Arabic Character Recognition

Nachum Dershowitz and Andrey Rosenberg[⋆]

School of Computer Science, Tel Aviv University, Ramat Aviv, Israel

Abstract. Although optical character recognition of printed texts has been a focus of research for the last few decades, Arabic printed text, being cursive, still poses a challenge. The challenge is twofold: segmenting words into letters and identifying individual letters. We describe a method that combines the two tasks, using multiple grids of SIFT descriptors as features. To construct a classifier, we do not use a large training set of images with corresponding ground truth, a process usually done to construct a classifier, but, rather, an image containing all possible symbols is created and a classifier is constructed by extracting the features of each symbol. To recognize the text inside an image, the image is split into "pieces of Arabic words", and each piece is scanned with increasing window sizes. Segmentation points are set where the classifier achieves maximal confidence. Using the fact that Arabic has four forms of letters (isolated, initial, medial and final), we narrow the search space based on the location inside the piece.
The performance of the proposed method, when applied to printed texts and computer fonts of different sizes, was evaluated on two independent benchmarks, PATS and APTI. Our algorithm outperformed that of the creator of PATS on five out of eight fonts, achieving character correctness of 98.87%–100%. On the APTI dataset, ours was competitive or better that the competition.

1 Introduction

After more than forty years of research, *optical character recognition* (OCR) systems for machine-printed text show impressive performance [5]. However, printed Arabic texts still present difficult challenge. The reasons are manifold:

(a) Even printed text is semi-cursive. Each word consist of one or more *pieces (of an Arabic word)* (paws). The letters inside a paw are connected (as in cursive script) and cannot be easily separated, since finding the correct segmentation point is itself a challenge.
(b) Many Arabic letters are distinguished one from another only by diacritical dots or strokes. Misclassifying them can lead to a completely different word. Diacritical marks representing vowels are often left out and the meaning of a word is identified from the context.

[⋆] This paper is based on A.R.'s M.Sc. thesis, *Using SIFT Descriptors for OCR of Printed Arabic*, Tel Aviv University, Feb. 2012 (available at http://nachum.org/papers/AndreyThesis.pdf).

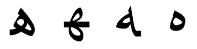


Fig. 1: Initial, medial, final and isolated forms of the letter hā' in the Arial font.

(c) The same letter may be written differently, depending on its location in the word, as there are up to four different variations in form for each letter, isolated, initial, medial and final (see Fig. 1).
(d) For some fonts, some combinations of letters may result in a new symbol (ligature). These multiple forms and combinations of letters significantly increase the number of different graphically-represented symbols a classifier needs to recognize to well over a hundred, besides punctuation marks and numerals.

By using features extracted with a grid of scale invariant feature transform (SIFT) descriptors and a sliding-window technique, we aim to *jointly* solve the segmentation and recognition problems for printed Arabic. We scan each paw and consider different segmentations of it into letters. Each form (initial, medial, final and isolated) has its own classifier. For each possible segmentation and based on the location of the window inside the paw, an appropriate classifier is chosen and a set of letters is suggested. The algorithm chooses those segmentation points for which the classifier achieves its highest confidence in the recognized letters.

Given a font, we construct a classifier based purely on the images of letters in all possible forms or combinations of letters that are graphically-represented by a single symbol. Our classifier does not undergo the classical training phase, where a part of a tested dataset is used for training, while the other part is used for performance evaluation. We did not use language models, morphological information, word lists or any other language resources that are commonly used to improve performance.

Recently, SIFT descriptors [8] have been suggested for use in OCR. They were used for Chinese character recognition in [15] and [6], and were applied to degraded handwritten Latin manuscripts in [4]. The authors of [1] showed that a SIFT descriptor outperform the classical feature representation methods, such as PCA, when performing word based classification of cursive Pashto printed text (very similar to the Arabic printed text). They used the keypoints suggested in [9] as centres of the SIFT descriptors. SIFT descriptors were also used to recognize the font of an Arabic printed text [16].

