has the most instances with 391

Fig. 5: Example of a clear font character and two morphed images based on it. Left: original Alef character. Middle and right: new characters after applying elastic morphing.

Dataset	Size
Pretraining Character Recognition	9.560
Training Character Recognition	9.514
Testing Character Recognition	1.371
Testing Character Recognition no morph	1.112
Training Style Classification	1.884
Testing Style Classification morph	347
Testing Style Classification	293

TABLE I

and Resh contains the least instances with 275. There are two different test sets for Character recognition: One that contains the image morphing to balance the classes and one that only contains characters from the original brill dataset.

The datasets used for Style classification are also not class imbalanced, there are is a similar number of character for each period.

B. Text Segmentation

Text segmentation in our project consisted of three consecutive segmentation procedures, namely line segmentation, word segmentation, and character segmentation. In the following sections, these procedures are detailed.

a) Pre-processing: The documents are all read as grayscale images, then subsequently binarized and inverted, where values below 127 are converted to 1 (white), and values above 127 are converted to 0 (black) so that the pixels of interest are non-zero. As artifacts are rather common in these documents, a single iteration of dilation with a kernel of 2x2 was used on the entirety of any input image. The images within the dataset are frequently tilted which makes line segmentation increasingly difficult, hence a rotation of all input images into a standard position is necessary. Using Hough-Transform for detecting lines (Duda & Hart, 1972; Likforman-Sulem et al., 1995) can consequently be used to find the amount of rotation needed to be performed on the image in the opposite direction. As the closing step, all images were either resized or padded to the shape of 2700x3600 which facilitated the application of empirical, absolute threshold values for segmentation algorithms, compared to the less stable biased and unbiased estimators that are highly influenced by hardly detectable outliers, due to the unknown distribution type of our data.

1) Line Segmentation: In order to have a great degree of information of the lines to be detected from the text regions, the application of horizontal projection profiles is considered a computationally efficient and intuitive solution (Mehul, Ankita, Namrata, Rahul, & Sheth, 2014; Likforman-Sulem, Zahour, & Taconet, 2007). The horizontal projection of a single row of an image of size MxN is defined as

$$P(y) = \sum_{x=1}^{M} I(x, y), \tag{1}$$

where $I(x, y) \in \{0, 1\}$ is a single pixel's value in the input image at coordinate (x, y). The minimas of this horizontal projection histogram are used to mark the beginning/ending of a candidate line. However, as these histograms contain a great number of local peaks (maximas) and minimas being too close to each other, with the inherent property of having equal length rows, the entirety of the histograms were slightly smoothed with a third-degree polynomial and a window size of seven of the Savitzky-Golay filter. Afterwards, the average value of all remaining peaks was set as the threshold for signalling the presence of the minimas of such peaks. An example smoothed histogram with its peak threshold being illustrated with a red line can be seen in Figure 6.

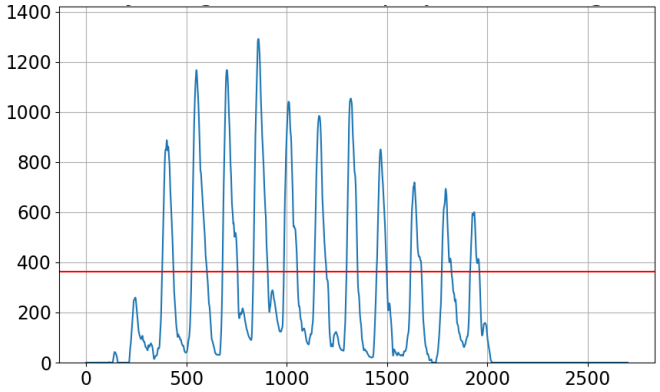


Fig. 6: Horizontal projection histogram of a binarized and inverted image.

Due to the nature of handwriting itself and the abundance of character artifacts on the images, admitting every filtered maximum still resulted in several undesired lines being detected. Therefore, an iterative merging algorithm had to be devised to ensure that our list of lines is placed within at least a standard distance between one another. This algorithm was used to erase every line whose distance was too small to the previous line, going from top to bottom, based on the average line distance across the image.

During the obtainment of the topmost and lowest lines of the image, solely detecting and using minimas and maximas was insufficient as a fraction of a character or any noise could easily produce false extremes. Therefore, at the top and bottom of the image 1-1 artificial lines were added.

Finally, the acquired set of lines were used in pairs of two from top to bottom, where the centre of the distance of such a pair of lines denoted the initialization point of the path-planning algorithm we used as the last step of the segmentation. The A* path-planning algorithm has already been successfully used in the work of (Surinta et al., 2014) for handwritten documents. Having employed the same base algorithm with a less sophisticated solution to the passing of potential obstacles, the majority of lines were sufficiently segmented, as illustrated in Figure 7.

Our solution to the agent's passing of obstacles during path-planning consists of two steps. Firstly, the starting locations of the obstacles were extracted by executing the path-planning algorithm in windows of 40 pixels (since all the images were resized to the same shape, 40 was found to be an ideal choice even for long obstacles), from left to right. Secondly, if there were any obstacles found, a horizontal cut

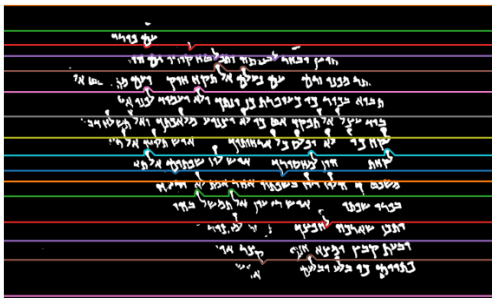


Fig. 7: Segmented lines using horizontal projection profiles and A* path-planning algorithm

of length 40 pixels was performed to ensure free passage to the agent. The algorithm is further described in Algorithm 1. After the algorithm had cleared the paths of obstacles the actual A algorithm was run through the line sections. After a successful traversal of these sections, the trajectory of the A algorithm was used to split the lines of the scroll. From here, the images of each line could be extracted and used in later steps.

```
initialize agent at the beginning of current obstacle;
while True do
    advance agent one step from left to right;
    if path in window is already free then
        break;
    else if a white pixel has just been passed then
        look ahead 40 pixels;
        if next 40 pixels don't contain any white
         pixels then
            perform horizontal cut;
            break;
end
```

