Offline Arabic Handwriting Recognition using Hidden Markov Models and Post-Recognition Lexicon Matching

Ahmed H. Metwally, Mahmoud I. Khalil, Hazem M. Abbas
Computers and Systems Engineering Department
Faculty of Engineering, Ain Shams University
Cairo, Egypt

***Abstract*— Arabic handwriting recognition is fairly complex operation due to the similarities between different letters under similar writing styles. This paper presents a new approach on off-line recognition of handwritten Arabic words. The method does not require the segmentation of words into characters for recognition, but requires segmentation of training data into separate letters. The method trains a Hidden Markov Model (HMM) for each letter in the alphabet along with its various writing styles and taking into consideration the letter position variations. Having different types of features to extract from each image in the dataset helps to further improve the recognition rate of the whole system. The performance of the proposed method is demonstrated after various experiments carried out using the IFN/ENIT (Institut fur Nachrichtentechnik/ Ecole Nationale d'Ingénieurs de Tunis) reference database which contains an estimate of 32,492 handwritten Arabic words with various writing styles and formats of hundreds of writers. The overall recognition rate of the system is 87% after applying various methods of fine tuning and optimizations.**

Keywords*— *Arabic Handwriting Recognition, Hidden Markov Models, Post-Recognition Lexicon Matching.

I. INTRODUCTION

The subject of text recognition has been an ongoing research interest for multiple years. To have the information in the digital form provides a huge motivation to find a way to be able to convert handwritten text into its digital equivalent. The problem of designing an ideal recognizer where all used patterns would be classified perfectly is unlikely to give good generalization. Since it seems to be tuned to the particular training system, rather than the feature(s) or model(s) that will have to be classified. The question is how to optimize this trade-off: generalization versus simple classifier [1].

In recent years, a lot of research has been done the field of offline Arabic handwriting recognition and specifically using HMMs [2]–[4]. Despite these efforts, offline Arabic handwriting recognition is still a highly complex task due to variations in the writing style of the same letter or word from person to another. Also the difficulty of segmentation because of the cursive nature of the Arabic writing. This paper will discuss a new approach for pre-processing image files. Also, a new technique to enhance recognition rate as a post-processing phase without adding much complexity to the system. Since HMMs can handle input noise and variations in the writing styles.

The paper will be organized as follows: Section II describes the overall System Architecture. Section III shows the Data Preparation phase. Section IV discusses the Feature Extraction stage. Section V shows the Post-Recognition Lexicon Matching. Section VI shows Experimental Results. Section VII is the Conclusion

II. SYSTEM ARCHITECTURE

An overview of the proposed system architecture is shown in Figure 1.

Starting from the database images provided by IFN/ENIT [5], the images are inserted into a pre-processing phase as it will be explained in Section 3. There is two ways: the first one is in case of the data will be used for training. This implies that the images will be segmented into separated characters and then goes to the next phase which is feature extraction. The second case will be during the recognition phase, i.e. the image is being used as a whole word without separation of characters.

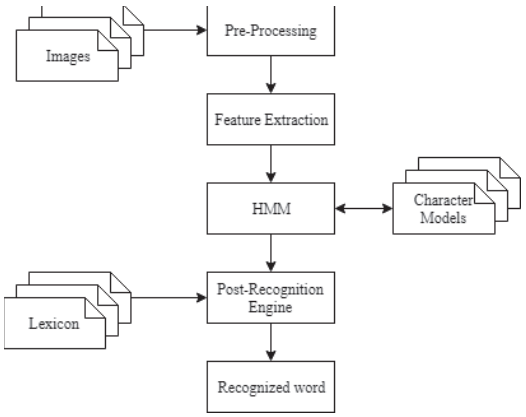

Figure 1: Overall System Architecture

The next stage will be the feature extraction phase. The image is used to extract some features to be used with the HMM without using any redundant information. This will speed up the whole process, and reduce the overall memory consumption of the system. The feature extraction phase was discussed briefly in Section 4.

The next step will be inserting the preprocessed features into the HMM engine, where the used approach is based on separate character models, i.e. each character in the lexicon is represented by a complete HMM model. Each character model has a right-to-left topology (since Arabic is written from right to left). Following the guidance of multiple researchers [6], [7] and also our experiments (more details can be found in Section 6), the following system parameters were chosen:

- A five-state model, where the first and last states provide no observations.
- Each state has three transitions only: a next state transition, a self-state transition and a skip-state transition.
- Four Gaussian mixture models used for each state.