As described in [14], Hidden Markov Model (HMM) serve as a base for most of the methods that perform recognition of cursive scripts. The author of PATS dataset [3], the first dataset we used to test our algorithm, uses HMM and a sliding window to segment and recognize Arabic scripts. See also the recent [2]. In a competition [13] recently conducted by the authors of APTI dataset [12], the second data set we used to test our algorithm, participated two HMM based systems. The system that was suggested by the authors themselves did not participate in the competition, but was also HMM based. The first system that participated in the competition is based on the Hidden Markov Model Toolkit (a toolkit that was originality designed to be used for speech recognition), which was customized to a purpose of character recognition. The second system that participated in the competition is based on Bernoulli HMMs (BMMs), that is, HMMs in which conventional Gaussian mixture density functions are replaced with Bernoulli mixture probability functions.

The remainder of this paper is organized as follows: In the next section, we present our proposed algorithm. In Sect. 3, we describe the datasets that were used to measure performance, compare the results of other OCR algorithms that were evaluated on those datasets, and analyze the results. Finally, we conclude with a few observations.

2 SOCR Algorithm

Our algorithm, which we call *SOCR* for "SIFT-based OCR", segments and recognizes the letters of an image containing a single paw, and is described in Sect. 2.7. Using a sliding-window technique a candidate letter is isolated inside each window (see Sect. 2.6) and classified using the appropriate classifiers (see Sect. 2.4). The appropriate classifiers are chosen base on the location of the window inside the paw and can be either of the four form classifiers (see Sect. 2.3). The best segmentation points and letters are chosen based on the confidence of the classifier for the letter to end at this segmentation point.

We assume that the image passed a preprocessing phase and contains only Arabic text consisting of letters of a predefined alphabet located on a white background and the baselines are horizontally aligned. We also assume that the image contains only one line of text. An accurate segmentation of an image containing more than one line of text to a set of images where each image contains only one line is out of the scope of this work. While the segmentation of a line or a word to paws is a significant part of the algorithm and described in Sect. 2.5, a segmentation of a line into words is required only when the performance in terms of word recognition rate is a significant performance metric (see Sect. 3.1). A segmentation into words can be achieved by distinguish between white spaces and spaces between paws inside a word. We suggest a method for this in Sect. 3.2. For each input image, we estimate the baseline to be the row having the most black ink among all rows. The baseline is needed to correctly split the image to paws (see Sect. 2.5) and to isolate correctly a letter inside a window see Sect. 2.6). A more accurate baseline estimation that was designed for handwritten texts and considers the diacritical dots and marks of Arabic [17] was considered, but not used in this work.

The extraction of features lies at the core of any classification process. The features of all the unique symbols of a given font (limited to an alphabet) are extracted during the construction of the classifier as described in Sect. 2.3. Those features are later compared to the features extracted from the image of a letter that we are trying to classify as described in Sect. 2.4.

We use a multiple grids of SIFT descriptors as the main feature set. The structure and extraction process of the grids of descriptors are described in

Sect. 2.1. To fine-tune the classification results produced by the use of SIFT descriptors, we use additional features. The structure and extraction process of those additional features are described in Sect. 2.2.

2.1 Extracting a Grid of SIFT Descriptors

Given an image that contains a whole paw or a part of a paw, the following steps are executed to extract the grid of SIFT descriptors:

1. The image is padded with white to the size of the smallest bounding *square* of the paw.
2. The image is split into $G \times G$ identical squares, where G is a small constant.
3. Let W be the width of each square and the scale be $W/4$.
4. In the middle of each square, extract N descriptors, where $M = \{m_1, \ldots, m_N\}$ are the magnification factors.
5. Each extracted descriptor is identified by $D_{x,y,m}$, where x, y are the coordinates of the square in the grid and m is the magnification factor.

When no magnification is used, by design, the grid of descriptors should cover the whole image without overlapping each other; hence the scale of the descriptor is always set to $W/4$ due to the fact that each descriptor has 4 spatial bins in each spatial direction. Throughout this work, $G = 3$ and $M = \{0.5, 1.0, 1.5\}$. Figure 2 shows an example of a grid of SIFT descriptors that were extracted without using magnification.