Algorithm 1: Algorithm to create doorways for agent in path-planning algorithm

2) Word Segmentation: After the lines had been recognized and extracted, it was necessary to split up the lines into individual words. The method that was used in this project was adapted from Muthukrishnan (2019). The process of extracting clean words from the obtained lines can be split up into three sections

a) Section Trimming: Once the sections containing the individual lines of text have been extracted, they include large borders of whitespace on either side as can be seen in Figure 7. As the whitespace between the words was used in the following steps to divide the lines appropriately, this large amount of whitespace was trimmed away to avoid average whitespace length inflation (see Figure 8, top).

This was done by obtaining a vertical projection of the section and, starting at the far sides moving inwards, checking the current column of the projection if it contains a value larger than 0 (which would indicate that it is not blank). If the value is larger than 0, this location is stored and if the following n column projections continue to contain values, then this location is used to mark the beginning or end of the section. Here, n is used as a threshold value, indicating

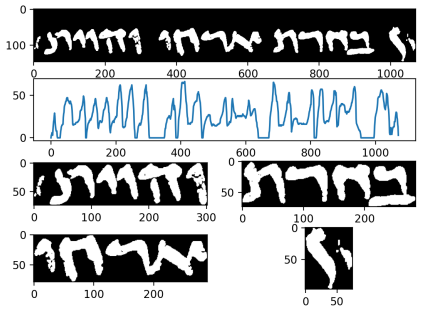


Fig. 8: Top: trimmed line segment. Middle: vertical projection of line segment. Bottom: resulting trimmed and segmented words

the minimum width that an element has to be for it to not be classed as an artifact.

b) Line Division: After the section is trimmed, the vertical projection is used to locate the areas that have whitespace. From here, the lengths of each whitespace section are collected and the average whitespace length is calculated. Word dividers are then placed throughout the section whenever the length of a whitespace section is large than the average whitespace length. Although this method was simple, it was able to reliably identify the words that would have been identified through a visual inspection of a line (see Figure 8).

c) Word Trimming: Once the words were obtained using the previous methods, they were individually trimmed using the method described in a. This time the trimming occurred on both the horizontal axis as well as the vertical axis resulting in clearly defined images of words (see Figure 8, bottom)

3) Character Segmentation: This step of the segmentation pipeline is the most crucial with regards to the final task, style classification. First of all, the acquirement of all the distinct groups of non-zero pixels (components) and their corresponding bounding boxes were carried out with the Connected Components algorithm which is suitable for the identification and extraction of such components, both for printed and handwritten documents (Casey & Lecolinet, 1996; Sambyal & Abrol, 2016). However, just using the connected components algorithm on a word and obtaining clearly defined characters is not a likely outcome. Luckily, in this dataset, the characters are usually not connected, however, due to the nature of the scrolls several issues had to be solved.

a) Dilation: Presumably due to erosion of the scrolls, some characters are in bad condition, made up of a series of disconnected marks. For these cases, connected components would capture the small elements that make up a character instead of the character itself (see Figure 9). To be able to obtain the bounding box of the entire character, the word was dilated. This technique, mentioned by Kumar, Bhatia, and Banger (2013) adds pixels to the sides of elements, allowing

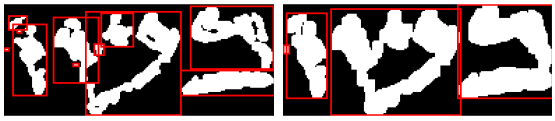


Fig. 9: Left: Connected components of word before dilation. Right: Connected components of words after dilation

Fig. 10: Left: character before artifact removal. Right: character after artifact removal

for neighbors that are within a certain proximity to reconnect after a number of iterations. In this paper, dilation was used until the number of components left in the word image was below a certain threshold. This allowed for characters in bad condition to be recognised successfully (see Figure 9).

b) Component connection: In some cases, characters consist of multiple elements that are detached from one another far enough away so that the dilation step is not effective. These elements are often smaller and within the left and right boundaries of the larger component. To avoid splitting up components unnecessarily, the boundary locations of the components are checked, whether one component is contained within the left and right boundary of another larger component. In such cases, the components are combined.

Often, the bounding boxes would include artifacts from the adjacent characters which could decrease the accuracy of the future character recognition. To remove such artifacts, the bounding box around the character in question was always set with a buffer of at least one pixel so that any elements that touch the bounding box and have an area below a certain threshold are identified as unwanted artifacts. These unwanted elements are then removed (Figure 10).

c) Connected Character Separation: As previously mentioned, the majority of characters in the scrolls dataset are separated by default. There are, however, instances where one character touches another. These characters were identified by obtaining the average character width of each character in the scroll and then identifying a connected character by its width exceeded the average. To separate these connected characters two methods were implemented.

Erosion and Dilation:
Characters that touch each other slightly can be separated using erosion as mentioned by (Kumar et al., 2013). To do so, a certain amount of pixels are removed from the sides of an element over a number of iterations. In some cases, this erosion leads to the separation of connected characters (Figure 11 b). If this occurred, the separated characters were located using connected components and placing a bounding box around them. To counteract the erosion, the characters were dilated by the same amount as they were eroded (Figure 11 c, d).

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

Fig. 11: a: connected characters. b: eroded character are no longer connected, c,d: dilated character after splitting

Fig. 12: Resulting split characters after sliding window segmentation

Sliding Window:
To identify the remaining connected characters, a sliding window approach was adapted from Wai, Zin, Yokota, and Mya (2019). In their paper, the researchers were able to achieve very good results by training a CNN and passing on snapshots of the connected character image by image and accepting the slide and the label when the CNNs prediction displayed a large enough confidence value. In our case, however, the dataset was much more varied in size and quality. Also, snapshots that were taken in between two characters sometimes returned high confidence values once this method was applied. To counteract this some heuristics were made.