The overall HMM topology can be seen in Figure 2. The toolbox used to model HMMs is called HTK (HMM Toolkit) [8]. HTK was mainly developed to be used for speech recognition, but has been deployed in various applications that uses HMMs.

III. Data Preparation

Figure 3 shows the steps taken to pre-process the image files before being fed to the training phase. The first three steps are done in both training and recognition phases, but the last stage (Separate Letters) is only done in case of preparation of training data. The reason of using this stage is to be able to divide the Arabic letters into separate groups of primitive letters. Primitive here means the removal of dots and diacritics accompanying the letters. This form will make a lot of letters look exactly the same, for example, letters such as 'beh', 'teh', 'theh' and 'noon' are actually the same letter or primitive but are accompanied with a dot in a certain position to differentiate between them. This grouping phase resulted in a total of 49 groups or primitives instead of approximately 100 in case of normal letters. Those groups or primitives can be seen in Table 1.

A. Diacritics removal

The recognition algorithm implemented in this paper operates when the diacritics are fully removed, so for this stage an algorithm was implemented to handle the removal of the diacritics with its various shapes. The algorithm works by calculating the area of each connected component in the image and then by means of averaging out the various areas calculated. A threshold is then calculated to be used as a baseline indicating anything smaller is a diacritic, and anything bigger is an actual letter in the image [9], [10].

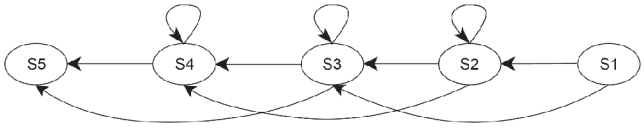

Figure 2: HMM topology

B. Image Resizing

The next step in the pre-processing stage, each word from various datasets is grouped together with all similar word images inside a common folder. All the files included in that directory will be resized according to the average size of all images, this eliminates any variation in image sizes, and only leaves writing styles and letter shape as the only variables.

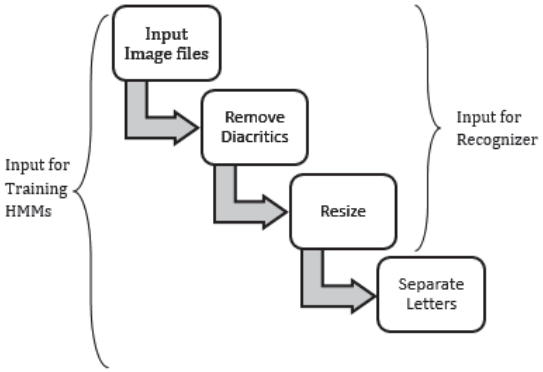

Figure 3: Pre-processing stage

TABLE 1 VARIOUS PRIMITIVES(LETTERS AFTER REMOVING DIACRITICS AND GROUPING THEM)

Position	Isolated	Start	Middle	End
Letter Cases	reh-zeen	geem-haah-khah	teh-theh-noon-yeh	beh-the-theh-noon
	beh-teh-theh-noon	seen-sheen	geem-hah-khah	alef
	geem-haah-khah	saad-daad	seen-sheen	ain-ghain
	daal-thaal	ain-ghain	saad-daad	seen-sheen
	lam-kaf	feh-qaf-	feh-qaf	saad-daad
	seen-sheen	kaf	kaf	
	saad-daad	lam	lam	kaf-lam
	tah-zah-(tah-zah)start	heh	heh	feh-qaf
	ain-ghain	beh-teh-theh-noon-yeh	tah-zah-(tah-zah)end	daal-thal
	feh-qaf		meem	meem
	meem	meem	ain-ghain	heh
	heh			yeh
	yeh			reh-zen-
	alef			waw
	waw			

C. Letter Separation

The last stage of pre-processing the images is letter separation or segmentation. In this stage, each word is taken and separated into its forming letter. This step is done for each word and all the resulting files will be merged together for each similar letter position. For each letter position, there are multiple entries containing variations of the same letter but with different writing styles. This is shown in Figure 4.

Figure 4: Variations in writing style of the letter

IV. Features Extraction

The next stage will be features extraction, where the words images will be processed to extract a set of features instead of using the whole image, reducing the overall redundancy in the images in the form of pixels. The extracted features are then represented as a vector of values that will be passed to the recognizer stage [11]. In order to proceed in producing the feature vector sequence, the image will be split into vertical overlapping windows [1]. Having a fixed window width. This window is passed through the image from right to left to extract the features from each isolated frame [1]. From each frame 21 features are extracted having two types of features: Concavity features and Distribution features.