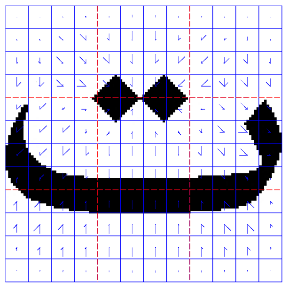
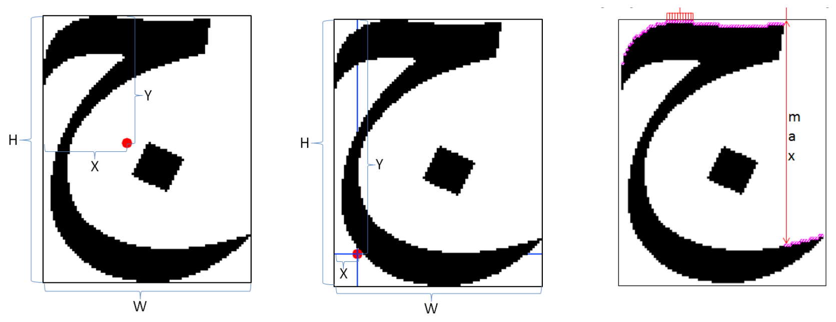


Fig. 2: A grid of 3×3 SIFT descriptors of the letter tā' of isolated/final form. Each descriptor is separated by a dashed (red) line.

Alternative Points for SIFT Descriptors Extraction It was suggested by the author of SIFT to extract descriptors at keypoints where maxima and minima of the result of difference-of-Gaussians function applied in scale-space to a series of smoothed and re-sampled images is achieved [9]. While this method might perform well for complex images, it achieved very low performance on Arabic letters and had major disadvantages:



(a) The center of mass of the letter jīm is marked by the (red) dot. The center of mass feature is (X/W,Y/H).

(b) The crosshair of the letter jīm is marked by the (red) dot. The crosshair feature is (X/W,Y/H).

(c) The top outline of the letter jīm is marked by the semi-filled (purple) dots. The top-outline feature is $(t_1, \ldots, 1, \ldots, 0, \ldots, t_W)$.

Fig. 3: Additional features of the letter jīm.

a) The number of keypoints depends on the resolution.
b) For some simple letters, the method did noy produce any keypoints at all.
c) Similar shapes can have almost identical sets of descriptors.

Those findings were later, independently, reported in [1]. We overcame these disadvantages by combining padding to the bounding square, a constant-size grid of descriptors and the quantization process described in Sect. 2.3.

2.2 Extracting Additional Features

Additional features are used to penalize the confidence of letters that are suggested by the SIFT classifier (see Sect. 2.4). The next sections describe the additional features that we used and how to extract them. We assume that the text inside the image is bounded by its borders and consists of black pixels located on a white background.

Center of Mass Feature The *center of mass feature* f_m is the relative location (relative to the height and width of the image) of the center of mass of the black ink. The center of mass of the letter jīm is shown in Fig. 3a. Given an image and a letter, where c and c' are their centers of mass, respectively, the *center of mass penalty* used is $p_m = 1/(1 + d^{\frac{1}{2}}(c, c'))$, where $d^{\frac{1}{2}}$ gives the square-root of the Euclidean distance (a commonly used measure).

Crosshair Feature The *crosshair feature*, f_c, is the relative location (relative to the height and the width of the image) of the vertical and horizontal slices with the largest portion of black ink compared to the white background. The crosshair of the letter jīm is shown in Fig. 3b. Given an image and a letter, where c and c' are their crosshair features, respectively, the *crosshair penalty* used is $p_c = 1/(1 + d^{\frac{1}{2}}(c, c'))$.

