IJDAR manuscript No.
(will be inserted by the editor)

Recognizing handwritten Arabic words using grapheme segmentation and recurrent neural networks

Gheith Abandah · Fuad Jamour · Esam Qaralleh

Received: May 12, 2013 / Accepted: January 22, 2014

Abstract The Arabic alphabet is used in around 27 languages, including Arabic, Persian, Kurdish, Urdu, and Jawi. Many researchers have developed systems for recognizing cursive handwritten Arabic words, using both holistic and segmentation-based approaches. This paper introduces a system that achieves high accuracy using efficient segmentation, feature extraction, and recurrent neural network (RNN). We describe a robust rule-based segmentation algorithm that uses special feature points identified in the word skeleton to segment the cursive words into graphemes. We show that careful selection from a wide-range of features extracted during and after the segmentation stage produces a feature set that significantly reduces the label error. We demonstrate that using same RNN recognition engine, the segmentation approach with efficient feature extraction gives better results than a holistic approach that extracts features from raw pixels. We evaluated this segmentation approach against an improved version of the holistic system MDLSTM that won the ICDAR 2009 Arabic handwritten word recognition competition. On the IfN/ENIT database of handwritten Arabic words, the segmentation approach reduces the average label error by 18.5%, the sequence error by 22.3%, and the execution time by 31%, relative to MDLSTM. This approach also has the best published accuracies on two IfN/ENIT test sets.

G. Abandah
Computer Engineering Department, The University of Jordan, Amman, 11942, Jordan
Tel.: +962-6-5355000
Fax: +962-6-5300813
E-mail: abandah@ju.edu.jo

F. Jamour
King Abdullah University of Science and Technology
E-mail: fjamour@gmail.com

E. Qaralleh
Princess Sumaya University for Technology
E-mail: qaralleh@psut.edu.jo

Keywords Optical character recognition · Handwritten Arabic words · Segmentation · Feature evaluation and selection · Recurrent neural networks

1 Introduction

Offline cursive handwriting recognition is a hard problem for which no satisfactory general solutions are yet available. Major challenges include the overlap and interconnection of neighboring characters, the huge variability in both quality and style of human handwriting, and the similarities of distinct character shapes [9, 33].

Arabic is the native language of more than 440 million people and its alphabet is used in around 27 languages, including Arabic, Persian, Kurdish, Urdu, and Jawi [34]. Arabic is always cursive in print and in handwriting. Despite decades of research, there is still a lack of accurate Arabic handwriting recognition systems [44].

Arabic handwritten script recognizers can be divided into *holistic* and *segmentation* approaches. Holistic approaches process the words as a whole without segmentation into smaller components. Segmentation approaches first segment the words into characters or strokes, then pass the segments to the recognizer [38].

Developing effective segmentation algorithms for cursive script is difficult and requires considerable expert language knowledge. However such algorithms generally lead to a substantial gain in recognition accuracy, since they provide richer feature extraction and allow the recognition of open vocabulary [35].

Holistic approaches have so far proved more successful in limited-vocabulary handwriting recognition [44]. These approaches build on advances in other areas such as speech recognition and recurrent neural networks (RNNs) [28, 14].

This paper presents an efficient system for recognizing handwritten Arabic words that combines efficient segmentation, feature selection, and recurrent neural networks to achieve state-of-the-art accuracy. The main contributions are as follows:

- A novel rule-based segmentation algorithm that segments cursive words into graphemes by collecting special feature points from the word skeleton.
- A selection of an efficient subset of features for recognizing handwritten Arabic words through an evaluation of wide-range of feature extraction techniques.
- A demonstration that the segmentation approach with efficient feature extraction gives better label and sequence error rates than a holistic approach that extracts features from the raw pixels, using the same recurrent neural network recognition engine.

This paper is organized as follows: The rest of this section reviews the Arabic writing system, gives an overview of related work in Arabic handwriting recognition, and provides an overview of our system's processing stages. Section 2 describes the sub-word and grapheme segmentation stages. Section 3 describes the feature extraction stage and the algorithm used in selecting features. Section 4 describes the RNN used in the character sequence recognition stage. Section 5 describes the algorithms used in the word matching stage. Section 6 presents the experimental results, while Section 7 compares our system with the best alternatives, including a detailed comparison with the record holder for the ICDAR offline Arabic handwriting recognition competitions [44]. Finally, Section 8 provides conclusions and plans for future work.

1.1 Overview of Arabic writing

Arabic is written from right to left and is always cursive. The Arabic alphabet has 28 basic letters [3]. Each letter has multiple forms depending on its position in the word. Each letter is drawn in an isolated form when it is written alone, and is drawn in up to three other forms when it is written connected to other letters in the word. For example, the letter **Ain** has four forms: *isolated* (ع), *initial* (عـ), *medial* (ـعـ), and *final* (ـع).

Within a word, every letter can connect from the right with the previous letter. The horizontal level where letters are connected is the *baseline*. There are six letters that do not connect from the left with the next letter. These letters only have the isolated and final forms. When one of these six letters is present in a word, the word is broken into *sub-words*, often called *parts of Arabic words* (PAWs). For example, the word "Arabic" (عربية) has two PAWs: the first PAW consists of the initial **Ain** (عـ) and final, left-disconnecting, **Reh** (ر); and the second PAW consists of the initial **Beh** (بـ), medial **Yeh** (ـيـ), and final **Teh Marbuta** (ـة).

1.2 Related work

Many researchers have developed algorithms to segment handwritten Arabic words into smaller components [6,45, 55,37,59,32]. A review of several relevant segmentation algorithms is available in [1]. Segmentation-based papers generally report the segmentation accuracy and do not report overall recognition accuracy or report accuracies that are lower than the holistic systems.

Wshah *et al.* use the skeleton and the boundary of the word to do over/under segmentation [59]. The output of the algorithm is graphemes that represent anything between one fifth of a letter and three connected letters. They report that their algorithm segments 93% of the test samples into segments that each has at most one letters.

Kundu *et al.* developed a complete system for the recognition of unconstrained handwritten Arabic words using over-segmentation of characters and variable duration hidden Markov model [32]. Their segmentation algorithm translates the 2-D image into 1-D sequence of sub-character symbols. They report recognition accuracies of only 66% and 60% on IfN/ENIT database test sets *d* and *e* [48], respectively.

Furthermore, many researchers have experimented with a wide range of feature extraction techniques to recognize handwritten Arabic text. Some examples can be found in [8, 54,16,38,15]. Feature selection is often used for excluding irrelevant and redundant features. When successful, it can greatly reduce system complexity and processing time as well as improving recognition accuracy [29].

Märgner, El Abed, and Pechwitz have organized a series of Arabic handwriting recognition competitions [40–44]. The purpose of these competitions is to advance the research and development of Arabic handwritten word recognition systems. These competitions use the IfN/ENIT database and have had excellent participations from the research leaders in the area. The participations have shown remarkable progress over seven years. In the following paragraphs, we review a selection of systems that have participated in these competitions and achieved top results. Moreover, the performance of these systems is summarized in Section 7.1.

The UOB system developed by Al-Hajj *et al.* won the first competition in ICDAR 2005. This system estimates the word baseline and extracts robust language independent features using a sliding window, without character segmentation [5]. The system uses a pure HMM recognizer originally developed for speech recognition.

The competitions organizers also participated with their ARAB-IFN system in ICDAR 2005. This system got the second place and is described in [47]. The authors performed a thorough analysis of the IfN/ENIT database. They experimented with advanced techniques to estimate the word baseline and top line and used these estimations to normalize the word. The normalization goal is to make the feature extraction more robust against size, slant, skew, and line width variations of a word. They used two feature extraction methods that have similar performance. The first method extracts pixel features using a sliding window with three columns and the second method extracts skeleton direction features in five zones using overlapping frames.

The Siemens system submitted by Alary *et al.* was the winner of the ICDAR 2007 competition. This system was adapted for Arabic script from the standard HMM-based Latin script word recognizer that is widely in use within Siemens AG for postal automation projects [56].

The MDLSTM system developed by Graves was the winner of the ICDAR 2009 competition. This system uses a hierarchy of multidimensional recurrent neural networks [28]. It is further described in Section 7.2. This system holds the record for recognition accuracy on the main competition test set, outperforming the winners of both ICFHR 2010 and ICDAR 2011 competitions.

The UPV-PRHLT system developed by Alkhoury *et al.* was the winner of the ICFHR 2010 competition. This system uses windows of raw, binary image pixels, which are directly fed into embedded Bernoulli HMMs [7]. The authors found that best results are obtained with a nine-column window and when the extracted window is repositioned to align its center of mass with the window center.

The RWTH-OCR system developed by Dreuw *et al.* was the winner of the ICDAR 2011 competition. This system uses sliding window for HMM-based handwriting recognition [14]. Discriminative training based on the maximum mutual information criterion is used to train writer independent handwriting models. Writer adaptation is used to improve accuracy through an unsupervised confidence-based discriminative training on a word and frame level within a two-pass decoding process.

The majority of these recognition engines use HMMs to build letter, PAW, or word models. The engines that build PAW or word models suffer from limitations in recognizing PAWs or words that are not represented in the training set; they are generally limited-vocabulary recognizers. Moreover, they generally rely on sliding windows to extract features from the word image. Such windows limit the variety of features that can be extracted. The size of these windows often requires tuning according to the writing style, which significantly affects the recognition accuracy as reported in [7].

Our system segments words to elementary graphemes to eliminate the problems associated with building PAW and word models; thus it may support open vocabulary. It also eliminates the need to tune the feature extractor for different writing styles without sacrificing the recognition accuracy. The use of an RNN with bi-directional long short-term memory (BLSTM) enables high letter recognition rates. This accuracy is possible even with difficult letters because of BLSTM's ability to exploit context information. The use of connectionist temporal classification (CTC) output layer facilitates training without the need to manually segment and annotate the letters in each word.