A. Concavity features

Concavity features provide information related to local concavity and writing stroke direction in each isolated vertical frame. This approach will be using logic '0' to indicate background pixels and logic '1' to indicate a foreground pixel [11].

A hybrid method between [12], [13] was implemented, where the concept of a vertical sliding window with fixed size moves from right to left to extract 8 types of features explored using a vertical fixed size window as shown in Figure 5.

Find all the (0: Background) pixels having neighboring (1: Foreground) pixels in any of the following directions: (Up *'U'*, Down *'D'*, Right *'R'*, Left *'L'*, Right-Up *'RU'*, Left-Up *'LU'*, Right-Down *'RD'*, and Left-Down *'LD'*) such that [13]:

- The *'R'* feature represents the right boundaries of the word, while the *'L'* feature represents the left boundaries of the word.
- The *'U'* feature represents the upper boundaries of the word, while the *'D'* feature represents the lower boundaries of the word.
- The *'RU'* feature represents the upper right boundaries of the word, while the *'LD'* feature represents the lower left diagonal boundaries of the word.
- The *'LU'* feature represents the upper right diagonal boundaries of the word, while the *'RD'* feature represents the lower right diagonal boundaries of the word.

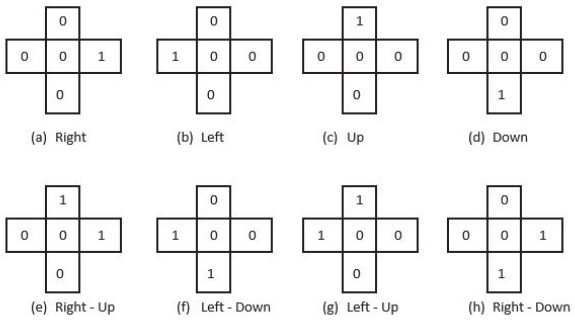

Figure 5: The eight filters used to extract the concavity features

The concavity features are calculated as follows [12], [13]:

$$F1 = \frac{\sum R}{H} \qquad F5 = \frac{\sum RU}{H}$$

$$F2 = \frac{\sum L}{H} \qquad F6 = \frac{\sum LD}{H}$$

$$F3 = \frac{\sum U}{H} \qquad F7 = \frac{\sum LU}{H}$$

$$F4 = \frac{\sum D}{H} \qquad F8 = \frac{\sum RD}{H}$$

Where *'H'* is the Height of the image frame; used to normalize the results. A demonstration of how concavity features work can be seen in Figure 6 where some samples were tested against various concavity features.

B. Distribution features

While extracting the Distribution features in this approach, the vertical sliding window shifts across the image. Isolating each frame into equally sized cells. All the computed features are based on the distribution of the foreground pixels '1' and background '0' in the word image along with their densities across each isolated frame.

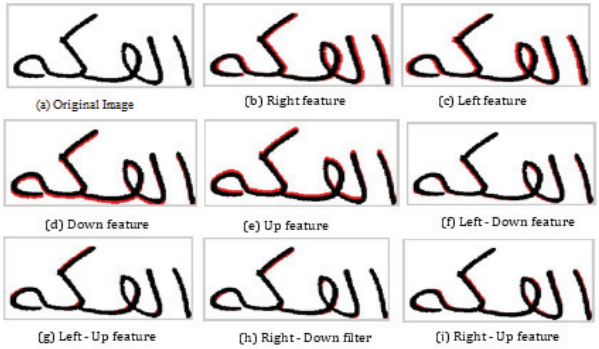

Figure 6: Effect of Concavity Features (Red lines represent the features)

Consider the following parameters and variables [12],[13]:

- *H:* Height of a frame, *h:* Fixed height of a cell

- *w:* Sliding window width
- n_c: No of cells in each isolated frame ($n_c = H/h$)
- *r(j):* No of foreground pixels in row *j* of a frame
- *n(i):* No of foreground pixels in cell *i, where b(i):* Density level of cell *i, b(i)=1 if if n(i)!=0 else b(i)=0*

The distribution features based on [12],[13] are calculated as follows:

- Density of foreground pixels in a frame

$$f1 = \sum_{i=1}^{nc} n(i)$$

- Number of background /foreground transitions between two neighboring frame cells

$$f2 = \sum_{i=2}^{nc} | b(i) - b(i-1) |$$

- The derivative feature representing the difference in vertical position of center of gravity *'g'* of foreground pixels in the current frame (t) and in the previous one (t-1).

$$f3 = g(t) - g(t-1)$$

Where position *'g'* is computed as

$$g = \frac{\sum_{j=1}^{H} j.r(j)}{\sum_{j=1}^{H} r(j)}$$

- The next number of features depends on the number of pixel columns per frame. So, from f4 and up, they represent the densities of foreground pixels in each column of each frame.