Ratio Feature The *ratio feature*, f_o, is the height divided by the width of the bounding box of the black ink. Given an image and a letter, where o' and o are their ratio features, respectively, the *ratio penalty* used is $p_o = 1/(1 + (o' - o)^2)$. The exponent was arbitrary set to be 2 without being optimized for any of the datasets tested in Sect. 3.

Outline Features Each image has four *outline features*, top, left, bottom and right. The top-outline feature, $f_t = (t_1, \ldots, t_W)$, where W is the width of the bounding box of the black ink, is calculated as follows:

1. For $i = 1, ..., W$, let d_i be the distance from the top of the bounding box to the first occurrence of a black pixel in the ith column of the image.
2. For $i = 1, ..., W$, let t_i be $(\max\{d_j\} - d_i)/(\max\{d_j\} - \min\{d_j\})$, where maximum and minimum are taken over all columns.

The left (f_l), bottom (f_b) and right (f_r) outline features are calculated in a similar manner. The top-outline of the letter jīm is shown in Fig. 3c.

Given an image and a letter, where t and t' are its top-outline features, respectively, the *top-outline penalty*, p_t, is calculated as follows:

1. If the two feature vectors are of unequal length, downscale the longer one, so they are both of some length n.
2. Define $p_t = 1/(1 + \operatorname{avg}_{i=1}^{n} |t_i - t'_i|)$, where avg takes the average.

The left (p_l), bottom (p_b) and right (p_r) outline penalties are calculated in a similar manner.

Black Ink Histogram Features Each image has a horizontal *black ink histogram feature* and a vertical one. The horizontal black ink histogram feature, $f_h = (h_1, \ldots, h_H)$, where H is the height of the bounding box of the black ink, is calculated as follows:

1. For $i = 1, ..., H$, let b_i be the number of black ink pixels in row i.
2. For $i = 1, ..., H$, let h_i be $b_i/\max\{b_i\}$.

The vertical black ink histogram feature (f_v) is calculated in a similar manner.

Given an image and a letter, where h and h' are its horizontal black ink histogram features, respectively, the horizontal *black ink histogram penalty*, p_h, is calculated as follows:

1. If the two feature vectors are of unequal length, downscale the longer one, so they are both of some length n.
2. Define $p_h = 1/(1 + \operatorname{avg}_{i=1}^{n} |h_i - h'_i|)$.

The vertical black ink histogram penalty (p_v) is calculated in a similar manner.

2.3 Constructing a Classifier

For each classifier C for font F and an alphabet Σ, we execute a series of operations as described below. In Sect. 2.3, we explain how we generated high-resolution images, each one containing a unique symbol of the alphabet Σ written in font F. In Sect. 2.3, we explain how to extract the SIFT descriptors and additional features from each image and group the SIFT descriptors into four groups based on the location where the unique symbol can appear in a word (isolated, initial, medial and final). In Sect. 2.3, we describe the *quantization* process on the SIFT descriptors and group them into four groups creating a separate classifier for each of the four forms of letters. Finally, in Sect. 2.3, we compute a *base confidence* for each unique symbol.

Creating Images for All Possible Symbols To create an image for each unique symbol for the alphabet Σ written in font F, a Word® document that contains $|\Sigma|$ rows, representing all possible letters of the alphabet, and four columns, representing the different possible letter forms (isolated, final, medial and initial) is created. Since some letter combinations are graphically-represented by a single symbol (ligature), these combinations are referred to as letters and belong to Σ. Each row can have one to four symbols, since some letters do not have all four letter forms, but only isolated or initial forms, or even only an isolated form. The resulting document is exported as a high resolution image. See Sect. 3 for details about the alphabet and the resolution of the image that was exported for each tested font. The exported image is split into lines and each line is split into the number of unique symbols it contains resulting in an image for each possible symbol. We denote each image by $L_{i,r}$, where $r \in \{$isolated, initial, medial, final$\}$ is the form of the ith letter of the alphabet Σ.