1.3 System overview

Figure 1 summarizes the five main stages of our approach to recognizing handwritten Arabic words: (*i*) sub-word segmentation, (*ii*) grapheme segmentation, (*iii*) feature extraction, (*iv*) sequence transcription, and (*v*) word matching. Unlike most of the available recognition systems, the sequence transcription stage does not use a language model. Thus, word matching in our system is optional, and words can be added or removed from the dictionary to enhance the word recognition accuracy without the need to re-train the system. An earlier version of this approach, based on modifying the OCR system 'Tesseract' [58], is described in Ref. [1]. Tesseract was chosen then because it is an accurate and open-source system.

In this paper we only consider isolated word recognition. When the system is used to recognize complete documents, line and word segmentation must be added as preprocessing stages.

We now describe each of the five recognition stages in detail.

2 Segmentation

The input word is segmented into graphemes in two stages; the word is first segmented into sub-words, then the sub-words are segmented into graphemes.

2.1 Sub-word segmentation

In this stage the baseline is estimated first; then the main and secondary bodies are identified, the main bodies of the sub-words are extracted, and the secondary bodies are assigned to their respective main bodies to yield the sub-words.

We estimate the *word baseline* using the horizontal projection histogram method [39]. The row that contains the maximum number of black pixels is the baseline. This simple method is sufficient for our purposes. We only use the baseline estimation in recognizing secondary bodies and extracting some configuration features, as described below.

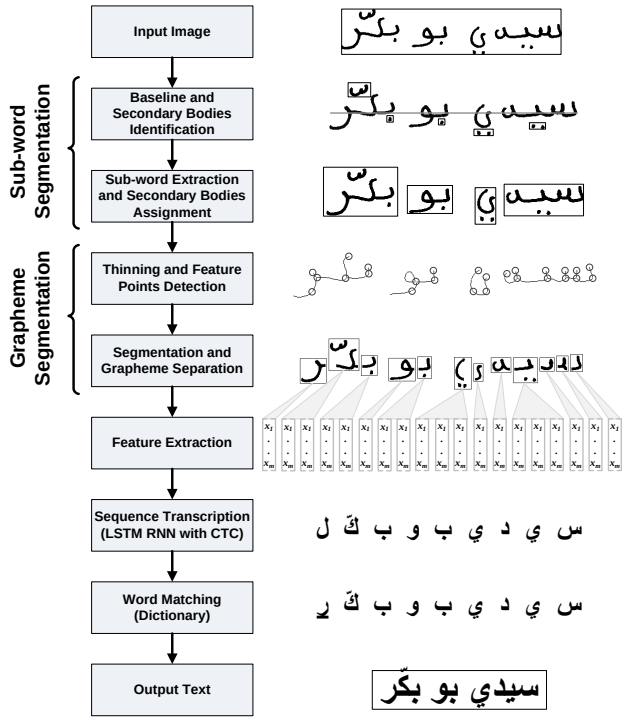


Fig. 1 Processing stages of our Arabic handwriting recognition system

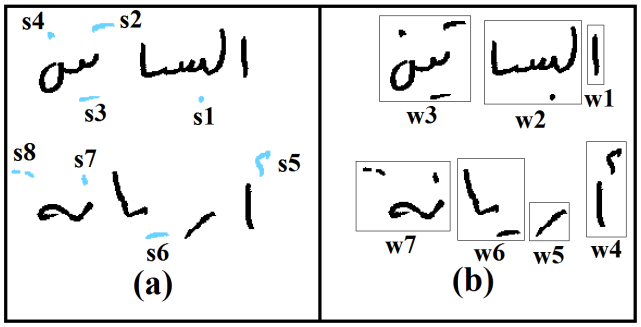


Fig. 2 (a) Secondary bodies identification, (b) sub-words extraction

We use a contour-based connected components extraction algorithm to identify the components of the image and its secondary bodies [11]. A component is classified as a *secondary body* when one of the following conditions applies: (*i*) it is very small compared to other components in the same image, (*ii*) it is relatively small and far from the baseline, or (*iii*) it is a vertical line with a relatively large component below it. The light bodies in Fig. 2(a) are examples of identified secondary bodies and are labeled s1 through s8. Furthermore, secondary bodies that are close to each other and are similar in size are considered one *secondary body group*, *e.g.*, s8. As explained below, this grouping is important in grapheme separation.

Components that are not secondary bodies are the *main bodies* of the sub-words. Every main body is extracted with its secondary bodies as one sub-word and is passed to the next stage. Secondary bodies are examined from right to left. For every secondary body, the algorithm assigns it to a main body according the the following rules (first that applies): (*i*) the main body that is above its midpoint, *e.g.*, s1, (*ii*) below its midpoint, *e.g.*, s4, (*iii*) above its left endpoint, *e.g.*, s6, (*iv*) below its left endpoint, *e.g.*, s2, (*v*) to its right, *e.g.*, s8, or (*vi*) the rightmost main body in the word, *e.g.*, s5. In Fig. 2(b), the upper word contains three sub-words (labeled w1 through w3), and the lower word contains four sub-words (labeled w4 through w7). Additionally, any overlap between adjacent sub-words is eliminated by separating them horizontally.

2.2 Grapheme segmentation

We now describe the two steps of grapheme segmentation: feature points detection and grapheme separation.

2.2.1 Feature point detection

In this step, morphological features are detected in the skeleton of the sub-words. The secondary bodies are temporarily removed before using Deutsch's thinning algorithm to get the skeleton of the sub-word's main body [12]. This algorithm is robust and works well with handwritten Arabic script. However, similar to other algorithms, it sometimes removes fine features such as **Seen** teeth (س) [3].

The feature points detected are *end points*, *branch points*, and *cross points*. They are detected by examining the eight neighbors of every skeleton pixel. An end point has one black neighbor, a branch point has three, and a cross point has four.

Our algorithm identifies continuities to detect edge points. A *continuity* is a continuous string of black pixels in the skeleton that connects two feature points. If the starting point is the ending point, then the continuity is a loop.

The *edge points* are points where the direction of the continuity changes and are detected using the polygonal approximation of the skeleton. The algorithm used to find the polygonal approximation is due to Douglas and Peucker [13].

Each vertex in the polygon that is not an end point, branch point, or cross point is considered an edge point. For each edge point the *bisector angle* is found. This is the angle between the bisector line of the edge and the horizontal line. For more details and examples, refer to Ref. [1]. Figure 3 shows example three sub-words with their feature points identified. The edge point marked with BA $\sim 0°$ is an example edge point with a horizontal bisector.

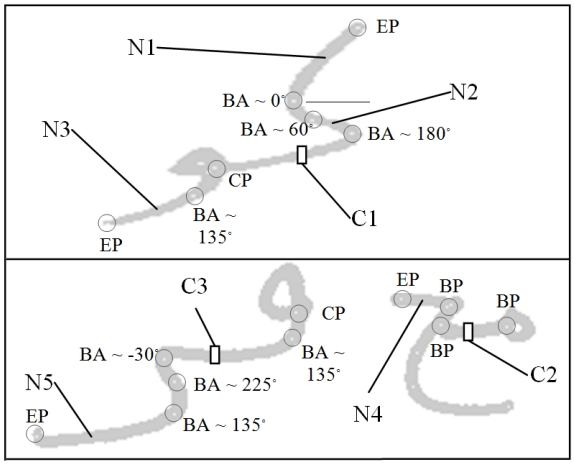


Fig. 3 Grapheme segmentation examples. EP: End point, BP: Branch point, CP: Cross point, BA: Bisector angle of an edge point

Once the feature points are detected, a continuity that contains them is split into as many child continuities as needed so that each new continuity connects two feature points and has no feature points inside it. This yields a list of continuities that each has two ends. The orientation of the continuity and the attributes of its ends are used to find the segmentation points, as explained in the following subsection.

2.2.2 Grapheme separation

Continuities that have the following properties are segmented to split the sub-word into graphemes. Figure 3 shows three continuities that are segmented at the shown points and marked C1, C2, and C3. Other example continuities that are not segmented are marked N1 through N5.

1. Low slope: the orientation of the continuity should be between $-45°$ and $+45°$. N1 is not segmented because its orientation is larger than 45°.
2. If the right end is an edge, its bisector angle should be between 45° and 225° as in C1 and C3.
3. The left end is not an end point. N3 and N4 are not segmented because they violate this property.
4. If the left end is an edge, its bisector angle should be between $-155°$ and 65° as in C3.
5. It is not totally covered from above or from below. This property avoids segmenting loops and some continuities such as N2.

The angle ranges above are selected to cover the various cases of Arabic sub-word configurations. For example, the sub-word (ﺒﺴ) contains two letters: the right letter is initial **Noon** (ﺴ) that has an edge with bisector angle of 135°, and the left letter is final **Alef** (ﺎ) that has an edge with bisector angle of 45°. The horizontal continuity connecting the two edge points in this sub-word satisfies the five properties above and is segmented.

Each continuity that should be segmented is then examined to find the best cut point in it. The cut point is searched for starting from the first left quarter of the continuity. The first point in a horizontal small segment of the continuity is the cut point. If the cut point happens to cut a letter (by inspecting the stroke width at this point), the cut point is shifted to the point where the stroke width is minimum. If the continuity has no horizontal segments, its midpoint is selected as the cut point. This method enhances the accuracy of secondary body assignment.

Finally, the secondary bodies are reassigned to the segmented graphemes using the assignment method described in Subsection 2.1. The entire secondary body group is assigned to one grapheme because such group belongs to one letter. The segmented main bodies and their respective secondary bodies are then passed to the feature extraction stage.

3 Feature Extraction and Selection

As the above segmentation algorithm identifies sub-words, secondary and main bodies, and feature points, it collects features of the identified and segmented bodies.

3.1 Feature extraction

Additional features are extracted in the feature extraction stage. We started with extracting a total of 103 features that are listed in Table 1. These features can be grouped into six classes: statistical, configuration, skeleton, boundary, elliptic Fourier descriptor, and directional features and are described in the following paragraphs. More details about the extraction of these features can be found in Refs. [3,4].