First of all, the predicted number of characters contained in the connected character image was taken into account and the window size that would be used to pass onto the trained CNN was the width of the average character in the scroll (see Section II-C for more information on the CNN). If only two characters were predicted, the first and last slide was assumed to be the image containing the character, as, after visual inspection, these predictions most consistently returned accurate results at later stages. In the case that more characters were predicted to be contained, the slide that returned the highest prediction confidence out of a collection of slides was chosen. The set of slides were taken by allowing overlap of the window into 25% into both the previously identified characters boundaries and the predicted location of the subsequent character. The predicted character locations were obtained using the average character widths. Although this method was not always able to obtain accurate results, its performance was acceptable and was thus used in the character recognition pipeline (Figure 12).

C. Character Recognition

For character recognition, three different models were tested. Two of them were Convolutional Neural Networks (CNN); one designed for this specific task and AlexNet

(Krizhevsky et al., 2012) a well-known architecture pre-trained on the ImageNet dataset (Deng et al., 2009).

1) CNN:

a) Handcrafted: Based on previous CNN architectures for handwriting recognition, a CNN was designed to perform the character recognition task on the 40x40 images. For a detailed description of a CNN please see (Albawi et al., 2017). The architecture designed for this task was inspired by the famous LeNet5 architecture (Rawat & Wang, 2017).

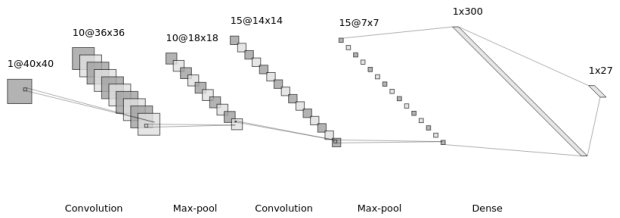


Fig. 13: Illustration of the handcrafted CNN used for character recognition

The architecture created for this task consists of two convolution, two max-pooling and two fully-connected layers (see Figure 13). Cross-entropy loss and Adam optimization were used to train the model. This model was first pretrained on the pretraining dataset with the following hyperparameters; $learning\_rate = 0.001$, $batch\_size = 15$, $L2\ weight\_decay = 0.0001$, for 5 epochs. Afterwards, learning rate and L2-weight decay were optimized on the training set, using 5-fold cross-validation. During this optimization the hyperparameters were: $learning\_rate = 0.002$, $L2\ weight\_decay = 0$, $batch\_size = 15$, for 10 epochs.

After finding the optimal values for learning-rate and L2-weight decay, the model was trained on the entire training set for 10 epochs.

b) AlexNet: An AlexNet was also used in the character recognition task. We used a pretrained AlexNet, trained on the ImageNet dataset (Deng et al., 2009). The last layer of the network was pre-trained on the HHD dataset, then trained on the brill dataset, while the other layers of the model were kept frozen. The brill dataset was resized to 224x224 before being fed into the architecture. Similarly to the handcrafted CNN, logarithmic Cross-entropy was used as the loss function, along with Adam optimizer. The model was trained on 10 epochs for the HDD dataset and fine tuned for 10 epochs on the training dataset. Hyperparameter tuning was done by using 5-fold cross-validation. Figure 14 shows an overview of the AlexNet architecture.

2) SVM: We used Support Vector Machines (SVM), first proposed by Boser et al. (1992). SVMs are known for their state-of-the-art accuracies in character recognition tasks such as the MNIST dataset (Khan, 2017).

SVMs work very well when combined with a feature extraction method, followed by dimensionality reduction. Specifically, Histogram of Oriented Gradients (HOG) has been a widely used technique, followed by Principal Component Analysis (PCA), which help SVMs achieve state-of-the-art accuracies in character recognition tasks for datasets

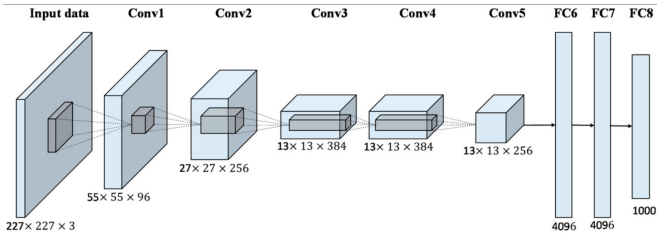


Fig. 14: Illustration of the AlexNet architecture used for character recognition. Image taken from `//www.mdpi.com/2072-4292/9/8/848`

like MNIST (Sarowar, Razzak, & Fuad, 2019). We trained the SVM classifier on 40x40 size characters from the brill dataset.

a) HOG: HOG involves extracting the gradient orientation of pixels in an image and storing them into histograms, done by dividing the image into small overlapping cells. These cells are part of bigger regions called blocks in which the features are normalized and the resulting data are stored in histograms as features (Dalal & Triggs, 2005). The parameters were kept as recommended by (Dalal & Triggs, 2005), with the number of gradient orientations set to 8, cell size set to 8x8, and block size set to 1x1. Each feature vector per character, therefore, consisted of 288 elements.

b) PCA: As mentioned above, PCA was then applied: the first 200 principal components were kept, which accounted for 96.3% of variance of the model. The resulting 200 principal components were then fed into the SVM for training.

D. Style Classification

The third and last task of the project is style classification. Style classification takes the labeled characters from the character recognition task and outputs the "dominant style" of the image. These three different styles come in three different periods, archaic, Hasmonean and Herodian. They are distinct in the sense that the handwriting style of each period is slightly different.

For this part of the project, we set up and tried two different models: we first developed a classifier that takes the extracted hinge features between a dataset/character and compares them with an appropriate metric to the style of the characters in an image. We also set up an SVM which was used to classify characters on one of the three styles in an image. A majority vote/top-10 nearest neighbors vote then concluded the dominant style of the image.