Feature Extraction Before we extract features, we assign to each image $L_{i,r}$ a unique identification number σ. We extract SIFT descriptors and additional features as described in Sects. 2.1 and 2.2, respectively. The number of extracted SIFT descriptors per symbol is NG^2, where, as we said, N is the number of different descriptor magnifications used and G is the size of the grid of descriptors that were extracted. Each SIFT descriptor of symbol σ is denoted $D^{\sigma}_{x,y,m}$, where x,y and m are as described in Sect. 2.1. We group the descriptors into $4NG^2$ groups, each SD_g, for a combination of $g = (x, y, m, r)$, containing all the relevant descriptors $D^{\sigma}_{x,y,m}$.

Quantization For each SD_g, a quantization is performed using k-means clustering. For each g, k_g is chosen to be the largest number such that the smallest energy among 1000 runs of k_g-means is smaller than E. For more information about how E was chosen, see Sect. 3. The k-means process is executed 1000 times to insure, with a high probability, that the clustering solution is near optimal (has the smallest energy) and consistent over many runs. The centers of each of the k_g clusters are the quantized descriptors of SD_g and denoted QD_g. Each quantized descriptor $QD \in QD_g$ is assigned a unique identification number, τ. For each τ, we save a mapping, MAP_g^{τ}, to the σs of the descriptors that QD is their quantized descriptor; $id \in MAP_g^{\tau}$ iff $QD_{\tau} \in QD_g$ is the center of the cluster to which $D_{id} \in SD_g$ belongs. We divide all QD_g into four groups, based on r, the form of the letter. Each group serves as the SIFT classifier for that form.

The quantization process is designed to improve the recognition rate. Since there are letters that look similar, their descriptors might also be very close to each other. By quantizing, we allow a letter descriptor $D_{x,y,m}$ to be matched to one $QD_{\tau} \in QD_g$, but since $|MAP_g^{\tau}| \geq 1$, the descriptor can be matched to more than one symbol.

Base Confidence For each symbol, we compute its base confidence. The base confidence is the confidence value returned by executing the classification process described in Sect. 2.4 on the image $L_{i,r}$ that the σ of the symbol was assigned to. Since the base confidence is used to divide the confidence as the last step in the classification process, its initial value is set to 1. Since all additional feature penalties will be equal to 1, the base confidence is actually the SIFT confidence.

In the classification process, the SIFT confidence of the classifier in the symbol σ is divided by its base confidence to create a more “comparable metric” between different symbols of the same form.

2.4 Single Letter Classification

Given an image I, a classifier C and a letter form r, the classification process returns the pair (σ, c), where c is the confidence of the classifier that I contains just the symbol σ.

First, SIFT descriptors and the additional features are extracted as described in Sect. 2.1 and Sect. 2.2, respectively. The grid size, G, and the magnification factors, M, must be the same once that were used to create C. The extracted features of I are: a) SD, the set of descriptors $D_{x,y,m}$, where $x, y \in \{1, \ldots, G\}$ and $m \in M$; b) the additional features $f_m, f_c, f_o, f_t, f_r, f_b, f_l, f_h, f_v$. Next, we execute the following operations:

1. Let P' be an empty list that will hold the predicted σs. The σs in P' can repeat since two descriptors can be matched to the same σ, as can be seen in the next step.
2. For each $D_{x,y,m} \in SD$, we execute the following:
   (a) Find $QD_{\tau} \in QD_g$, where $g = (x, y, m, r)$, such that Euclidean distance between $D_{x,y,m}$ and QD_{τ} is smaller or equal to any other descriptor in QD_g.
   (b) Add all the σs of MAP_g^{τ} to P'.
3. Let P be the set of unique values of P'.
4. For each $\sigma \in P$ execute the following:
   (a) Calculate the additional feature penalties $p_m, p_c, p_o, p_t, p_l, p_b, p_r, p_h, p_v$ as described in Sect. 2.2.

(b) Let the SIFT confidence, p_s, be the number of occurrences of σ in P' divided by $|P'|$.
(c) Let the confidence, c_{id}, of the classifier C in I being the symbol σ, be $p_s p_m p_c p_o p_t p_l p_b p_r p_h p_v/(\text{base confidence of } id)$.