A sample of how distribution features are affecting the feature vector can be seen in Figure 7.

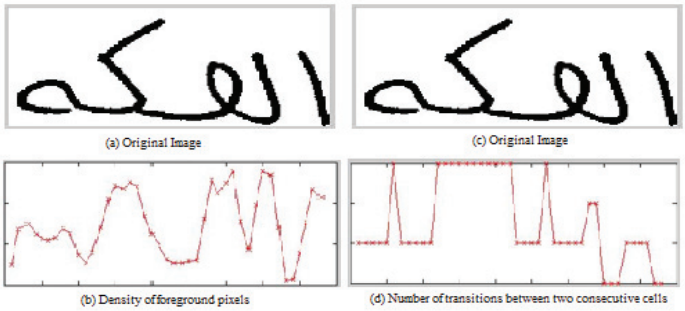

Figure 7: Sample of Distribution Features

V. Post-Recognition Lexicon Matching

This Section will focus on the algorithm of Post-Recognition Matching. How it works, why it was introduced, and how it will be used in our proposed system.

As the name implies, this algorithm will be applied post the HMM recognition phase to further improve the recognition of the complete words. Also returning the recognized primitive labels into their original form or name. So as stated, this algorithm has two main functionalities. The first is where recognized primitives are matched with words from the Lexicon to generate a complete word. This functionality will be called 'Primitive matching'. The second feature of the algorithm is to convert the primitive letters into their original form and further improve the recognition rate of the words without changing the letters recognized by HMM. This will be called (Original Letter Matching). The two functionalities and their complete description on how they operate will be discussed in the following sub-sections:

A. Primitive Matching

In this part of the Post-Recognition Lexicon Matching algorithm the main concern is matching the primitive letters output from the HMM with their entries in the lexicon (also in the primitive form). This is where the Lexicon comes in to the scene. But matching cannot be done unless there is a valid lexicon entry with this letter in this current position (start, end, middle, and isolated). Consider the following example: The sequence of primitives was the output from HMM recognition phase 'saad_start', 'nabra_mid', 'nabra_mid' and 'hah_end'. The Lexicon used contains the labels of the words in Figure 8a. In Figure 8b, the same lexicon words but in their primitive form. The algorithm of primitive matching will start with the first letter recognized by HMM (Primitive form) and start to match it with all Lexicon entries (also in primitive form). The lexicon- or what is called dictionary- will then be reduced to the words having the same first letter as the one from the HMM recognition as in Figure 8c. Then, it will keep on matching the rest of letters and reduce the dictionary entries as in Figure 8d until it finds a complete match shown in Figure 8e or otherwise will fail. The failure may be due to various reasons, the main one might be that the letter was not correctly recognized by the HMM, so its entry will not be found in the dictionary, other reasons might be that the word is not even in the lexicon to start with.

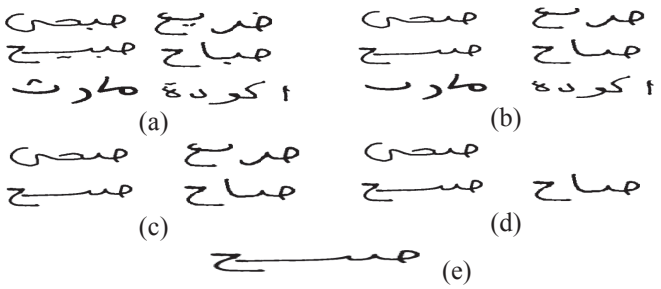

Figure 8: (a) Lexicon entries, (b) Primitive shape of Words in Lexicon, (c) After matching the first letter, (d) After matching the first and second letter, (e) After matching the first three letters