1) Hinge feature extraction: The first method used for style classification is hinge feature extraction, this is based on the idea that, due to behavioral biometrics, the slant of individual writers will be different (Bulacu & Schomaker, 2007). The feature vector is based on the contours of the character. To extract the contours the image was first blurred with Gaussian kernel and then canny edge detection (Canny, 1986) was used. The next step towards the hinge feature vector is to sort the contour coordinates, this was done by first selecting a random coordinate on the contour as a

starting point. Afterwards, the closest contour coordinate is found using Manhattan distance, if there are two coordinates at the same distance, the closest coordinate in the y-direction is chosen. In the special case that there are two coordinates at distance two, one being diagonal (1,1) and one being straight (0,2) to the current coordinate, the coordinate being diagonal is chosen. The closest coordinate found is always removed from the list of contour coordinates to ensure that we do not go back. Since characters contours can be incomplete, the sorted contours are only extracted for connected contours with no gap larger than two. To verify that this worked the sorted coordinates were plotted sequentially on some challenging character images. The extracted hinge contour fragments were also tested on these images.

After contour coordinate sorting there is one or more lists of sorted contour coordinates. To calculate the feature vector a hinge is slid over the sorted contours. This hinge is made out of two contour fragments (red and green line in Figure 15) of length k with a common origin (see Figure 15). In our implementation, $k = 4$ is used, since our images are slightly smaller than the images used in Bulacu and Schomaker (2007). The features extracted by this hinge are the co-occurrences of the angles, ϕ_1 and ϕ_2, that the contour fragments make with the horizontal axis (blue line in Figure 15).

Fig. 15: A "hinge" being placed on a character with the two corresponding angles

The angles are calculated using $arctan2(delta_y, delta_x)$ and then rescaled so that the angles are in the range [0,2π]. Afterwards, the angles are put into a histogram with 24 bins, spanning the range [0,2π]. For the final histogram only non-redundant angles are used, so $\phi_2 => \phi_1$. Now we have a 300-dimensional vector containing the co-occurrences of ϕ_1 and ϕ_2 found by sliding a hinge over the contour of a character. This vector is then transformed into a pdf, which is the final hinge feature vector for a given character.

2) Support Vector Machines: As mentioned in Section II-D, the second one of the two models used in character style classification was an SVM. For our SVM architecture, the kernel was set to radial basis function. We fit one SVM classifier per pair, meaning that three binary classification SVMs were fit on the training data, one for each style period.

a) Histogram of Oriented gradients: We again used HOG as feature extraction method as HOG has been used in writer verification and has achieved impressive results in discriminating between writers, both in Arabic with an accuracy of up to 86 % (Hannad, Siddiqi, El Merabet, & El Youssfi El Kettani, 2016) for writer verification, when combined with SVMs (Yilmaz, Yanikoglu, Tirkaz, & Kholmatov, 2011). As with the hinge textural extraction method mentioned above, orientations in the image can capture varying information of the "slant" of a character which varies from one time period to the other. Cell, block size and number of orientations are kept as recommended by Dalal and Triggs (2005).

3) Model feature clustering: For a preliminary analysis, we plotted the feature space of extracted features. Figure 16 shows plotted feature space of HOG features calculated across the three-time periods. The decision functions created by the SVM have also been plotted to aid in visualizing and clustering. PCA has been conducted to reduce dimensionality to a 2-dimensional space.

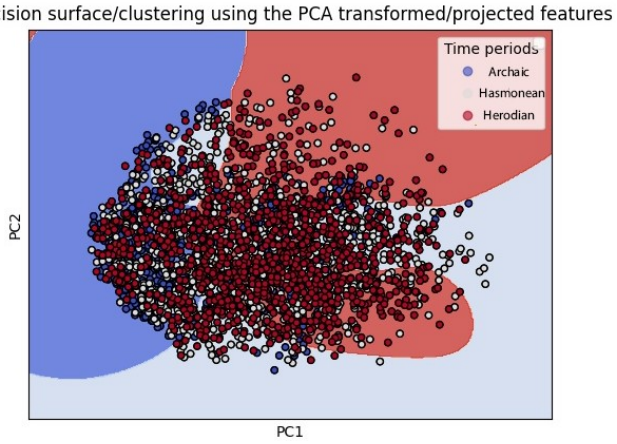


Fig. 16: HOG features applied into a two-dimensional space by plotting the first to principal components extracted from PCA. The decision boundaries created by the SVM are also plotted.

4) Methodology: In the sections below we will be explaining the methodology of how we set up the task to classify the three different styles. We developed the textural feature extraction method first proposed by Bulacu and Schomaker (2007), but the SVM classifier provided better results. We, therefore, selected the SVM classifier for the final implementation of the system.

a) Hinge global and local training: For calculating the trained pdf vectors, either a global or a local training method was used. In the case of local training, the classifier took into consideration the output label of the character recognition task. Here, as is evident from the name, the hinge pdf vector calculation is local; the classifiers are trained only on the labelled characters from the character recognition task. This means that for each segmented character in the image, the hinge feature pdf was calculated and compared. In the cases where the archaic dataset lacked some labels like Tsadi-final for feature calculation, global pdf vectors were used. The global method computes a pdf feature vector on all the data of the archaic, Hasmonean, and Herodian training sets respectively, resulting in three global pdf feature vectors,

one for each time period. For pdf comparison between the segmented character and local/global pdfs, we used the chi-squared distance:

$$\chi^2_{qi} = \frac{1}{2} \sum_{i=1}^{ndims} \frac{(p_{qn} - p_{in})^2}{p_{qn} + p_{in}} \quad (2)$$

where n are the number of dimensions in the feature vector, in our case 300, and q and i denote the different entries from the two pdfs to be compared. We used χ^2, as noted by Bulacu and Schomaker (2007), for its better performance when used with textural features methods.