3.1.1 Statistical features

These 15 features are extracted from the object's binary image. The area is calculated from the binary image using $A = \sum_x \sum_y B(x,y)$. The object's width W and height H are also used to derive the scale invariant feature: width to height ratio W/H. The fraction of black pixels in each of the four quadrants are UR/A, UL/A, LL/A, and LR/A.

The object's center of mass $(\bar{x},\bar{y})$ is used in computing the normalized central moments of order $(u+v)$ by $\eta_{u,v} =$

Table 1 List of extracted features

	No	Feature	Description
Statistical Features	1	A	Area
	2	W	Width
	3	H	Height
	4	W/H	Width to height ratio
	5	UR/A	Upper right quadrant pixels fraction
	6	UL/A	Upper left quadrant pixels fraction
	7	LL/A	Lower left quadrant pixels fraction
	8	LR/A	Lower right quadrant pixels fraction
	9	$\bar{x}$	Center of mass: x coordinate
	10	$\bar{y}$	Center of mass: y coordinate
	11	$\eta_{2,0}$	Normalized central moment
	12	$\eta_{0,2}$	Normalized central moment
	13	$\bar{x}_N$	Normalized COM: x coordinate
	14	$\bar{y}_N$	Normalized COM: y coordinate
	15	θ	Orientation angle
Configuration Features	16	U/A	Fraction of pixels above the baseline
	17	$D_{\bar{y}}$	Center of mass to baseline distance
	18	D_{Top}	Top pixel to baseline distance
	19	*Loops*	The number of closed loops
	20	*Form*	The position in the sub-word; isolated, initial, medial, or final
	21	*Is_Sec*	Used to distinguish secondary bodies from main bodies
	22	S	No. of secondary bodies for a main body ($S = S_a + S_b$)
	23	S_a	No. of secondary bodies above
	24	S_b	No. of secondary bodies below
	25	*Sec_Conf*	Configuration of secondary bodies
Skeleton	26	*Branchs*	No. of branch points in the skeleton
	27	*Ends*	No. of end points in the skeleton
	28	E_1	Edge point feature 1
	29	E_2	Edge point feature 2
Boundary	30	m	The number of boundary pixels
	31	T	Perimeter Length
	32	$T/2D$	Perimeter to diagonal ratio
	33	γ	Compactness ratio
	34–59	a_n, b_n, c_n, d_n	26 Elliptic Fourier Descriptors, $n = 0, 1, \ldots, 6$
	60–103	$D^{R,C}_{r,c,d}$	44 Directional Features

$\frac{1}{A^k}\sum_x\sum_y(x-\bar{x})^u(y-\bar{y})^v B(x,y)$ where $k = 1+(u+v)/2$. In this work, we only generated the top two most relevant moments: $\eta_{2,0}$ and $\eta_{0,2}$ [4].

The normalized center of mass $(\bar{x}_N, \bar{y}_N)$ is calculated using $\bar{x}_N = \frac{\bar{x}-(W-1)/2}{W/2}$ and $\bar{y}_N = \frac{\bar{y}-(H-1)/2}{H/2}$.

The orientation θ of an elongated object is the orientation of the elongation axis with respect to the horizon [3].

3.1.2 Configuration features

These 10 features are extracted from the configuration of the object and its surroundings. Relative to the word's baseline, we extract three features: the fraction of pixels above the baseline U/A, the distance from the center of mass to the baseline $D_{\bar{y}}$, and the distance from the top black pixel to the baseline D_{Top}.

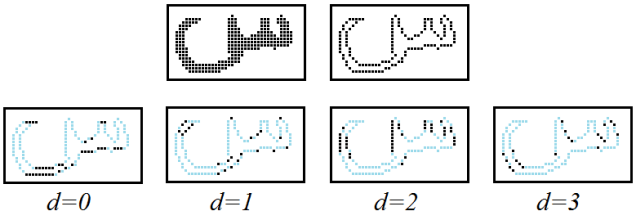


Fig. 4 Example handwritten letter, its boundary, and four images that contain boundary pixels (dark pixels) of direction codes $d = 0$, 1, 2, and 3, respectively.

The *Loops* feature is the number of closed loops in the object. The *Form* feature is a categorical feature of the position of the object in its sub-word.

The remaining five features are related to the object type and its secondaries. The binary feature *Is_Sec* is true when the object is a secondary object. For a main object, the number of its secondary objects is S. S_a and S_b specify for a main object the number of secondary bodies above it and below it, respectively. The secondary configuration *Sec_Conf* feature specifies one of four main object types: object without secondaries, with secondaries above it, below it, and with secondaries above and below it.

3.1.3 Skeleton features

These four features are extracted from the object's skeleton. *Branchs* is the number of branch points in the object's skeleton and *Ends* is the number of end points. The features E_1 and E_2 are edge points experimental features computed as the sum of moments of the bisection angles of the body's edge points using $E_1 = \sum_i x_i y_i |BA_i|$ and $E_2 = \sum_i x_i y_i BA_i$, respectively.

3.1.4 Boundary features

These four features are extracted from the object's boundary. Figure 4 shows an example handwritten letter and its boundary. For the outer boundary pixels $(x(t), y(t)), t = 1, 2, \ldots, m$, the number of boundary pixels is m. The Freeman chain code is used to compactly encode the boundary pixels [17]. The direction from every boundary pixel to the next boundary pixel is put in the chain. The direction codes $f(t) \in \{0, 1, \ldots, 7\}$ are used such that right is 0, up-right is 1, up is 2, *etc*.

The perimeter length is $T = \sum_{t=1}^{m} L(f(t))$ where $L(f(t)) = 1$ for $f(t)$ even and $\sqrt{2}$ for $f(t)$ odd. The perimeter to diagonal ratio is $T/2D = (T/2)/\sqrt{W^2+H^2}$. The compactness ratio $\gamma = T^2/4\pi A$.

3.1.5 Elliptic Fourier descriptors

The outer boundary is a piece-wise linear closed curve and is used in extracting the elliptic Fourier descriptors (EFD) [31]. The four descriptors of order n are found by

$$a_n = \frac{T}{2n^2\pi^2}\sum_{i=1}^{m}\frac{\Delta x_i}{\Delta t_i}[\cos\phi_i - \cos\phi_{i-1}] \tag{1}$$

$$b_n = \frac{T}{2n^2\pi^2}\sum_{i=1}^{m}\frac{\Delta x_i}{\Delta t_i}[\sin\phi_i - \sin\phi_{i-1}] \tag{2}$$

$$c_n = \frac{T}{2n^2\pi^2}\sum_{i=1}^{m}\frac{\Delta y_i}{\Delta t_i}[\cos\phi_i - \cos\phi_{i-1}] \tag{3}$$

$$d_n = \frac{T}{2n^2\pi^2}\sum_{i=1}^{m}\frac{\Delta y_i}{\Delta t_i}[\sin\phi_i - \sin\phi_{i-1}] \tag{4}$$

where

$$\begin{aligned} \phi_i &= 2n\pi t_i/T, \qquad \Delta x_i = x(i) - x(i-1), \\ \Delta y_i &= y(i) - y(i-1), \quad \Delta t_i = \sqrt{\Delta x_i^2 + \Delta y_i^2}, \\ t_i &= \textstyle\sum_{j=1}^{i}\Delta t_j, \qquad T = t_m = \textstyle\sum_{j=1}^{m}\Delta t_j. \end{aligned} \tag{5}$$

We extract a set of 26 EFDs comprising a_0, c_0, and a_n, b_n, c_n, d_n for $n = 1, 2, \ldots, 6$.

3.1.6 Directional features

The directional features are extracted from the chain codes of the object's boundaries [36]. Only four of the eight directions are relevant as the last four directions are mirror images of the first four. The four directional features D_d for $d = 0, 1, 2, 3$ are defined as $D_d = \sum_t C_d(f(t))$ where $C_d(f(t)) = 1$ for $(f(t) \bmod 4) = d$ and 0 otherwise. Figure 4 shows four images for the boundary pixels of directions codes $d = 0$, 1, 2, and 3, respectively. In other words, the feature D_d is the number of pixels of direction code d.

To exploit the topological distribution of the directional features, the object is split into regions of R rows and C columns. Then the four directional features are found for each region separately to get features $D^{R,C}_{r,c,d}$. For an $R \times C$ split, there are $4RC$ directional features where $r = 0, 1, \ldots, R-1$, $c = 0, 1, \ldots, C-1$, and $d = 0, 1, 2, 3$. For example, the feature $D^{2,3}_{0,1,2}$ is the number of vertical pixels in Row 0, Column 1 of an image split into 2 rows and 3 columns.

We extracted features from three splits 1×1, 2×2, and 2×3, totaling $4 \times (1 \times 1 + 2 \times 2 + 2 \times 3) = 44$ features.

3.2 Feature selection

Feature selection is important in many pattern recognition problems for excluding irrelevant and redundant features. It typically decreases system complexity and processing time and often improves recognition accuracy [29].

We tested several feature selection techniques ranging in complexity from simply picking the best individual features to evolutionary optimization algorithms [4]. We concluded that the *minimal-redundancy-maximal-relevance* (mRMR) technique [50] offers the best compromise between accuracy and speed, and therefore we use this technique in this paper.

In the mRMR algorithm, the subset S of m best features is grown iteratively from the complete set of features X using forward search algorithm. The following criterion is used to add the $\mathbf{x}_j$ feature to the previous subset of $m-1$ features:

$$\underset{\mathbf{x}_j \in X - S_{m-1}}{\arg\max}\left[I(\mathbf{x}_j;\omega) - \frac{1}{m-1}\sum_{\mathbf{x}_i \in S_{m-1}} I(\mathbf{x}_j;\mathbf{x}_i)\right] \tag{6}$$

This algorithm maximizes the difference between the mutual information of the feature $\mathbf{x}_j$ and class ω (*relevance*) and the mean of the mutual information values between feature $\mathbf{x}_j$ and previously selected features $\mathbf{x}_i \in S_{m-1}$ (*redundancy*). Where the mutual information for two random variables x and y is found in terms of their probabilistic density functions $p(x)$, $p(y)$, and $p(x,y)$ as

$$I(x;y) = \iint p(x,y)\log\frac{p(x,y)}{p(x)p(y)}dxdy. \tag{7}$$

As described in Section 6.3, we use a feature subset of length $m = 30$. For each object i, the m selected features are assembled in a feature vector $x_{i1}, x_{i2}, \ldots, x_{im}$.

