Pattern Recognition Project Final Submission

Arabic OCR System

Semester - Team 14

Ahmed Essam S.1 BN. 6 Ahmed Khaled S. 1 BN. 03

Sara Maher S. 1 BN. 28 Ibrahim Mahmoud S. 1 BN. 1

Direct contact to: Ahmed Khaled at akregeb@gmail.com.

1 Pipeline

The main pipeline we have is as follows:

1. Preprocess the image: split it into lines, and split the lines into words.
2. For each word: compute the feature vector through the Discrete Cosine Transform (DCT) and the Blocked DCT.
3. From a trained library of cluster centers, find the cluster to which the word belongs.
4. Find the nearest neighbor to the query word in the cluster, find its Arabic equivalent, and output it.

There is no post-processing module.

2 Preprocessing

For preprocessing, we do the following:

1. Turn the image into grayscale.
2. Deskew the image (if skewed) by recognizing the major angle and rotating the document accordingly.
3. Invert the image colors and binarize it.
4. Segment the image into lines, using horizontal projection. We split at the zero crossings of the horizontal projection after smoothing and thresholding by an appropriate value.
5. Segment the lines into words, using vertical projection. Again, we smooth every line and threshold it, then detect zero crossings.

The line segmentation accuracy is 100% measured on a few sample files. The word segmentation accuracy is about 70% in the sense that 70% of files have their exact number of words segmented correctly. Even then, the files where the word segmentation made mistakes usually had only one or two mistakes. The main problem is that sometimes brackets are not added correctly to the word in the word segmentation stage. However, this would not be a major problem given that the word-to-character segmentation will segment correctly.

3 Feature Extraction and Recognition

For this phase, we followed the holistic, word-based approach [6]. For every word, we extract features by performing the following steps:

1. Crop the word such that there are no edge columns or edge rows with no white pixels.

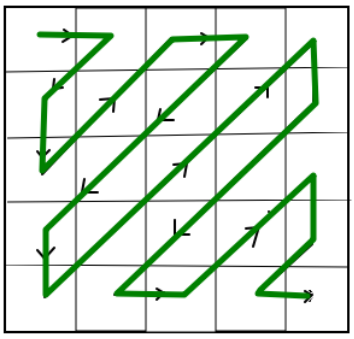

Figure 1: Zigzag traversal of a matrix.

2. Compute the two-dimensional Discrete Cosine Transform (DCT) of the word as follows: Given a word image W of size $M \times N$, we have

$$\mathrm{DCT}(W)[i,j] = C_i C_j \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} W[x,y] \cos\left(\frac{(2x+1)i\pi}{2M}\right) \cos\left(\frac{(2y+1)j\pi}{2N}\right).$$

   where $C_i = 1/\sqrt{M}$ if $i = 0$ and is equal to $2/\sqrt{M}$ otherwise, and similarly for C_j with N instead of M.

3. Extract the first K features by traversing the DCT matrix in a zigzag fashion until K features have been collected, as shown in Figure **??**. The zigzag approach ensures we include the most important DCT components.

4. Find the centroid of the image, divide it into four blocks, and compute a feature vector based on the DCT (as in the last two steps) for each subblock.

5. Concatenate the main image feature vector and all subblocks feature vectors. For our purposes, $K = 100$ was chosen and the subblocks contributed 100 components as well. In total, each word was transformed into a feature vector with 200 numbers. When any of the constitutent feature vectors were too small, they were padded with zeros.

The previous steps explain feature extraction. In the training phase, we do the following:

1. We loop over all the words in the training set, calculate the features, and store a mapping between the features and the word text. We collect all the features into a big feature matrix.

2. We use k-Means to cluster all of the collected feature vectors in the feature matrix into 1024 clusters.

3. We find all the words associated with each cluster and store them in an external file.

We can then use the extracted database in the inference phase as follows:

1. Extract the words from the image, find the feature vector corresponding to each word (through the DCT and the block DCT).

2. From the database of clusters, find the cluster to which the word belongs.

3. Find the nearest neighbor to the query word in the cluster, find its Arabic equivalent, and write it out.

We partitioned the given dataset into a train/test split and tested this method on 60 file: the method spent an average time of 3.4 seconds on every file (disregarding the initial cost of loading the database) and had a total accuracy (in terms of the edit distance) of 91%.