The SVM was trained globally on all characters of each time period and therefore resulted in three different binary classification SVMs.

b) Character probability thresholding: We took extra steps to improve performance. Specifically, characters that were classified in character recognition with a probability lower than 70 $\%$ were not considered for style classification. This was done to account for potentially wrongly labelled characters, which would then undermine both classifier's performance.

c) selection of dominant style: After computing all individual character styles, to compute the dominant style of the image, we used two methods: 1) majority vote on all classified characters of the image, 2) top-10 nearest neighbors according to smallest χ^2 distances. The second method was therefore only restricted to the hinge vector classifier. We strayed from just using a majority vote for the hinge method to account for the model being biased toward a majority class, although we used it in the SVM classifier as SVMs work with maximizing decision boundaries when mapping characters to hyperplanes, which are not distance metrics, and cannot be therefore ordered in a top-10 table.

III. Results

The resulting performance of the methods can be split up into three sections

A. Text Segmentation

In order to assess the performance of both line segmentation and character segmentation, a quantitative evaluation via visual inspection was used.

a) Line segmentation: In the case of line segmentation, the system was tested on 22 images, where the evaluation procedure consisted of counting the number of intuitively required, the number of detected, and the number of sufficiently (subjectively) segmented lines. Table II gives a summary of these quantities. In total, from the 22 images, 66.25% of the required lines were detected and segmented properly, and there were 22.92% more lines detected than required. From these values, it is clear that the line segmentation was too sensitive, which resulted in several, wrong line position initialization that reduces the chances of an ideal segmentation with the A* path-planning algorithm. Figure 18 and Figure 19 in the Appendix illustrate the two test images with their segmented lines.

Image	Required	Detected	Sufficiently segmented
A (narrow)	10	10	8
B (wide)	11	11	7
20 non-labeled images	219	274	144

TABLE II: Number of lines identified in a visual inspection of 2 style-labeled test images, as well as 20 non-labeled images used for development.

b) Character segmentation: To evaluate the performance of the character segmentation, the number of characters were counted for two of the scrolls (Figure 18 and Figure 19 in the Appendix). The total number of identified characters were extracted after the model had performed character segmentation. Furthermore, the number of obvious artifacts were noted. The results can be seen in Table III. We can see that the model consistently identifies more characters than there are in the image, however, once the number of artifacts is subtracted from the number of identified characters we see that fewer characters were identified than present in the image. The probable reason for this discrepancy is due to some characters that were not identified to contain multiple characters, as well as single characters having been broken into artifacts being filtered. Looking more closely at the performance of the character segmentation we found that although the erosion/dilation method seemed to perform well (see Figure 11), however, this step was only managed to successfully erode 26.7% of all of the predicted connected characters and missed 73.2%. The characters that were missed were segmented using the sliding window approach which worked in some cases (see Figure 12), however, in other cases, segmentation was erroneous. As can be seen in Figure 17, sometimes the error originated with the calculation of the number of characters contained in a connected character, in other cases, however, these errors appeared when not enough characters were suspected to be contained.

Image	Visual	Model	Artifacts	Model - Artifacts
A (narrow)	74	101	39	62
B (wide)	223	244	28	216

TABLE III: Number of characters identified in a visual inspection of two test images compared to the number of identified characters by the model, as well as the number of artifacts erroneously identified as a character by the model. The final column is the number of characters identified by the model subtracted by the number of artifacts

Image	successful erosions	remaining characters	missed
A (narrow)	16	31	65.9%
B (wide)	19	65	77.4%
overall	35	96	73.2%

TABLE IV: Number of characters that were suspected to contain multiple characters that were segmented using erosion (successful) vs the number of characters that were not eroded successfully and passed onto the sliding window method

(a) (b) (c)

Fig. 17: a: suspected connected character. b: first split character, c: second split character

B. Character Recognition

a) Cross-validation: For the handcrafted CNN learning-rate of optimized for values in [0.0005, 0.0035] with steps of 0.0005. L2-weight decay was optimized for values in [0, 0.002] with steps of 0.0002. For the results of both cross-validations please see Figure 21. The best values found were: $learning\_rate = 0.0015$ and $L2\ weight\_decay = 0.0001$. These were used for training the model on the entire training set.

For AlexNet, the best values found were: $learning\_rate = 0.0005$ and weight decay of $L2\ weight\_decay = 0.001$.

b) Final results of all models: Results of the three different models that we used in for the character recognition task can be found in Table V.

Models	Test morph	std	Test no morph	Std
Handcrafted CNN	88.15	1.81	91.57	2.26
AlexNet	81.01	0.92	85.37	2.01
SVM	89.84	-	89.74	-

TABLE V: Accuracy of the character recognition models, measured on an augmented testing set and a testing set with only original characters. Performance was averaged over five runs for each model.

As can be seen, both the handcrafted CNN and the SVM perform similarly. Since the CNN gave a higher accuracy on the no-morph testing set it was used in the pipeline. AlexNet performs worse than the other two models.

C. Style Classification

The accuracies of both models used to classify the Style of a character can be seen in Table VI.

Models	Acc morph	Acc no-morph
Hinge-extraction w/ chi-squared	44.78 %	44.45%
SVM w/ HOG	80.43%	81.04%

TABLE VI: Accuracy for individual character style classification for hinge extraction and SVM model. Accuracies were measured on an augmented test set version as well as the non-augmented test set.

As can be seen, the SVM performs a lot better than minimal distance classification based on the extracted hinge features. Therefore the SVM was used in the final pipeline.

IV. Discussion and Conclusion

After reviewing the results several findings can be extracted. These findings, along with the possible shortcomings of the model will be presented in the following sections.

A. Text Segmentation

The results showed that horizontal projections seem to be a powerful and effective method to segment the lines, however, for lines that were closer together and curved this method required a rather high amount of processing steps to get reliable results. Even then, the method that was employed tended to be oversensitive and located more lines than present in the images.

Using A to segment the lines showed to be an effective and viable method when lines are clearly separated as a result from the previous step, however, if the area the agent is allowed to traverse through is too narrow, cuts must be made in certain areas that visually would not need to be cut. Although these were corrected partially, if the sections it is given are too small, these corrections will not be enough. Thus we can conclude that this method is reliable if it is presented with the appropriate starting information.