B. Original Letter Matching

This part of the algorithm is mainly focused on converting the recognized primitives into their original form of Arabic letter with diacritics as shown in Figure 9. Also, this part of the Post-Recognition Lexicon Matching has a big role in further

improving the recognition rate of the system. This occurs on two different stages; the first is the conversion of the letters to primitives which was already done in previous stages of the system. This step insures that diacritics have no effect on either training or recognition with HMM. The second stage is further grouping of similar-looking letters, whether they have the same primitive or not. This stage was done due to the observation of the output from the system and realizing that some recognition errors occur due to letters having almost the same shape but differ in minor parts. This occurs due to the characteristics of handwriting and mistakes in the writing style. An example of the primitive groups is shown in Figure 10. These are three different letters the first one is 'meem_mid', the second is 'feh_mid', and the third is 'ain_mid'. The three letters can barely be distinguishable that's why they were grouped into what is called primitive group. Another example of this is 'feh', 'qaf' and 'waw' letters that sometimes are nearly identical to each other when written by hand.

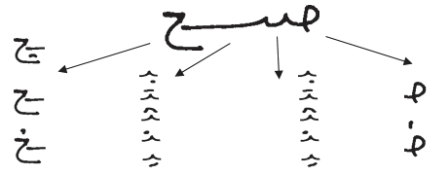

Figure 9: Original letters of the primitive word

Figure 10: Two Images of Three different letters looking like each other

VI. Experimental Results

In this Section, the results of the proposed engine vs. other engines developed by other researchers will be shown. This Section will also discuss the effect of using Post-Recognition Lexicon Matching stage with our system and how it compares with the results of not using it, along with the analysis of the recognition results. Also, the effect of various HMM parameters will be shown.

As stated previously in Section 4, two kinds of features were used. Concavity features provided a vector of length 8, and distribution features have a dynamic vector size. Having 3 static features, and the rest are based on the window size and overlap used. In the current system, the total distribution feature vector size is 13. So overall, the total length of feature vector is 21.

A. Effect of HMM parameters

In this sub section, the effect of various HMM parameters variations on the whole system and how these parameters affect the recognition rate of the whole system will be shown.

Table 2 shows the effect of changing number of mixtures per character on the whole system.

In Table 3, the effect of number of states variation per character on the whole system is shown. This resulted in selecting the number of HMM states to be 5 and number of Gaussian mixtures to be 4 for this system.

TABLE 2 EFFECT OF NUMBER OF GAUSSIAN MIXTURES PER STATE

Number of Mixtures	Training sets	Testing sets	Avg. recognition rate
4-6	a,b,c	d,e	87%
7-10	a,b,c	d,e	64%
11-14	a,b,c	d,e	30%

TABLE 3 EFFECT OF NUMBER OF HMM STATES

Number of States	Training sets	Testing sets	Avg. recognition rate
5	a,b,c	d,e	87%
7	a,b,c	d,e	86%
9	a,b,c	d,e	79%

B. Overall System Performance

In Table 4 and Table 5 the average recognition rate of segmented characters across dataset 'd' and 'e' can be seen when varying the number of Gaussian Mixtures and number of HMM states.

TABLE 4 EFFECT OF NUMBER OF GAUSSIAN MIXTURES ON SEGMENTED CHARACTERS RECOGNITION RATE

Number of Gaussian Mixtures	Training sets	Testing sets	Avg. recognition rate
4-6	a,b,c	d,e	96%
7-10	a,b,c	d,e	86%
11-14	a,b,c	d,e	60%

TABLE 5 EFFECT OF NUMBER OF NUMBER OF STATES ON SEGMENTED CHARACTERS RECOGNITION RATE

Number of States	Training sets	Testing sets	Avg. recognition rate
5	a,b,c	d,e	96%
7	a,b,c	d,e	95%
9	a,b,c	d,e	91%

In Table 6, the effect of enabling the Post-Recognition Lexicon Matching feature to enhance the recognition rate vs. having the results directly from the HTK toolbox can be seen. Note that these results illustrate the recognition rate of a complete word rather than a single letter.

TABLE 6 EFFECT OF POST-RECOGNITION LEXICON MATCHING

Post-Recognition Lexicon Matching	Recognition Rate
Enabled	***87%***
Disabled	***78%***

C. System Comparison

Table 7 shows the results from previous work done by other researchers using the same feature categories but different length of feature vector, and different method of applying the features. Different feature vector size implies change in the amount of data provided to the HMM in both training and recognition stages. This indicates the effectiveness of the proposed system along with the used features against other systems.