5. The pair (σ, c) is the result of the classification process, where $c = \max_{id \in P} c_\sigma$ and σ is such that $c_\sigma = c$.

2.5 Splitting Words into Paws

The classification process described in Sect. 2.7 requires that the classified image contain a single piece of Arabic word (paw). Given an image containing one line of Arabic text, we split it into paws.

First, we find and label in ascending order, based on the horizontal position of the first pixel, all *8-connectivity* connected components (CCs). Next we group the CCs into "rough" paws by executing the following steps for each CC starting from the one labelled using the smallest label until the one labelled with the largest label:

1. If the CC does not belong yet to any "rough" paw, add the CC to the paw.
2. While there are CCs that vertically intersect with the "rough" paw, add them to the paw. A CC and a paw vertically intersect if there is a column of pixels in the image that contains both pixels that belong to the CC and the paw.

Next, we split each "rough" paw into "regular" paws (referred to as just paws) by first finding and labelling in ascending order, based on the horizontal position of the first pixel, all *4-connectivity* CCs. At this point, each 4-connectivity CC has 2 labels, one 8-connectivity label and one 4-connectivity label. Next, all 4-connectivity CCs that are located on the *baseline* are marked as the anchors of each paw. For each anchor CC, we execute the following steps:

1. Add the anchor CC to the paw.
2. All CCs that have the same 8-connectivity label as the anchor, that either vertically intersect only with the anchor, or do not intersect with any other CC at all, are added to the paw.
3. All CCs that are not an anchor, but intersect by more than $X\%$ with the anchor of the current paw are added to the paw.
4. All other CCs that have the same 8-connectivity are added to the paw only if their width is 10 times smaller or less than the width of the paw. The value 10 was chosen arbitrarily without being optimized for any of the datasets tested in Sect. 3.

Each "rough" paw is eventually split to a number of paws as the number of anchor CCs. The percentage of intersection, X, is a font-specific characteristic that can be a priori calculated for each font. For each font, X will be the minimal amount of vertical intersection that a diacritical dot or a mark has with the other parts of the letter. In this work, we used an intersection percentage of 50% to all fonts but Andalus, which had an intersection percentage of 30%.

2.6 Isolating a Letter Inside a Window

Given a paw and starting and ending positions inside this paw, we would like to isolate the black ink that belongs to the possible letter that starts and ends at those positions. If we will always take all the ink between the starting and ending positions of the window, we might get black ink that belongs to the next or previous letter. Instead of taking all the black ink, we remove black ink that is located on the external side of the borders of the window and on the baseline of the paw. If no black ink is located on the baseline, we remove all the black ink located on the outer side of the border of the window. This process is expected to create three connected components that are located on the baseline. On the image with the cut baseline, we execute the process described in Sect. 2.5, which splits an image into paws. The second paw retuned by this process is the isolated letter. If the starting point of the window is the beginning of the paw, the first paw returned by this process is the isolated letter.

2.7 Paw Classification

The classifications of a paw is one of the main challenges we address in this work. A paw can consist of one or more letters of different forms depending on their location in the paw. Since we do not know where one letter ends and the next letter begins we use a sliding window to scan for letters throughout the paw. A paw can be one of the three types described below:

- **Type 1**: A paw that contains one isolated letter.
- **Type 2**: A paw that starts with initial letter, ends with a final letter and contains zero or more middle letters.
- **Type 3**: A paw that contains two isolated-form letters. This case is very rare and happens when the algorithm that splits words into paws (see Sect. 2.5) fails to split an image containing two isolated letter into two paws of type 1.

A high level overview of the steps of classifying a paw are as follows:

1. Scan for an initial or isolated letter at the beginning of the paw.
2. Scan for a final or isolated letter at the end of the paw.
3. Decide if the paw is of type 1, 2 or 3.
4. If the paw is of type 1 or type 3, then return the best isolated letters and the confidence in them.
5. Otherwise, if the paw is of type 2, scan for a middle or final letter until some final letter is found.
6. Return the list of one initial letter, zero or more middle letters and one final letter. Also return the confidence of the classifier in those letters.