The largest problems seem to occur during character segmentation. Although in the cases where the characters were already separated the connected components method was able to perform extremely well, especially after dilating images to connect characters that had deteriorated over time, when characters were connected, however, this method of course requires several processing steps for the character to be returned cleanly. For one, the artefact removal was susceptible to the image size, as a number of threshold values were used to determine the minimum sizes of elements to pass an artefact check. To avoid artefact removal for removing important elements if the image size was too small, the image was scaled to a certain size at the beginning of the pipeline. This shows that this section of the model is prone to errors if certain assumptions are not met.

When separating connected characters, the erosion-dilation technique seemed to perform rather well when characters could be separated, however, this didn't happen frequently enough. The remaining characters were processed using the sliding window. This approach was not optimal and many erroneous characters managed to be passed on to the recognizer. In future research, this method of character separation should be redesigned and the erosion method strengthened.

B. Character Recognition

During testing the different models the results showed that although the transfer learning approach using a pre-trained AlexNet had previously been shown to work well for character recognition (Aneja & Aneja, 2019), it was not able to outperform the simpler, handcrafted model that was pre-trained on the HDD dataset. Perhaps the reason for this was the AlexNet was not only pre-trained on the HHD dataset but also the ImageNet dataset. Besides AlexNet performing worse than the handcrafted CNN, both CNNs outperformed the SVM suggesting that using a CNN is a

better approach to character recognition. Interestingly, both CNNs showed improved performance when the brill dataset was not morphed. This is possibly due to the morphing being too strong, or alternatively, the morphing used in this paper increased the ratios of class imbalance. For future studies, better morphing strategies should be investigated to avoid this decrease in performance. Furthermore, during training, not all training images were centred. This was an oversight and in a future study, each training image should be preprocessed such they represent the images that it would receive during character segmentation.

C. Style Classification

When comparing the methods used for style classification we can see a dramatic difference in performance. The results show that using an SVM with HOG greatly outperforms classification using the pdfs obtained from the hinge feature vector extraction. The reason for the large difference could point to possible shortcomings in the hinge extraction itself. Parts of the hinge extraction algorithm were coded by hand, which may have lead to missing values. The hinge extraction method used by Bulacu and Schomaker (2007) may be a way to improve our results as their results show a much better performance. Besides the possible errors found in the hinge feature extraction method, it seemed that the SVM using HOC was better suited for this dataset. It has already been shown that HOC performs well in helping to distinguish between writing styles (Hannad et al., 2016; Yilmaz et al., 2011).

D. Conclusion

Overall this paper tested several different methods in text segmentation, character recognition and style classification on the Dead Sea Scrolls. It results show that, although connected components perform well when the characters to be extracted are not connected, otherwise many artefacts are bound to appear. To counter this, more powerful connected character segmentation techniques should be employed, as the ones used did not manage to segment the characters cleanly. When it comes to character recognition, simple pre-trained CNNs give the highest test accuracies. These results could presumably be improved in future studies by using morphine techniques that create an equal split of the data. Finally, using SVMs using HOG are well suited for style recognition although hinge extraction has been shown to be better in other studies.

V. Contributions

See Appendix -D for an overview of the team contributions during this project.

References

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (icet)* (p. 1-6). doi: 10.1109/ICEngTechnol .2017.8308186

Aneja, N., & Aneja, S. (2019). Transfer learning using cnn for handwritten devanagari character recognition. In *2019 1st international conference on advances in information technology (icait)* (pp. 293–296).

Atwill, J., & Braunheim, S. (2004). Redating the radiocarbon dating of the dead sea scrolls. *Dead Sea Discoveries*, *11*(2), 143–157. Retrieved from `http://www.jstor.org/stable/4193320`

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the 5th annual acm workshop on computational learning theory* (pp. 144–152). doi: 10.1145/130385.130401

Bulacu, M., Brink, A., Zant, T., & Schomaker, L. (2009, 01). Recognition of handwritten numerical fields in a large single-writer historical collection. In (p. 808-812). doi: 10.1109/ICDAR.2009.8

Bulacu, M., & Schomaker, L. (2007, 05). Text-independent writer identification and verification using textural and allographic features. *IEEE transactions on pattern analysis and machine intelligence*, *29*, 701-17. doi: 10.1109/TPAMI.2007.1009

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-8*(6), 679-698. doi: 10.1109/TPAMI.1986.4767851

Cao, Z., Huang, M., & Wang, Y. (2010). A new drop-falling algorithms segmentation touching character. In *2010 ieee international conference on software engineering and service sciences* (p. 380-383). doi: 10.1109/ICSESS.2010.5552365

Casey, R. G., & Lecolinet, E. (1996). A survey of methods and strategies in character segmentation. *IEEE transactions on pattern analysis and machine intelligence*, *18*(7), 690–706. doi: 10.1109/34.506792

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 ieee computer society conference on computer vision and pattern recognition (cvpr'05)* (Vol. 1, p. 886-893 vol. 1). doi: 10.1109/CVPR.2005.177

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 ieee conference on computer vision and pattern recognition* (pp. 248–255). doi: 10.1109/CVPR.2009.5206848

Ding, J., Chen, B., Liu, H., & Huang, M. (2016). Convolutional neural network with data augmentation for sar target recognition. *IEEE Geoscience and remote sensing letters*, *13*(3), 364–368.

Dos Santos, R. P., Clemente, G. S., Ren, T. I., & Calvalcanti, G. D. (2009). Text line segmentation based on morphology and histogram projection. In *Proceedings of the international conference on document analysis and recognition, icdar* (pp. 651–655). doi: 10.1109/ICDAR.2009.183

Duda, R. O., & Hart, P. E. (1972). Use of the hough transformation to detect lines and curves in pictures.