4 Other Approaches we tried

4.1 Character Segmentation

4.1.1 Latifa et al. (2002) [2]

The method consists of the following steps:

1. Segment words into connected parts using vertical projections.

2. Find the junction line of each connected part: the line with the highest concentration of black pixels.

3. Find the top line of each connected part: the highest line corresponding to the first encountered black pixel.

4. Find the bottom line of each connected part: the lowest line corresponding to the last black pixel.

5. Find the histogram of pixels per column for each connected part.

6. Extract some statistical data (the mode and number of transitions).

7. Depending on various rules relating to the statistical data and the relation between junction, bottom, base, and top lines segment the characters (described in detail in the paper).

Unfortunately this method does not have very high accuracy, as many of the assumptions made in the paper are violated by our font: for example, some characters are overlapping and for some characters (like raa) there is no rise after connection. For these reasons, we did not continue with this method to the classification phase. Some results are given in Figure 2.

4.1.2 Sari et al. (2002) [8]

This method consists of the following steps:

1. **Connected component extraction:** the outer contour of the word is obtained using a contour following algorithm. To achieve this step, We used OpenCV's function

   ```
   cv2.findContours()
   ```

Figure 2: Latifa et al. [2] method results. Every two lines determine a segmentation region.

2. **Morphological feature extraction:** this step is concerned with extracting the useful information from characters' shape and labelling them with different labels (Ascender, descender, holes, turning point, hamza, one-dot, two-dot, ...). E.g: characters that take the "Ascender" label are characters that extend above the median of the word's horizontal projection & characters that take the "Holes" label are the ones with has holes in their structures and so on. This step is unimplementable due to the fact that there is no character segmentation so far. This looks like the "chicken or the egg" problem as we need segmented characters to classify them with the correct label and we need the character classification in order to get a good segmentation as we will see in the next step.

3. **Topological filtering operations (TOFO):** in order to develop a tough Arabic segmentation algorithm, it is important to exploit the properities of the Arabic writing. We take the observation concerning the hand movement at the writing time: We draw the body of the first letter and when we want to draw the second letter, we mark the separation by going up which generates local points inevitable. This local minima is exploited in this step as a candidate segmentation point in order to accept or reject them according to some rules like rejecting the local minima if it cuts a hole (a loop) or accepting it if it follows a hole (a loop).

At that point, we should have some Valid Segmentation Points (VSPs) that we can use to segment the word into its characters.
After implementing a reasonable part of the previously discussed method, it was clear that it wouldn't result in a good segmentation algorithm due to a number of reasons:

1. The fact that a segmentation point is always a candidate local minima to start with, is not

Figure 3: Sari et al. [8] method results

viable in all fonts (as shown in figure 3)

2. The rules used in the 3rd step relies heavily on the classification of the letters which can't be done without proper segmentation.

We include some examples of the segmentation done using this method which shows its shortcomings in some cases in figure 3.

4.1.3 Qaroush et al. (2019) [7]

This method takes the following steps:

1. Detect the baseline of the word (the horizontal row with the most pixels). This involves thinning the line, computing the horizontal projection, and then finding peak values.
2. Find the points of maximum transition: i.e. the point where the horizontal derivative is high. These points represent potential segmentation points.
3. Identify all potential cut points as possible lines that separate segmentation regions. This is done by computing certain histogram statistics as well the existence of paths in the image treated as a graph, and then comparing the computed quantities against thresholds.
4. Separation region filtration: tries to fix over-segmentation by removing cut points that are not admissible according to certain morphological and graph-theoretic criteria.

Some of the results are shown in Figure 4. Unfortunately, the method's accuracy is too low and we abandoned it.