TABLE 7 AVERAGE RECOGNITION RATES OF PREVIOUS WORK

System	Avg. recognition rate
Al-Hajj et al.[12]	87.2%
Azeem et al.[13]	93.0%
Pechwitz et al.[14]	89.74%
Al-Hajj et al.[6]	90.26%
Xiang et al.[15]	84.15%
AlKhateeb et al.[16]	83.55%
Hamdani et al.[17]	82%
Proposed system	87%

VII. CONCLUSION

In this paper. An offline Arabic handwriting recognition system was proposed. New techniques regarding the pre-processing of the images; where diacritics are removed from the images to provide a clearer view for the HMM. Another technique was used to separate words into segmented letters for the training phase. Another contribution was provided in the means of the Post-Recognition Lexicon Matching algorithm, where the recognition rates are further improved by using lexicon matching along with primitive groups matching. Causing this system to reach very promising results compared to previous systems.

REFERENCES

[1] I. M. Elanwar, "Arabic Handwritten Script Recognition Towards Generalization: A Survey," *7th Conf. Eng. Lang. Ain Shams Univ. Cairo, Egypt*, pp. 1–26, 2007.

[2] M. T. Parvez and S. a. Mahmoud, "Offline arabic handwritten text recognition: A Survey," *ACM Comput. Surv.*, vol. 45, no. 2, pp. 23:1–23:35, 2013.

[3] R. Plamondon and S. N. Srihari, "On-Line and Off-Line Handwriting Recognition : A Comprehensive Survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 1, pp. 63–84, 2000.

[4] T. Plötz and G. a. Fink, "Markov models for offline handwriting recognition: A survey," *Int. J. Doc. Anal. Recognit.*, vol. 12, no. 4, pp. 269–298, 2009.

[5] H. El Abed and V. Märgner, "The IFN/ENIT-database - A tool to develop Arabic handwriting recognition systems," in *2007 9th International Symposium on Signal Processing and its Applications, ISSPA 2007, Proceedings*, 2007.

[6] R. Al-Hajj Mohamad, L. Likforman-Sulem, and C. Mokbel, "Combining slanted-frame classifiers for improved HMM-based arabic handwriting recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 7, pp. 1165–1177, 2009.

[7] M. Pechwitz and V. Maergner, "HMM based approach for handwritten arabic word recognition using the IFN/ENIT - database," *Seventh Int. Conf. Doc. Anal. Recognition, 2003. Proceedings.*, no. Icdar, 2003.

[8] "HTK Speech Recognition Toolkit." [Online]. Available: http://htk.eng.cam.ac.uk/.

[9] H. El Abed and V. Märgner, "Comparison of different preprocessing and feature extraction methods for offline recognition of handwritten Arabic words," *Proc. Int. Conf. Doc. Anal. Recognition, ICDAR*, vol. 2, pp. 974–978, 2007.

[10] H. Boukerma and N. Farah, "Preprocessing Algorithms for Arabic Handwriting Recognition Systems," in *2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, 2012, pp. 318–323.

[11] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77. pp. 257–286, 1989.

[12] R. El-Hajj, L. Likforman-Sulem, and C. Mokbel, "Arabic handwriting recognition using baseline dependant features and hidden Markov modeling," *Proc. Int. Conf. Doc. Anal. Recognition, ICDAR*, vol. 2005, pp. 893–897, 2005.

[13] S. A. Azeem and H. Ahmed, "Effective technique for the recognition of offline Arabic handwritten words using hidden Markov models," *Int. J. Doc. Anal. Recognit.*, vol. 16, no. 4, pp. 399–412, 2013.

[14] S. Alma'adeed, C. Higgins, and D. Elliman, "Off-line recognition of handwritten Arabic words using multiple hidden Markov models," in *Knowledge-Based Systems*, 2004, vol. 17, no. 2–4, pp. 75–79.

[15] A. Benouareth, A. Ennaji, and M. Sellami, "HMMs with Explicit State Duration Applied to Handwritten Arabic Word Recognition," in *18th International Conference on Pattern Recognition (ICPR'06)*, 2006, pp. 897–900.

[16] J. H. Alkhateeb, J. Ren, J. Jiang, and H. Al-Muhtaseb, "Offline handwritten Arabic cursive text recognition using Hidden Markov Models and re-ranking," *Pattern Recognit. Lett.*, vol. 32, no. 8, pp. 1081–1088, 2011.

[17] M. Hamdani, H. El Abed, M. Kherallah, and A. M. Alimi, "Combining Multiple HMMs Using On-line and Off-line Features for Off-line Arabic Handwriting Recognition," in *2009 10th International Conference on Document Analysis and Recognition*, 2009, pp. 201–205.