Using feature statistics from the training samples, the feature vector is normalized to zero mean and unit standard deviation using $\hat{x}_{ik} = (x_{ik} - \bar{x}_k)/\sigma_k$, for $k = 1, 2, \ldots, m$. The normalized vector is then passed to the sequence transcription stage.

4 Sequence Transcription

The sequence transcription stage maps from sequences of feature vectors to sequences of recognized characters. Our sequence transcription is carried out using a recurrent neural network (RNN) with the bi-directional long short-term memory architecture (BLSTM) [27]. The connectionist temporal classification (CTC) [23] output layer is used to determine a probability distribution over all possible character sequences, given a particular feature sequence. A list of the most probable output sequences are then selected and passed along to the final word matching stage of recognition.

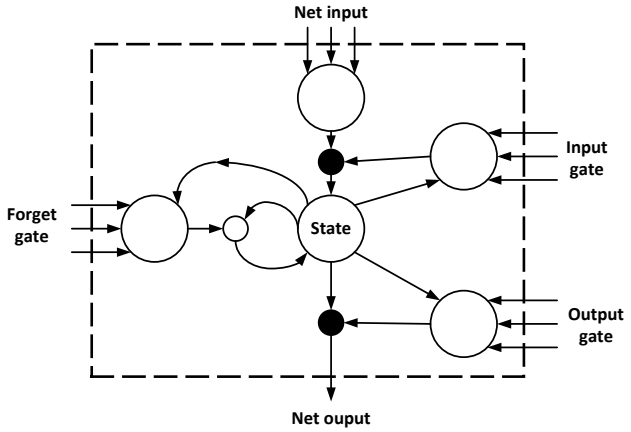


Fig. 5 LSTM block

The combination of BLSTM and CTC has been successfully applied to both online and offline handwriting recognition in the past [26]. Our experiments on BLSTM-CTC were carried out with the open source software library RNNLIB [19].

4.1 Bi-directional long short-term memory

Recurrent neural networks exploit the sequence context through cyclic connections in the hidden layer [20]. Given an input sequence $(x_1,\ldots,x_T)$, a traditional RNN computes the hidden vector sequence $(h_1,\ldots,h_T)$ and the output sequence $(y_1,\ldots,y_T)$ by iterating the following equations from $t=1$ to T:

$$h_t = \mathscr{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right) \tag{8}$$
$$y_t = \mathscr{Y}\left(W_{hy}h_t + b_y\right) \tag{9}$$

where W_{xh} is the input-hidden weight matrix, W_{hh} is the hidden-hidden weight matrix, W_{hy} is the hidden-output weight matrix, b_h and b_y are bias terms, $\mathscr{H}$ is the hidden layer function and $\mathscr{Y}$ is the output layer function.

In traditional RNNs $\mathscr{H}$ is usually element-wise application of the *tanh* or *logistic sigmoid* $\sigma(x) = 1/(1+\exp(-x))$ functions. However, the *long short-term memory* (LSTM) architecture is better at finding and exploiting long range context information [30,18]. The LSTM blocks replace the non-linear units in the hidden layer of traditional RNNs [30]. Figure 5 shows an LSTM block which consists of a core state cell and three gates. The input gate controls storing into the state cell and allows holding information for long periods of time. The forget gate affects the internal state and the output gate controls the output activation function.

For the version of LSTM used in this paper $\mathscr{H}$ is implemented by the following composite function:

$$\alpha_n = \sigma\left(W_{i\alpha}i_n + W_{h\alpha}h_{n-1} + W_{s\alpha}s_{n-1}\right) \tag{10}$$
$$\beta_n = \sigma\left(W_{i\beta}i_n + W_{h\beta}h_{n-1} + W_{s\beta}s_{n-1}\right) \tag{11}$$
$$s_n = \beta_n s_{n-1} + \alpha_n \tanh\left(W_{is}i_n + W_{hs}\right) \tag{12}$$
$$\gamma_n = \sigma\left(W_{i\gamma}i_n + W_{h\gamma}h_{n-1} + W_{s\gamma}s_n\right) \tag{13}$$
$$h_n = \gamma_n \tanh(s_n) \tag{14}$$

where α, β, γ and s are respectively the *input gate*, *forget gate*, *output gate* and *state* vectors, all of which are the same size as the hidden vector h. The weight matrix subscripts have the obvious meaning, for example $W_{h\alpha}$ is the hidden-input gate matrix, $W_{i\gamma}$ is the input-output gate matrix etc. The weight matrices from the state to gate vectors are diagonal, so element m in each gate vector only receives input from element m of the state vector. The bias terms (which are added to α, β, s and γ) have been omitted for clarity.

A *bi-directional RNN* [57] computes the *forward* hidden sequence $(\overrightarrow{h}_1,\ldots,\overrightarrow{h}_T)$, the *backward* hidden sequence $(\overleftarrow{h}_1,\ldots,\overleftarrow{h}_T)$, and the output sequence $(y_1,\ldots,y_T)$ by first iterating the backward layer from $t=T$ to 1:

$$\overleftarrow{h}_t = \mathscr{H}\left(W_{i\overleftarrow{h}}i_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right), \tag{15}$$

then iterating the forward and output layers from $t=1$ to T:

$$\overrightarrow{h}_t = \mathscr{H}\left(W_{i\overrightarrow{h}}i_t + W_{\overrightarrow{h}\overrightarrow{h}}\overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right) \tag{16}$$
$$y_t = \mathscr{Y}\left(W_{\overrightarrow{h}y}\overrightarrow{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_y\right) \tag{17}$$

The advantage of using bi-directional RNNs (BRNNs) is that the output layer is able to make use of both past and future context at each point along the sequence. Combing BRNNs with LSTM gives *bi-directional LSTM* (BLSTM; [27]), the RNN architecture used throughout this work.

4.2 Subsampling layers

Using multiple bi-directional RNN hidden layers can lead to improved performance, compared to a single-level architecture. However it can lead to a very large number of connection weights between the forward and backward layers in successive levels, which may increase overfitting as well as computational cost. One way to reduce the number of weights is to separate the levels with feedforward *subsampling layers*. Figure 6 shows one subsampling layer between the two hidden layers of a neural network.

The two layers in each level feed forward to a subsampling layer, which then feeds forward to the two layers in the next level up. The total number of inter-level weights can therefore be controlled by adjusting the sizes of the subsampling layers. For the experiments in this paper, all the

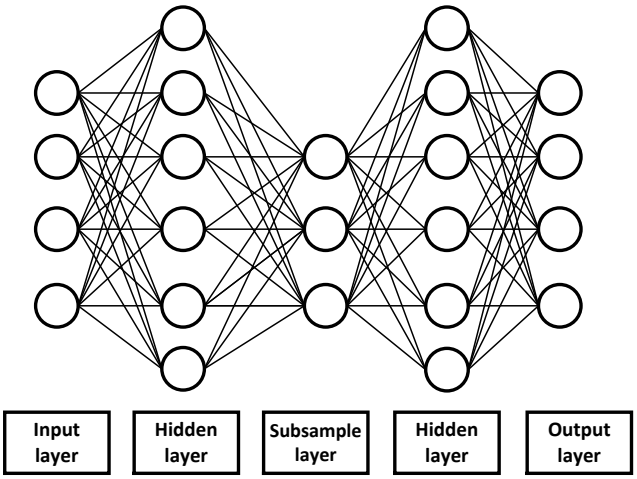


Fig. 6 Neural network topology with subsampling layer

subsampling layers consist of standard summation units with *tanh* activation functions.

The RNN topology (specifically the number of hidden layers, and the number of LSTM cells in each hidden layer) was optimized through experimentation as described in Section 6.4.

4.3 Connectionist temporal classification

Connectionist temporal classification (CTC) is an output layer designed for sequence transcription with RNNs [23]. It trains the network to predict a conditional probability distribution over all possible output transcriptions, or *labelings*, given the complete input sequence. It therefore does not require pre-segmented training data.

A CTC output layer contains one more unit than there are elements in the alphabet L of labels for the task. The first $|L|$ outputs estimate the probabilities of observing the corresponding labels at that time, and the extra output estimates the probability of observing a 'blank', or no label. For a length T input sequence $\mathbf{x}$, the complete sequence of CTC outputs therefore defines a probability distribution over the set L'^T of length T sequences over the alphabet $L' = L \cup \{blank\}$. We refer to the elements of L'^T as *paths*. Since the probabilities of the labels at each time step are conditionally independent given $\mathbf{x}$, the conditional probability of a path $\pi \in L'^T$ is given by

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} y^t_{\pi(t)}, \tag{18}$$

where y^t_k is the activation of output unit k at time t.

Paths are mapped onto labelings $\mathbf{l} \in \mathbf{L}^{\leq T}$ by an operator $\mathscr{B}$ that removes first the repeated labels, then the blanks. The conditional probability of some labeling $\mathbf{l} \in \mathbf{L}^{\leq T}$ is the sum of the probabilities of all paths corresponding to it:

$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi \in \mathscr{B}^{-1}(\mathbf{l})} p(\pi|\mathbf{x}). \tag{19}$$

This 'collapsing together' of different paths onto the same labeling is what allows CTC to use unsegmented data, because it means that the network only requires the order of the labels to define its distribution, and not their alignment with the input sequence.

For a training set S, consisting of pairs of input and target sequences $(\mathbf{x}, \mathbf{z})$, the CTC objective function $\mathscr{O}$ is the negative log probability of the network correctly labeling all of S:

$$\mathscr{O} = -\sum_{(\mathbf{x},\mathbf{z}) \in S} \log p(\mathbf{z}|\mathbf{x}). \tag{20}$$

This function can be efficiently differentiated with respect to the network outputs using the CTC forward-backward algorithm [23]. Back propagation through time [53] can then be used to find the derivatives of $\mathscr{O}$ with respect to the weights, and the network can be trained with gradient descent.