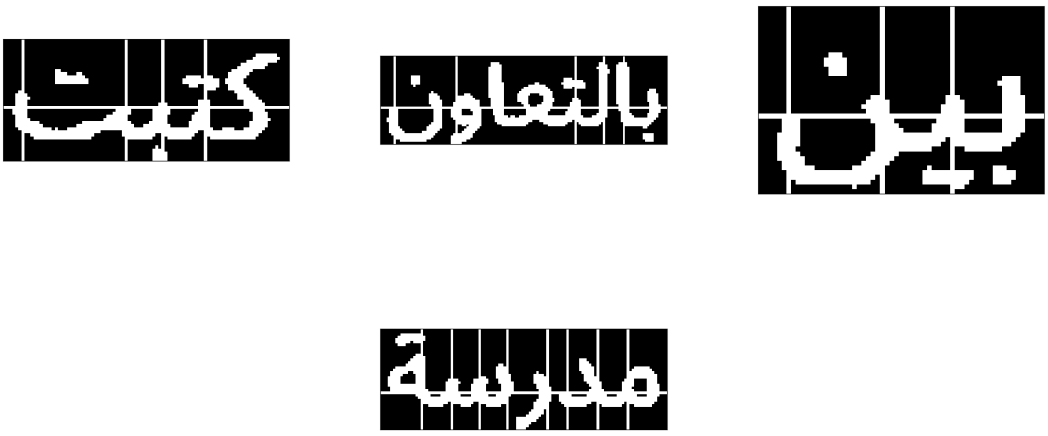

Figure 4: Qaroush et al. [7] method results

4.2 Histogram Application and Baseline Detection (HABD) method

We tried to adapt the character segmentation to the specific font used and came up with the HABD algorithm:

1. Find all contours of the word. Throw away contours that have parent contours (these represent holes).
2. Find all dots of the word, throw them away.
3. Segment the word into subwords by finding the block corresponding to each contour.
4. If the size of the subword is below a certain threshold, it is immediately returned for recognition (as it is probably an isolated word).
5. Find the skeleton of the word, find the baseline (horizontal line with the maximum number of white pixels).
6. Traverse the image column-by-column from left to right: if there are no pixels both above and below the baseline, then this is a possible segmentation point, mark it as such. If there is a run of segmentation points (i.e. they form a continuous horizontal line) take only the last of these points.
7. The possible segmentation points returned by the algorithm are then filtered for specific edge cases like the over-segmentation of seen/sheen. This is done mainly using checks on the histogram of the image as well as baseline values.

The chief difficulty in HABD is the final step: there are just too many edge cases. Nevertheless, it was the best segmentation-based method we tried. Some examples of segmentation by this method are included in Figure 5.

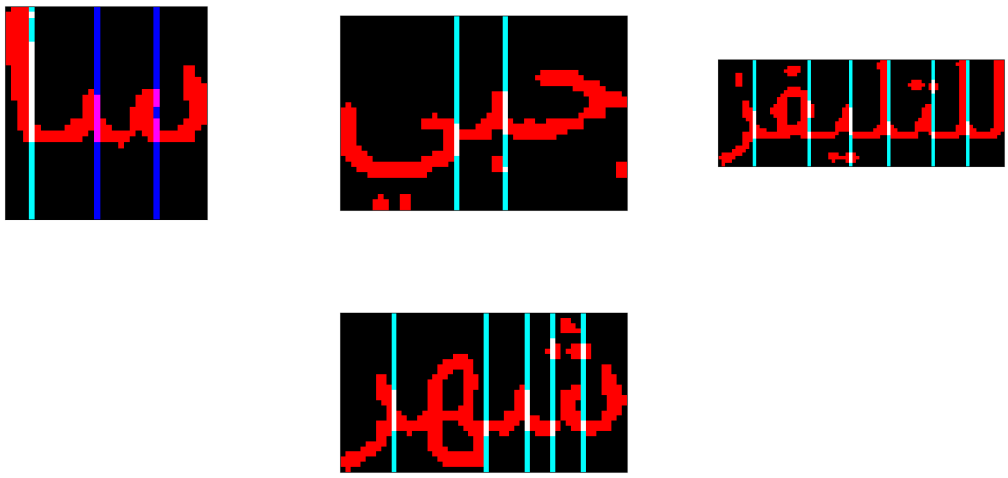

Figure 5: HABD method results. Teal lines indicate actual segmentation while blue lines indicate potential segmentation points that were removed by the algorithm.

4.3 Recognition-Only Methods

We tried extracting some features from the characters before passing them to the model in order to help the model recognize the letters better. We used 14 features, 4 of which are global features for the image:

1. height / width
2. Number of white pixels / number of black pixels
3. Number of vertical transitions
4. Number of horizontal transitions

and 10 of which are local to specific regions of the character letter:

1. White Pixels in Region 1/ Black Pixels in Region 1
2. White Pixels in Region 2/ Black Pixels in Region 2
3. White Pixels in Region 3/ Black Pixels in Region 3
4. White Pixels in Region 4/ Black Pixels in Region 4
5. White Pixels in Region 1/ White Pixels in Region 2
6. White Pixels in Region 3/ White Pixels in Region 4
7. White Pixels in Region 1/ White Pixels in Region 3

8. White Pixels in Region 2/ White Pixels in Region 4
9. White Pixels in Region 1/ White Pixels in Region 4
10. White Pixels in Region 2/ White Pixels in Region 3

After extracting those features from each image, we train a support vector machine (SVM) model and k-nearest neighbors (KNN) model on the generated features and it gave an F1-score of nearly 90% for each character separately.