Communications of the ACM, *15*(1), 11–15. doi: 10.1145/361237.361242

Garris, M., & Dimmick, D. (1996). Form design for high accuracy optical character recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *18*(6), 653-656. doi: 10.1109/34.506417

Hannad, Y., Siddiqi, I., El Merabet, Y., & El Youssfi El Kettani, M. (2016). Arabic writer identification system using the histogram of oriented gradients (hog) of handwritten fragments. In *Proceedings of the mediterranean conference on pattern recognition and artificial intelligence* (p. 98–102). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3038884.3038900

Khan, H. (2017, 01). Mcs hog features and svm based handwritten digit recognition system. *Journal of Intelligent Learning Systems and Applications*, *09*, 21-33. doi: 10.4236/jilsa.2017.92003

Kim, G., & Govindaraju, V. (1997). A lexicon driven approach to handwritten word recognition for real-time applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*(4), 366-379. doi: 10.1109/34.588017

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012, 01). Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, *25*. doi: 10.1145/3065386

Kumar, G., Bhatia, P. K., & Banger, I. (2013). Analytical review of preprocessing techniques for offline handwritten character recognition. *International Journal of Advances in Engineering Sciences*, *3*(3), 14–22. doi: 10.13140/RG.2.1.3896.7842

Likforman-Sulem, L., Hanimyan, A., & Faure, C. (1995). A hough based algorithm for extracting text lines in handwritten documents. In *Proceedings of 3rd international conference on document analysis and recognition* (Vol. 2, pp. 774–777). doi: 10.1109/ICDAR.1995.602017

Likforman-Sulem, L., Zahour, A., & Taconet, B. (2007). Text line segmentation of historical documents: a survey. *International Journal of Document Analysis and Recognition (IJDAR)*, *9*(2-4), 123–138.

Malakar, S., Paul, S., Kundu, S., Bhowmik, S., Sarkar, R., & Nasipuri, M. (2020). Handwritten word recognition using lottery ticket hypothesis based pruned cnn model: a new benchmark on cmaterdb2. 1.2. *Neural Computing and Applications*, *32*(18), 15209–15220.

Marti, U. V., & Bunke, H. (2001). Text line segmentation and word recognition in a system for general writer independent handwriting recognition. *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, *2001-Janua*, 159–163. doi: 10.1109/ICDAR.2001.953775

Mehul, G., Ankita, P., Namrata, D., Rahul, G., & Sheth, S. (2014). Text-based image segmentation methodology. *Procedia Technology*, *14*, 465–472. doi: 10.1016/j.protcy.2014.08.059

Muthukrishnan. (2019). *Segmenting lines in handwritten documents using a* path planning algorithm.* Retrieved from `https://muthu.co/segmenting-lines-in-handwritten-documents-using-a-path-planning-algorithm/`

Qaroush, A., Awad, A., Modallal, M., & Ziq, M. (2020). Segmentation-based, omnifont printed arabic character recognition without font identification. *Journal of King Saud University-Computer and Information Sciences*.

Rabaev, I., Barakat, B., Churkin, A., & El-Sana, J. (2020, 08). The hhd dataset.. doi: 10.1109/ICFHR2020.2020.00050

Rawat, W., & Wang, Z. (2017, 06). Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, *29*, 1-98. doi: 10.1162/NECO_a_00990

Sambyal, N., & Abrol, P. (2016). Connected component based english character set segmentation. *Int J Scientific Tech Advancements*, *2*(4), 303–306.

Sarowar, M. G., Razzak, M. A., & Fuad, M. A. A. (2019). Hog feature descriptor based pca with svm for efficient accurate classification of objects in image. In *2019 ieee 9th international conference on advanced computing (iacc)* (p. 171-175). doi: 10.1109/IACC48062.2019.8971585

Seni, G., & Cohen, E. (1994). External word segmentation of off-line handwritten text lines. *Pattern Recognition*, *27*(1), 41–52.

Surinta, O., Holtkamp, M., Karabaa, F., Oosten, J. P. V., Schomaker, L., & Wiering, M. (2014). A Path Planning for Line Segmentation of Handwritten Documents. *Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR*, *2014-Decem*, 175–180. doi: 10.1109/ICFHR.2014.37

Tang, Y., Peng, L., Xu, Q., Wang, Y., & Furuhata, A. (2016). Cnn based transfer learning for historical chinese character recognition. In *2016 12th iapr workshop on document analysis systems (das)* (pp. 25–29).

Tian, Y., Song, Y., Xia, F., Zhang, T., & Wang, Y. (2020, July). Improving Chinese word segmentation with wordhood memory networks. In *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 8274–8285). Online: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/2020.acl-main.734` doi: 10.18653/v1/2020.acl-main.734

Tov, E. (2016). *Dead sea scrolls electronic library.* Leiden, The Netherlands: Brill. Retrieved from `https://brill.com/view/package/dsso`

Wai, M. T., Zin, T. T., Yokota, M., & Mya, K. T. (2019). Handwritten character segmentation in tablet based application. In *2019 ieee 8th global conference on consumer electronics (gcce)* (p. 760-761). doi: 10.1109/GCCE46687.2019.9015340

Yilmaz, M. B., Yanikoglu, B., Tirkaz, C., & Kholmatov, A. (2011, 10). Offline signature verification using

classifier combination of hog and lbp features. In (p. 1-7). doi: 10.1109/IJCB.2011.6117473

APPENDIX

A. Line segmentation results on two test images

Fig. 18: First test image (image A) of line segmentation. 10/10 lines were detected, with 6 of them being sufficiently segmented.