4.4 Error measures

A common error measure for sequence transcription tasks is the test set *label error rate*, which is the total number of insertions, deletions and substitutions on the test set, divided by the total number of target labels in the test set, and multiplied by 100.

For complete word recognition however, a more appropriate error measure is the *word, or sequence, error rate*, which is simply the fraction of test set sequences that were correctly transcribed.

5 Word Matching

When a dictionary is available, a word matching algorithm is needed. Word matching can greatly improve accuracy since many transcription mistakes that result in non-dictionary words are corrected. The example in Fig. 1 shows how word matching corrects the miss-transcribed last letter from **Lam** (ل) to the letter **Reh** (ر), drawn underlined.

The two algorithms we use select the k most probable dictionary words using the RNN output of the sequence transcription stage. If the dictionary is a set of words **S** and the

output sequences are $\mathbf{l} \in \mathbf{L}^{\leq T}$, then the word matching stage outputs the k words $\mathbf{s} \in \mathbf{S}$ that have lowest values of the cost function $D(\mathbf{s},\mathbf{L})$, where $\mathbf{L}$ is the set of the l most probable output sequences (highest $p(\mathbf{l}|\mathbf{x})$). This function estimates the average distances between the dictionary word and the output sequences. The following subsections describe the two word matching algorithms used.

5.1 Weighted edit distance

The weighted edit distance algorithm *WED* is based on the *edit distance* between two sequences $ed(\mathbf{l},\mathbf{s})$ [52] and is found through the summation

$$D(\mathbf{s},\mathbf{L}) = \sum_{\mathbf{l}\in\mathbf{L}} (1 - p(\mathbf{l}|\mathbf{x})) \times ed(\mathbf{l},\mathbf{s}), \tag{21}$$

where $p(\mathbf{l}|\mathbf{x})$ is the output conditional probability. The standard edit distance is defined as the minimum total number of changes, insertions, and deletions required to change pattern $\mathbf{l}$ to pattern $\mathbf{s}$. However, in *WED*, changes take values between 0 and 1 that represent how dissimilar the two interchanged letters are. For this purpose, we created a look-up table based on the Arabic letter shapes. For example, as the letters **Teh** (ت) and **Theh** (ث) have similar shapes, the table gives low change value for these two letters. More detail on this algorithm is in Ref. [2].

5.2 RNNLIB word matching

The RNNLIB has a word matching algorithm that is integrated in the CTC output layer. This algorithm is the CTC token passing algorithm described in Refs. [24,21]. Through this algorithm the CTC labeling is constrained to only the sequences of the complete dictionary words. For any word that has variant spellings, this algorithms sums the probabilities of all the word's variants to find the word probability.

6 Experimental Results

This section describes the experiments carried out to tune the recognition system for efficient results.

Table 2 The IfN/ENIT database of handwritten Arabic words

	Set	Names	PAWs	Characters
Training sets	*a*	6,537	28,298	51,984
	b	6,710	29,220	53,862
	c	6,477	28,391	52,155
	d	6,735	29,511	54,166
	e	6,033	22,640	45,169
Test sets	*f*	8,671	32,918	64,781
	s	1,573	6,109	11,922

6.1 Samples

Our experiments are based on the IfN/ENIT database of handwritten Arabic words [48]. This database is used by more than 110 research groups in about 35 countries [44]. The database version used is v2.0p1e and consists of 32,492 Arabic words handwritten by more than 1,000 writers. These words are 937 Tunisian town/village names. This database is divided into five training sets and two test sets. The numbers of names, parts of Arabic words (PAWs), and characters in each of the sets are shown in Table 2.

The two test sets were publicly unavailable and were only used in competitions. Therefore, we use the five training sets for training, validation, and testing as described below. However, the two test sets were recently released and we were able to use them in some tests as shown in Table 5.

In the rest of this paper, we show results of experiments carried out by training the system using some of these sets and testing using some other set. We refer to each experiment by its training sets followed by its test set, separated by a hyphen. For example, *abcd-e* experiment indicates that the training sets are *a* through *d* and the test set is *e*. Ten percents of the training samples are randomly held out for validation in the RNN training.

6.2 Segmentation algorithm evaluation

We have evaluated the segmentation algorithm through inspection. Figure 7 shows four output examples of this algorithm. The four examples a through d consist of 7, 8, 8, and 7 letters, respectively. Most letters are segmented into one

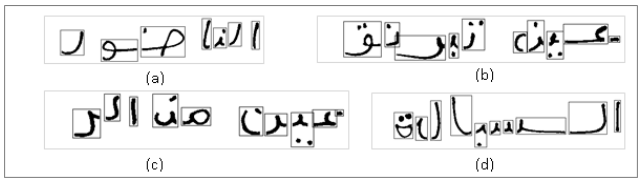


Fig. 7 Examples of the segmentation algorithm results

Table 3 Summary of segmentation algorithm evaluation

Measure	Count	Percentage
Total words	107	100%
Correctly-segmented words	103	96%
Under-segmented words	1	1%
Over-segmented words	3	3%

grapheme but some are over-segmented into two or three. The over-segmented letters in these examples are: ن in words b and c, ة in word d, and سـ in word d. It is the responsibility of the transcriber in the following stage to map one or more graphemes into their corresponding letters. Note that there are also some cases (not shown) where multiple letters in a vertical ligature are segmented into one grapheme.

Table 3 summarizes the evaluation of 107 samples selected from the IfN/ENIT database. The segmentation algorithm produces the expected graphemes accurately in more than 96% of the samples. Only one sample is under segmented and three samples suffer over segmentation. This evaluation was done by human inspection at an early stage of the system development to ensure that the segmentation algorithm is accurate. As illustrated in Section 7, the overall system evaluation on tens of thousands of samples validates the results of this early evaluation.

6.3 Selecting features

We used the mRMR feature selection tool developed by H. Peng to select the best subsets of features for recognizing the segmented objects [49]. We extracted 103 features from 17,943 objects. These sample objects belong to 1,696 randomly selected word images of sets *a* and *b*: 1,233 words from set *a* and 463 words from set *b*. The features along with their grapheme codes were presented to the mRMR tool. This tool discretizes continuous features using their means and standard deviations at thresholds $\bar{x}_k \pm nk\sigma_k$, where $n = 0,1,2,\ldots$ and we selected $k = 0.2$. Table 4 lists the mRMR output of the 103 features ordered according to their mRMR scores (where 'score' is 'relevancy minus redundancy' of the feature as computed by Eqn. 6).

It is interesting to note that the top two features are the configuration features: *Is_Sec* that distinguishes a secondary object from a main object and *Form* that specifies the object's position in the sub-word. Note also that the subset of $m = 30$ includes only three statistical features, five configuration features, two boundary features, seven EFDs, and 10 directional features mainly of the 2×2-region split.

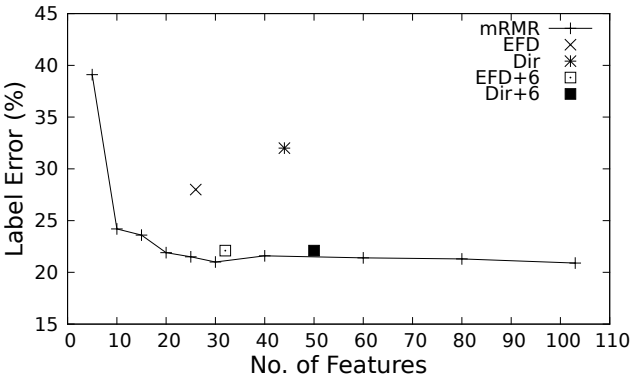


Fig. 8 The label error for *abcd-e* experiments using multiple feature subsets

To find how many features are needed to achieve good recognition accuracy, we find the label error as a function of the number of features used. In each experiment, we use best *m* mRMR features; for *m* from 5 to 103 in varying steps. The results are shown in Fig. 8. In each experiment, we train the **3S** RNN (described the next subsection) using the *m* features from sets *a* through *d* and find the label error of the test set *e* (*abcd-e* experiments).

The curve in this figure shows that the label error decreases rapidly as *m* increases from 5 to 20 and does not improve for $m > 30$. Therefore, we adopt the top 30 features only in our system. The remaining 73 features are not used in the final system as their extraction does not provide added accuracy.

Set *e* is used here for testing because it is the hardest; it has the highest label error among the five sets. However, the other sets demonstrate similar curve shapes but with lower label error and curve knees at smaller *m* values.

6.3.1 Simple feature sets

The set of 30 features used in our final system requires multiple feature extraction techniques. We also performed four more experiments to test simpler feature sets; their results are shown as four points in Fig. 8. The four sets and their label errors are:

EFD is the 26 elliptical Fourier descriptor features, 28%.
Dir is the 44 directional features, 32%.
EFD+6 is the 26 EFDs and six selected features, 22%.
Dir+6 is the 44 directional features and six selected features, 22%.