4.4 Segmentation-Free Methods

4.4.1 Khorsheed (2007) [3] and Khorsheed (2015) [4]

Hidden Markov Models have the ability to automatically segment and recognise characters, and are naturally well-suited to handling sequences of characters. In [3] statistical features are used with Hidden Markov Models to segment and recognise characters. This approach was improved by the same author in [4] using run-length coding. We tried the second approach, which takes the following steps:

1. Preprocess every line image and normalize its height to a certain number (we chose 60 pixels).
2. Find the run-length code of each column of the line.
3. Based on a training set, cluster the run-lengths into 1024 different clusters, and find the center for each cluster.
4. Given a new line column to recognise (outside of the training set), find the center of the cluster closest to the line. This center is then the feature vector for this column. Repeat this for all columns.
5. Replace each center vector by its index (i.e. the cluster number it belongs to). Pass that into the trained Hidden Markov Model.
6. Infer the sequence of states that maximizes the likelihood of the observed line. The sequence of states will indicate which characters are in the line.

The method did not work well: we believe that is due to the following difficulties:

1. The toolkit used in [4], the Hidden Markov Model Toolkit (HTK), had a very steep learning curve. Therefore, we had to use other tools which were not as good.
2. Training the Hidden Markov Model (HMM) required a database of segmented characters. We had no access to such a database and even when we manually created our own it was a very time consuming process and there were not enough examples for the HMM to learn each character.

As a result of the dearth of data, there were not enough examples for the HMM to even pass cross validation. Therefore, we did not continue this approach further.

5 Future Work

There are several approaches that we did not try that look promising:

1. In [1], a sliding window approach is used with Hidden Markov Models to segment and recognise characters in conjunction.
2. In [9], an approach that uses the Generalized Hough Transform in conjunction with Hidden Markov Models has also been used to segment and recognise characters.
3. In [5], features are extracted using the Wavelet transform, which can be seen as an alternative to the DCT we used.

Many of these approaches are promising, and we leave exploring them for future work.

6 Workload Distribution

Ahmed Khaled implemented [2], [6], [4], and the HABD method. Ibrahim implemented [8] and character-only recognition, Sara implemented [7] and participated partially in implementing [4]. Ahmed Essam implemented preprocessing and participated partially in implementing [6].

References

[1] Husni A. Al-Muhtaseb, Sabri A. Mahmoud, and Rami S. Qahwaji. Recognition of off-line printed arabic text using hidden markov models. *Signal Processing*, 88(12):2902 – 2912, 2008.

[2] Latifa Hamami and Daoud Berkani. Recognition system for printed multi-font and multi-size arabic characters. *Arabian Journal for Science and Engineering*, 27, 04 2002.

[3] M.S. Khorsheed. Offline recognition of omnifont arabic text using the hmm toolkit (htk). *Pattern Recognition Letters*, 28(12):1563 – 1571, 2007.

[4] M.S. Khorsheed. Recognizing cursive typewritten text using segmentation-free system. *The Scientific World Journal*, 2015(818432), 2015.

[5] A. Mowlaei, K. Faez, and A. T. Haghighat. Feature extraction with wavelet transform for recognition of isolated handwritten farsi/arabic characters and numerals. In *2002 14th International Conference on Digital Signal Processing Proceedings. DSP 2002 (Cat. No.02TH8628)*, volume 2, pages 923–926 vol.2, July 2002.

[6] Farhan Nashwan, Mohsen Rashwan, Hassanin Al-Barhamtoshy, Sherif Abdou, and Abdullah Moussa. A holistic technique for an arabic ocr system. *Journal of Imaging*, 4:6, 12 2017.

[7] Aziz Qaroush, Bassam Jaber, Khader Mohammad, Mahdi Washaha, Eman Maali, and Nibal Nayef. An efficient, font independent word and character segmentation algorithm for printed arabic text. *Journal of King Saud University - Computer and Information Sciences*, 2019.

[8] Toufik Sari, Labiba Souici-Meslati, and Mokhtar Sellami. Off-line handwritten arabic character segmentation algorithm: Acsa. pages 452– 457, 02 2002.

[9] S. Touj, N. E. B. Amara, and H. Amiri. Two approaches for arabic script recognition-based segmentation using the hough transform. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 654–658, Sep. 2007.