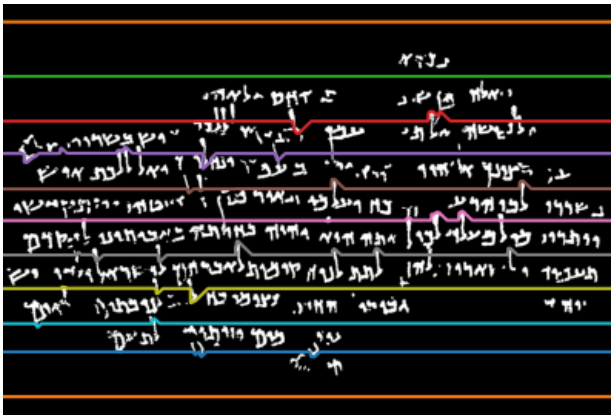
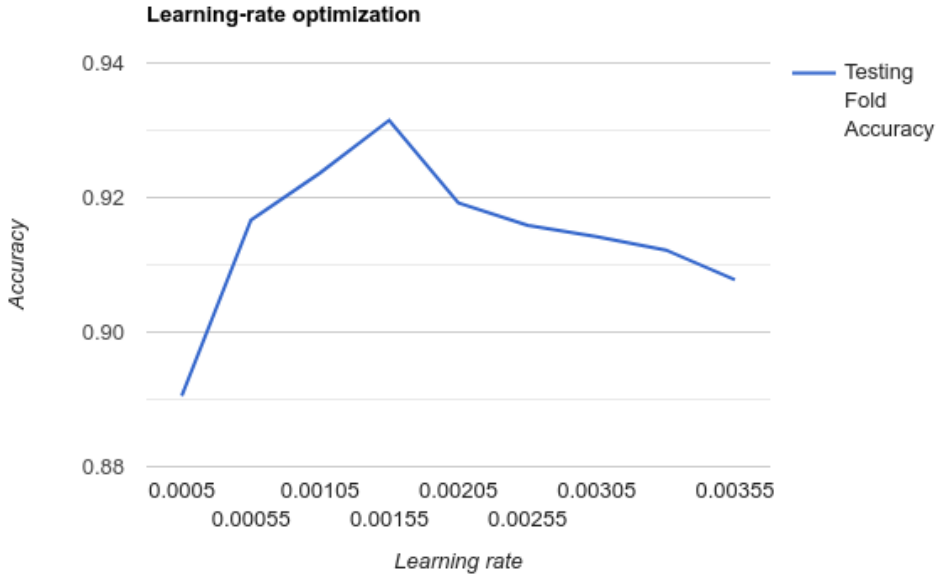


Fig. 19: Second test image (image B) of line segmentation. 11/11 lines were detected, with 6 of them being sufficiently segmented.

B. Characters that hinge extraction and contour sorting were tested on

Fig. 20: Some sample characters that contour coordinate sorting and hinge extraction were tested on

C. Cross-validation graphs



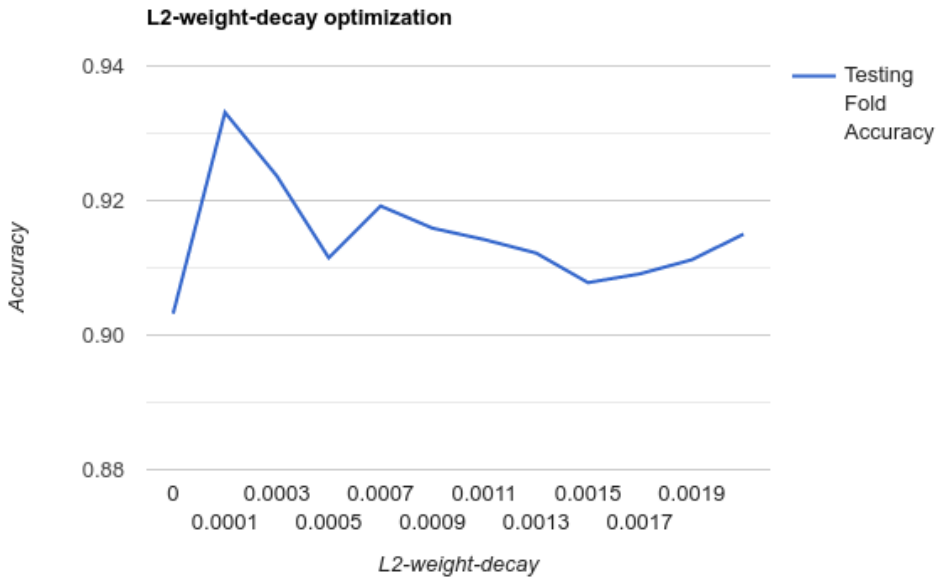


Fig. 21: Optimization of learning rate and L2-weight decay of the handcrafted CNN, using 5-fold cross validation

D. Individual contributions

In the table below you may find a summary of the individual contributions. For the majority of the project, we had been working in groups of two. Charlie and Péter focused on the entirety of Text Segmentation, whereas Jan and Panos started with Character Recognition. Except for the implementation of Word Segmentation, Text Segmentation was carried out while being in a call for every coding session, therefore specifying the contributions in more detail would not be feasible. As Jan and Panos managed to finish the Character Recognizer a lot earlier than Charlie and Péter did the Text Segmentation, they proceeded with the Style Classification. Overall we all agree that we contributed to the project equally and that the workload was split fairly.

Regardless of the split-up into groups, we had one or two meetings every week to discuss the significant decisions regarding any element of the whole pipeline.

Name	Coding	Writing
Charlie	- Line segmentation - Word segmentation - Character segmentation	- Methods/Parts of Datasets - Methods/Parts of Line segmentation - Methods/Word Segmentation - Methods/Character Segmentation - Results/Character segmentation - Proofreading
Péter	- Line segmentation - Character segmentation - First (naive) attempt of PCA based dimensionality reduction on entire images	- Methods/Line segmentation - Methods/Parts of Character segmentation - Results/Line segmentation - Proofreading and editing parts of Methods/Datasets, Character Recognition:Models, Style Classification sections
Jan	- Handcrafted CNN for character recognition - Data morphing and some data preprocessing - Hinge feature extraction (constructing the feature vector for a given image) - SVM for style classification (using functions Panos made for SVM character recognition)	- Methods/Datasets/Morphing - Methods/Character Recognition:Models/CNN/Handcrafted CNN - Methods/Style Classification/Hinge feature extraction -Results/Character recognition -Results/Final results of all models -Proofreading
Panagiotis	-Alexnet development -SVM development for char recognition -hinge classfier set-up for style classification -data preprocessing	-Methods/Datasets -Methods/Character Recognition:Models/CNN/AlexNet -Methods/Style Classification/Methodology -Methods/Style Classification/Support Vector Machines -Results/Style Classification -Results/Final results of all models