The six selected features are: *A*, *W*, *H*, *Is_Sec*, *Form*, and $D_{\bar{y}}$. These features are selected because they are readily

Table 4 The features as ordered by the mRMR tool; the best m features are the top features from 1 through m, inclusive

	Feature	Score		Feature	Score		Feature	Score		Feature	Score
1	*Is_Sec*	0.720	31	m	0.243	61	c_4	0.192	91	$D^{2,3}_{0,0,0}$	0.136
2	*Form*	0.399	32	c_5	0.240	62	$D^{2,3}_{1,1,2}$	0.192	92	c_6	0.135
3	c_1	0.422	33	$D^{2,3}_{1,2,1}$	0.238	63	$D^{2,3}_{0,1,1}$	0.192	93	S_a	0.133
4	$T/2D$	0.392	34	$D^{2,2}_{1,0,3}$	0.238	64	W	0.193	94	LR/A	0.128
5	*Ends*	0.370	35	a_3	0.238	65	$D^{2,3}_{1,0,3}$	0.190	95	$D^{2,3}_{0,2,1}$	0.122
6	a_2	0.377	36	E_1	0.230	66	$D^{2,2}_{1,0,1}$	0.190	96	d_6	0.122
7	$\eta_{0,2}$	0.345	37	$D^{2,2}_{0,1,0}$	0.230	67	$D^{2,3}_{0,0,1}$	0.188	97	$D^{2,3}_{0,0,3}$	0.117
8	T	0.322	38	$D^{2,2}_{1,0,2}$	0.229	68	$D^{2,3}_{0,1,2}$	0.185	98	d_1	0.114
9	$D_{\bar{y}}$	0.314	39	A	0.231	69	*Sec_Conf*	0.184	99	UR/A	0.114
10	*Branchs*	0.320	40	$D^{1,1}_{0,0,3}$	0.226	70	$D^{2,3}_{0,2,0}$	0.183	100	b_6	0.112
11	$\bar{x}_N$	0.310	41	c_2	0.224	71	θ	0.179	101	$\bar{x}$	0.104
12	H	0.311	42	$D^{2,2}_{0,0,1}$	0.219	72	d_4	0.176	102	a_0	0.089
13	b_1	0.288	43	c_3	0.216	73	$D^{2,2}_{1,1,0}$	0.174	103	S_b	0.088
14	b_3	0.292	44	$D^{2,3}_{0,1,0}$	0.216	74	b_4	0.173			
15	$D^{2,2}_{1,1,1}$	0.289	45	$D^{2,3}_{1,1,1}$	0.216	75	a_5	0.173			
16	$D^{1,1}_{0,0,2}$	0.289	46	$D^{2,3}_{1,2,2}$	0.215	76	c_0	0.172			
17	$D^{2,3}_{1,0,2}$	0.271	47	$D^{2,3}_{1,2,3}$	0.211	77	$\eta_{2,0}$	0.171			
18	b_2	0.268	48	$D^{2,2}_{1,0,0}$	0.213	78	$D^{2,3}_{1,1,0}$	0.166			
19	γ	0.271	49	$D^{2,3}_{0,2,2}$	0.212	79	$D^{2,3}_{0,1,3}$	0.164			
20	*Loops*	0.255	50	E_2	0.202	80	d_3	0.162			
21	b_5	0.250	51	$D^{1,1}_{0,0,0}$	0.202	81	a_1	0.159			
22	$D^{2,2}_{0,0,2}$	0.249	52	$D^{2,3}_{1,1,3}$	0.201	82	$\bar{y}_N$	0.159			
23	a_4	0.251	53	$D^{2,3}_{0,2,3}$	0.198	83	$D^{2,3}_{1,0,1}$	0.156			
24	$D^{2,2}_{0,1,3}$	0.250	54	W/H	0.195	84	$\bar{y}$	0.155			
25	D_{Top}	0.248	55	S	0.195	85	LL/A	0.154			
26	$D^{2,2}_{0,1,2}$	0.251	56	$D^{2,3}_{0,0,2}$	0.195	86	$D^{2,2}_{0,0,3}$	0.150			
27	$D^{2,3}_{1,0,0}$	0.250	57	$D^{2,3}_{1,2,0}$	0.192	87	$D^{2,2}_{0,1,1}$	0.149			
28	$D^{2,2}_{1,1,3}$	0.249	58	U/A	0.192	88	UL/A	0.144			
29	$D^{1,1}_{0,0,1}$	0.248	59	$D^{2,2}_{0,0,0}$	0.192	89	d_5	0.141			
30	$D^{2,2}_{1,1,2}$	0.247	60	d_2	0.193	90	a_6	0.136			

available from the segmentation process and have relatively high mRMR score.

The homogeneous sets of EFD or Dir have low accuracy. However, complementing them with the six selected features produces simple feature sets with high accuracy. Note that the best label error achieved from the entire 103 features is 21%.

6.4 RNN tuning

We carried out two sets of experiments: the first set for selecting the number of layers and the second set for selecting the size of each layer. Similar to the feature selection experiments, we used $abcd-e$ experiments. As neural network training involves some randomness, each configuration was repeated four times to get the average performance.

To select the number of layers, we used the following configurations:

1 One hidden layer of size 100
2 Two hidden layers of sizes 60 and 180
2S Two hidden layers of sizes 60 and 180 with sub-sampling layer of size 60
3 Three hidden layers of sizes 40, 80, and 180
3S Three hidden layers of sizes 40, 80, and 180 with two sub-sampling layers of sizes 40 and 80
4 Four hidden layers of sizes 40, 80, 120, and 180

These layer sizes are the default sizes that are found in the RNNLIB library's configuration files. Figure 9 shows the results of testing these six configurations. These results show that the accuracy improves with more layers and with using sub-sampling layers. However, the accuracy does not increase when increasing the number of layers from three to four. Therefore, we adopt the topology of three layers with two sub-sampling layers.

In order to achieve higher accuracy, we use layer sizes larger than those used in the default **3S** configuration [51]. The selected *tuned* neural network size: three hidden layers of sizes 100, 100, and 360 with two sub-sampling layers

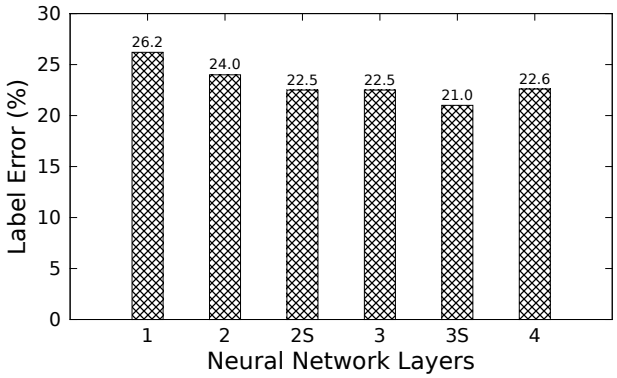


Fig. 9 The label error for *abcd-e* experiments using the top 30 selected features on neural networks of six topologies

of sizes 120 and 180. This configuration achieves a label error of 19.8%, whereas the label error of the **3S** configuration is 21.0%. The tuned RNN size is used in the following comparison.

7 Comparison

This section compares our results with the results of other best systems. We also present a detailed comparison with the MDLSTM system by Graves and Schmidhuber which has the best accuracy on test set *f* [21].

7.1 Arabic handwriting recognition competitions

Table 5 summarizes the results of the best systems that have participated in the series of Arabic handwriting recognition competitions. We also include here the results of: our earlier system JU-OCR that has participated in ICDAR 2011, the recently published system AUC, an improved version of the system that has won ICDAR 2009 competition (MDLSTM2), and the system described in this paper JU-OCR2. The table shows the word recognition accuracy of four test sets (*d*, *e*, *f*, and *s*) and the training sets used. For test sets *d* and *e*, the table also shows the percentage of correct result within the first five results and within the first 10 results, when available. The UOB, Siemens, UPV-PRHLT, RWTH-OCR, and AUC systems' results for these two test sets are the published results [7, 14, 10].

In the first competition (ICDAR 2005), set *e* was unknown to the participants and was used as the test set. The UOB system scored 75.93% on this set. In later competitions, set *e* was made available to the participants and new test sets (sets *f* and *s*) were used.

Our earlier system JU-OCR has relatively low accuracy. Although it uses the grapheme segmentation algorithm described in this paper, but it uses inferior classifier (random forests) and less efficient feature set and word matching algorithm [44].

The last two rows in Table 5 summarize the word recognition accuracy of our experiments conducted to evaluate JU-OCR2 against MDLSTM2. These results illustrate that the two systems hold the best published accuracies on the four test sets. JU-OCR2 has the highest accuracy on set *d* and *s* at 98.96% and 84.80%, respectively. MDLSTM has the highest accuracy on sets *e* and *f* at 94.76% and 94.13%, respectively. The JU-OCR2 results are using *WED* word matching. However, the *abc-d* result is using the RNNLIB word matching because it has the best accuracy here (98.96% *vs.* 98.75%).

We didn't expect that JU-OCR2 scores lower than MDLSTM2 on set *e* because, as shown below, the former system has lower label and sequence error rates. However, it seems that MDLSTM2 benefits more from the integrated word matching stage and achieves higher accuracy on set *e*.

The following subsections describe the MDLSTM system and provide a detailed comparison between it and JU-OCR2.

7.2 Multidimensional LSTM (MDLSTM)

A system similar to the RNN employed in this work won the Arabic, Farsi and French offline handwriting recognition competitions at ICDAR 2009. This system also employed a CTC output layer together with the LSTM network architecture to transcribe handwritten text. The main difference is that the network used for ICDAR 2009 extracted input features directly from the raw pixel data of the images, using a hierarchy of network layers. Because images are two dimensional, this necessitated the use of multidimensional LSTM (MDLSTM; [25]). MDLSTM differs from ordinary bi-directional LSTM in that there are four distinct hidden layers instead of two, and each of these layers receives information from two previous states instead of one. The complete MDLSTM recognition system is described in detail in Ref. [22].

The advantage of using raw pixels is that the system can be easily adapted to new languages. However, as the results in the next section demonstrate, expertly chosen input features from the segmented sub-words can give better performance.

7.3 Detailed comparison

Figure 10 shows the label error of five test sets and the average label error on MDLSTM and JU-OCR2. The figure demonstrates that JU-OCR2 achieves better label error; its average label error is 8.63% and the MDLSTM average label error is 10.39%. Relative to MDLSTM, JU-OCR2 average label error is 16.9% lower. These results are obtained using the open-source RNNLIB version that was used by MDLSTM in ICDAR 2009 competition [19].

Table 5 Recognition results of correctly recognized images in % for nine systems

Competition	ID	Approach	*abc-d* top 1	*abc-d* top 5	*abc-d* top 10	*abcd-e* top 1	*abcd-e* top 5	*abcd-e* top 10	*abcde-f* top 1	*abcde-s* top 1
ICDAR 2005 [40]	UOB	HMM	85.0			75.93	87.99	90.88		
ICDAR 2007 [41]	Siemens	HMM				81.89			87.22	73.94
ICDAR 2009 [42]	MDLSTM	NN							93.37	81.06
ICFHR 2010 [43]	UPV-PRHLT	HMM	95.2			93.9			92.20	84.62
ICDAR 2011 [44]	RWTH-OCR	HMM	96.53			92.74			92.20	84.55
ICDAR 2011 [44]	JU-OCR	RF	75.49 [1]	89.29	92.47	63.75	80.84	86.06	63.86	49.75
– [10]	AUC	HMM	97.7			93.44			93.1	84.8
–	MDLSTM2	NN	98.57	99.76	99.87	**94.76**	98.71	99.20	**94.13**	84.66
–	JU-OCR2	NN	**98.96**	99.87	99.91	93.46	98.09	98.85	92.46	**84.80**

[1] Trained on only 1,696 words from sets *a* and *b*

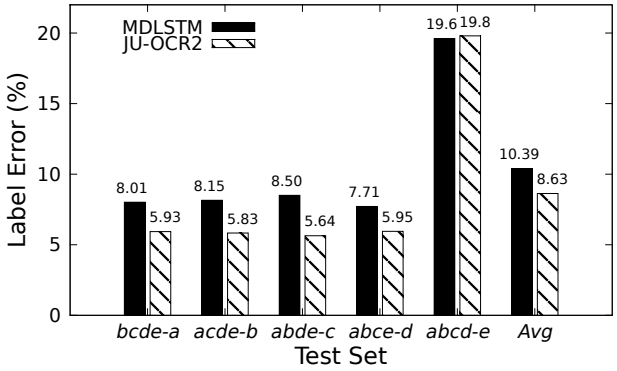


Fig. 10 Label error comparison between MDLSTM and JU-OCR2 over five test sets

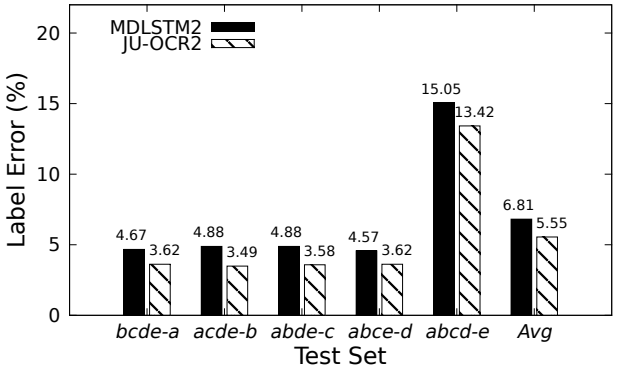


Fig. 11 Label error comparison between MDLSTM2 and JU-OCR2 over five test sets with the weight noise option

We have also repeated the above experiments using a new version of RNNLIB and their results are shown in Fig. 11. The new version has slightly better performance as it has some minor bugs solved. However, the results of both systems shown in Fig. 11 are much better than the results in Fig. 10. The main reason for this improvement is that both systems were trained with the *weight noise* option. Injecting weight noise is used to improve the convergence and generalization abilities of RNNs [46]. We have noticed that best results are achieved when we first train the system without this option then retrain it with this option starting from the weights of the last best epoch. JU-OCR2 achieves better label error for all sets; its average label error is 5.55% and the MDLSTM2 average label error is 6.81%. Relative to MDLSTM2, JU-OCR2 average label error is 18.5% lower.

Figure 12 shows the sequence error of the five test sets on the two systems using the experiments described above. Consistent with the label error results, JU-OCR2 achieves better sequence error for all sets; its average sequence error is 24.28% and the MDLSTM2 average sequence error is 31.26%. Relative to MDLSTM2, JU-OCR2 average sequence error is 22.3% lower.

Moreover, although MDLSTM is the fastest system in ICDAR 2009 [42], JU-OCR2 is faster. The JU-OCR2's average time to recognize one image in the *abcd-e* experiment is 147 ms *vs.* 213 ms for MDLSTM2 (31% advantage). As MDLSTM processes the entire image pixels in the RNN sequence transcription, it takes 137 ms in sequence transcription and word matching *vs.* only 73 ms in JU-OCR2. The remaining 76 ms of MDLSTM2 are spent preparing the input files from the word images, whereas, the remaining 74 ms of JU-OCR2 are spent in the segmentation and feature extraction stages. These experiments were carried out on Ubuntu

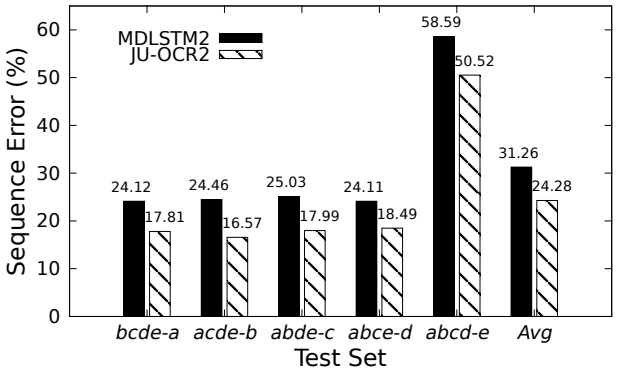


Fig. 12 Sequence error comparison between MDLSTM2 and JU-OCR2 over five test sets

12.04 computer with Intel Core 2 Quad CPU Q9550 running at 2.83 GHz and equipped with 2 GB memory.

8 Conclusion

We have described the JU-OCR2 system for recognizing handwritten Arabic words. This system segments the cursive words into graphemes; most of the letters are segmented into one grapheme each, some letters are over-segmented into multiple graphemes, and some vertical letter ligatures are under-segmented into one composite grapheme each. A set of 30 features is selected from a rich set of features that are extracted from the segmented bodies. The feature vector is passed to an RNN sequence transcriber that features bi-directional long short-term memory to exploit grapheme contexts and achieve high recognition accuracy.

The transcriber maps the feature vectors of the segmented bodies directly into letters and is able to achieve a low label error rate (5.55% on average for the IfN/ENIT database), without the use of a dictionary. This suggests that the system may be suitable for unlimited vocabulary recognition. The high label accuracy is made possible by the robust segmentation algorithm, the careful feature selection, and the well-tuned RNN.

Although the best systems in recent Arabic handwriting recognition competitions have used holistic approaches, we have demonstrated that our segmentation-based approach gives lower label and sequence error rates and has best published accuracy on some test sets. The MDLSTM system that holds the record accuracy in these competitions on test set f uses the same RNN classifier as JU-OCR2. The main difference between the two systems is the feature extraction approaches. Whereas MDLSTM extracts features from the raw pixels, JU-OCR2 extracts features from the segmented bodies.

We evaluated JU-OCR2 against an improved version of MDLSTM. Compared with MDLSTM2, JU-OCR2 reduces the label error, sequence error, and execution time by 18.5%, 22.3%, and 31%, respectively.

We have inspected 100 randomly-selected, miss-classified samples to find what went wrong. As Table 6 shows, we classify these samples according to the problem source into four types: transcription problem, bad hand writing, segmentation problem, and feature extraction problem (ordered from most to least frequent). The table shows eight example samples, two of each type with explanations. We think that many of these cases can be solved by improving the segmentation, feature extraction, and transcription stages.

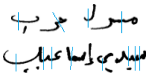
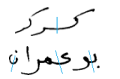
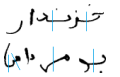
As part of our future work plans, we intend to improve the system to solve these problems. We also plan to test our system on complete documents with unlimited vocabulary. We also intend to add some pre-processing stages that have been proven to enhance the recognition accuracy [7, 14, 10].

Acknowledgements This work was supported by the Deanship of the Scientific Research in the University of Jordan. Some of this research was completed when G. Abandah was in a sabbatical leave in Princess Sumaya University for Technology. We would like to thank Alex Graves for making the RNNLIB publically available [19], for giving us a copy of the latest RNNLIB version, and for his help in using it. We also thank him for providing parts of the RNN sequence transcriber description included in Section 4. We would like to thank Hanchuan Peng for making mRMR tools publically available [49]. We would like also to thank Haikal El Abed for giving us copies of sets f and s of the IfN/ENIT database.

References

1. Abandah, G., Jamour, F.: Recognizing handwritten Arabic script through efficient skeleton-based grapheme segmentation algorithm. In: Int'l Conf. Intelligent Systems Design and Applications, pp. 977–982 (2010)
2. Abandah, G., Jamour, F.: Word matching algorithms for recognizing handwritten Arabic words (2013). Submitted
3. Abandah, G., Khedher, M.: Analysis of handwritten Arabic letters using selected feature extraction techniques. Int'l J. Computer Processing of Languages **22**(1), 49–73 (2009)
4. Abandah, G., Malas, T.: Feature selection for recognizing handwritten Arabic letters. Dirasat Engineering Sciences J. **37**(2), 242–256 (2010)
5. Al-Hajj, R., Likforman-Sulem, L., Mokbel, C.: Arabic handwriting recognition using baseline dependant features and hidden Markov modeling. In: Int'l Conf. Document Analysis and Recognition, pp. 893–897 (2005)
6. Alginahi, Y.M.: A survey on Arabic character segmentation. Int'l J. Document Analysis Recognition **16**(2), 105–126 (2013)
7. Alkhoury, I., Giménez, A., Juan, A.: Arabic handwriting recognition using Bernoulli HMMs. In: V. Märgner, H. El Abed (eds.) Guide to OCR for Arabic Scripts, pp. 255–272. Springer London (2012)
8. Amin, A.: Arabic character recognition. In: H. Bunke, P. Wang (eds.) Handbook of Character Recognition and Document Image Analysis, pp. 397–420. World Scientific (1997)
9. Arica, N., Yarman-Vural, F.: Optical character recognition for cursive handwriting. IEEE Trans Pattern Anal Mach Intell **24**(6), 801–813 (2002)
10. Azeem, S.A., Ahmed, H.: Effective technique for the recognition of offline Arabic handwritten words using hidden Markov models. Int'l J. Document Analysis and Recognition (2013). DOI 10.1007/s10032-013-0201-8
11. Chang, F., Chen, C.J., Lu, C.J.: A linear-time component-labeling algorithm using contour tracing technique. Computer Vision and Image Understanding **93**(2), 206–220 (2004)
12. Deutsch, E.: Thinning algorithms on rectangular, hexagonal, and triangular arrays. Comm of the ACM **15**(9), 827–837 (1972)
13. Douglas, D., Peucker, T.: Algorithms for the reduction of the number of points required to represent a line or its caricature. The Canadian Cartographer **10**(2), 112–122 (1973)
14. Dreuw, P., Rybach, D., Heigold, G., Ney, H.: RWTH OCR: A large vocabulary optical character recognition system for Arabic scripts. In: V. Märgner, H. El Abed (eds.) Guide to OCR for Arabic Scripts, pp. 215–254. Springer London (2012)
15. El Abed, H., Märgner, V.: Comparison of different preprocessing and feature extraction methods for offline recognition of handwritten Arabic words. In: Int'l Conf. Document Analysis and Recognition, pp. 974–978 (2007)
16. El-Hajj, R., Likforman-Sulem, L., Mokbel, C.: Arabic handwriting recognition using baseline dependant features and hidden Markov

Table 6 Miss-classified samples grouped according to the problem type. The table shows the frequency of the each problem type, the sample with its segmentation points in blue, its correct target word, the wrong output, and explanation.

Problem type	Sample	Target	Output	Explanation
Transcription (37%)		القصر الخلصة	الفصر الخلية	Letter ق miss transcribed as ف Letter ص miss transcribed as ي
Writing (27%)		منزل حرب سيدي اسماعيل	مركمرب سيدي اساعيلي	The writer didn't write the dots of Letters ن and ز The writer wrote very small ه and shifted the dots of last ي to the left
Segmentation (25%)		كركر بوعمران	كرد بوعران	The segmentation algorithm missed segmenting the second كر The segmentation algorithm missed the second segment in عمر
Feature Ext. (11%)		خزندار بومرداس	توندار بومراس	The three dots are very small and not detected as secondary bodies As the د is very small, it was detected as a secondary body

modeling. In: Int'l Conf. Document Analysis and Recognition, pp. 893–897 (2005)
17. Freeman, H.: On the encoding of arbitrary geometric configurations. IRE Trans. Electron Computer **10**(2), 260–268 (1961)
18. Gers, F.: Long short-term memory in recurrent neural networks. Ph.D. thesis, Ecole Polytechnique Fédérale de Lausanne (2001)
19. Graves, A.: RNNLIB: a recurrent neural network library for sequence learning problems. http://sourceforge.net/projects/rnnl/
20. Graves, A.: Supervised sequence labelling with recurrent neural networks. Ph.D. thesis, Technische Universität München (2008)
21. Graves, A.: Offline Arabic handwriting recognition with multidimensional recurrent neural networks. In: V. Märgner, H. El Abed (eds.) Guide to OCR for Arabic Scripts, pp. 297–313. Springer London (2012)
22. Graves, A.: Supervised Sequence Labelling with Recurrent Neural Networks, *Studies in Computational Intelligence*, vol. 385. Springer (2012)
23. Graves, A., Fernández, S., Gomez, F., Schmidhuber, J.: Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In: Int'l Conf. Machine Learning (2006)
24. Graves, A., Fernández, S., Liwicki, M., Bunke, H., Schmidhuber, J.: Unconstrained online handwriting recognition with recurrent neural networks. Advances in Neural Information Processing Systems **20**, 1–8 (2008)
25. Graves, A., Fernández, S., Schmidhuber, J.: Multi-dimensional recurrent neural networks. In: Int'l Conf. Artificial Neural Networks (2007)
26. Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A novel connectionist system for unconstrained handwriting recognition. IEEE Trans. Pattern Analysis Machine Intelligence **31**(5), 855–868 (2009)
27. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks **18**(5), 602–610 (2005)
28. Graves, A., Schmidhuber, J.: Advances in Neural Information Processing Systems, NIPS'22, vol. 22, chap. Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks, pp. 545–552. MIT Press, Vancouver (2009)
29. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Maching Learning Research **3**(1), 1157–1182 (2003)
30. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Computation **9**(8), 1735–1780 (1997)
31. Kuhl, F., Giardina, C.: Elliptic Fourier features of a closed contour. Computer Graphics and Image Processing **18**(3), 236–258 (1982)
32. Kundu, A., Hines, T., Phillips, J., Huyck, B.D., Van Guilder, L.C.: Arabic handwriting recognition using variable duration HMM. In: Int'l Conf. Document Analysis and Recognition, vol. 2, pp. 644–648 (2007)
33. Lee, H., Verma, B.: Binary segmentation algorithm for English cursive handwriting recognition. Pattern Recognition **45**(4), 1306–1317 (2012)
34. Lewis, M.P. (ed.): Ethnologue: Languages of the World. SIL International (2009)
35. Likforman-Sulem, L., Mohammad, R.A.H., Mokbel, C., Menasri, F., Bianne-Bernard, A.L., Kermorvant, C.: Features for HMM-based Arabic handwritten word recognition systems. In: V. Märgner, H. El Abed (eds.) Guide to OCR for Arabic Scripts, pp. 123–143. Springer London (2012)
36. Liu, C.L.: Handwritten Chinese character recognition: Effects of shape normalization and feature extraction. In: D. Doermann, S. Jaeger (eds.) Arabic and Chinese Handwriting Recognition, vol. LNCS 4768, pp. 104–128. Springer Berlin / Heidelberg (2008)
37. Lorigo, L., Govindaraju, V.: Segmentation and pre-recognition of Arabic handwriting. In: Int'l Conf. Document Analysis and Recognition, pp. 605–609 (2005)
38. Lorigo, L., Govindaraju, V.: Offline Arabic handwriting recognition: A survey. IEEE Trans Pattern Anal Mach Intell **28**(5), 712–724 (2006)
39. Maddouri, S.S., El-Abed, H., Samoud, F.B., Bouriel, K., Ellouze, N.: Baseline extraction: Comparison of six methods on IFN/ENIT database. In: Int'l Conf. Frontiers in Handwriting Recognition (2008)
40. Märgner, V., El Abed, H.: ICDAR 2005 - Arabic handwriting recognition competition. In: Int'l Conf. Document Analysis and Recognition, pp. 70–74 (2005)
41. Märgner, V., El Abed, H.: ICDAR 2007 - Arabic handwriting recognition competition. In: Int'l Conf. Document Analysis and Recognition, pp. 1274–1278 (2007)
42. Märgner, V., El Abed, H.: ICDAR 2009 - Arabic handwriting recognition competition. In: Int'l Conf. Document Analysis and Recognition, pp. 1383–1387 (2009)
43. Märgner, V., El Abed, H.: ICFHR 2010 - Arabic handwriting recognition competition. In: Int'l Conf. Frontiers in Handwriting Recognition, pp. 709–714 (2010)
44. Märgner, V., El Abed, H.: ICDAR 2011 - Arabic handwriting recognition competition. In: Int'l Conf. Document Analysis and Recognition, pp. 1444–1448 (2011)
45. Motawa, D., Amin, A., Sabourin, R.: Segmentation of Arabic cursive script. In: Int'l Conf. Document Analysis and Recognition, pp.

625–628 (1997)
46. Murray, A., Edwards, P.: Synaptic weight noise during multilayer perceptron training: Fault tolerance and training improvements. IEEE Trans Neural Networks **4**(4), 722–725 (1993)
47. Pechwitz, M., El Abed, H., Märgner, V.: Handwritten Arabic word recognition using the IFN/ENIT-database. In: V. Märgner, H. El Abed (eds.) Guide to OCR for Arabic Scripts, pp. 169–213. Springer London (2012)
48. Pechwitz, M., Maddouri, S.S., Märgner, V., Ellouze, N., Amiri, H.: IFN/ENIT - database of handwritten Arabic words. In: Colloque Int'l Francophone sur l'Ecrit et le Document, pp. 129–136 (2002)
49. Peng, H.: mRMR (minimum redundancy maximum relevance feature selection). http://penglab.janelia.org/proj/mRMR/
50. Peng, H., Long, F., Ding, C.: Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. IEEE Trans Pattern Anal Mach Intell **27**(8), 1226–1238 (2005)
51. Qaralleh, M., Abandah, G., Jamour, F.: Tuning recurrent neural networks for recognizing handwritten Arabic words. J. Software Engineering & Applications **6**(10), 533–542 (2013)
52. Ratcliff, J., Metzener, D.: Pattern matching: the Gestalt approach. Dr. Dobb's J. **13**(7), 46–72 (1988)
53. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: foundations, chap. Learning Internal Representations by Error Propagation, pp. 318–362. (1986)
54. Safabakhsh, R., Adibi, P.: Nastaaligh handwritten word recognition using a continuous-density variable-duration HMM. The Arabian J. Science and Eng. **30**(1B), 95–118 (2005)
55. Sari, T., Souici, L., Sellami, M.: Off-line handwritten Arabic character segmentation algorithm: ACSA. In: Int'l Workshop on Frontiers in Handwriting Recognition, pp. 452–457 (2002)
56. Schambach, M.P., Rottland, J., Alary, T.: How to convert a Latin handwriting recognition system to Arabic. In: Int'l Conf. Frontiers in Handwriting Recognition (2008)
57. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. IEEE Trans. Signal Processing **45**, 2673–2681 (1997)
58. Smith, R.: An overview of the Tesseract OCR engine. In: Int'l Conf. Document Analysis and Recognition, vol. 2, pp. 629–633 (2007)
59. Wshah, S., Shi, Z., Govindaraju, V.: Segmentation of Arabic handwriting based on both contour and skeleton segmentation. In: Int'l Conf. Document Analysis and Recognition, pp. 793–797 (2009)