Steps 1, 2 and 3 are explained in detail in Sect. 2.7; step 5 is explained in detail in Sect. 2.7. The scanning procedure in steps 1, 2 and 5 is described in Sect. 2.7.

Scanning Given a letter form r and a starting point inside the paw, scanning is done by classifying a set of windows of increasing sizes, starting at the given point. Inside each window, a letter is isolated as described in Sect. 2.6 and then classified using the classifier of form r as described in Sect. 2.4. Each window of size s is assigned with the result of the classifier on it, (σ_s, c_s), and the following is executed:

1. Let B be the window size that has the highest classifier confidence c_B.
2. Let σ_B be the σ that was assigned by the classifier to the window of size B.
3. Return σ_B and a list of all pairs (p, c_s), where p is the ending point of a window that the classifier assigned the letter σ_B and c_s is the confidence of the classifier for the isolated letter inside the window to be σ_B.

Scanning is done by increasing the window size from the starting point either towards the end of the paw or towards the beginning of the paw. The latter scanning is used when scanning for the best starting point of a final or isolated letter that end exactly at the end of a paw.

Scanning for Initial and a Final Letter or an Isolated Letter We execute this kind of scan to identify the type of the paw. First we scan for an initial letter candidate that starts at the beginning of a paw and a final letter candidate that ends at the end of the paw, assuming the paw is of type 2:

1. Scan for σ_0 and $P_0 = \{(p_1, c_1), \ldots, (p_n, c_n)\}$, where σ_0 is the σ of the best initial letter starting at the beginning of the paw and P_0 are the possible ending positions and confidences of the letter to end at those positions.
2. Scan for σ_* and $P_* = \{(p_1, c_1), \ldots, (p_n, c_n)\}$, where σ_* is the σ of the best final letter ending at the end of the paw and P_* are the possible starting positions and confidences of the letter to start at those positions.

Second, we scan for an isolated letter candidate that begins at the beginning of the paw and an isolated letter candidate that ends at the end of the paw, assuming the paw is either of type 1 or type 3:

1. Scan for $\sigma_{@..}$ and $P_{@..} = \{(p_1, c_1), \ldots, (p_n, c_n)\}$, where $\sigma_{@..}$ is the σ of the best isolated letter starting at the beginning of the paw and $P_{@..}$ are the possible ending positions and confidences of the letter to end at those positions.
2. Scan for $\sigma_{..@}$ and $P_{..@} = \{(p_1, c_1), \ldots, (p_n, c_n)\}$, where $\sigma_{..@}$ is the σ of the best isolated letter ending at the end of the paw and $P_{@_e b}$ are the possible starting positions and confidences of the letter to start at those positions.

Third, we calculate some intersection ratios between the windows of the candidates as follows:

1. Let the intersection ratio, $I_@$, between the isolated letters, $\sigma_{@..}$ and $\sigma_{..@}$, be twice the number of pixels shared by the windows with the highest confidence of $\sigma_{@..}$ and $\sigma_{..@}$ divided by the sum of the window sizes.

2. Let the intersection ratio, I_{0*}, between the initial and final letters, be twice the number of pixels shared by the windows with the highest confidence of σ_0 and σ_* divided by the sum of the window sizes.
3. Let the "unclassified ratio", $I_{@@}$, be one minus the ratio of the number of pixels not covered by the windows with the highest confidence of $\sigma_{@..}$ and $\sigma_{..@}$ and the width of the paw.

Fourth, we calculate the confidence of a paw to be one of the three possible types:

- The confidence of a paw to be of type 1 is $C_1 = c_{@..}c_{..@}I_{@}$
- The confidence of a paw to be of type 2 is $C_2 = c_0c_*(1 - I_{0*})$
- The confidence of a paw to be of type 3 is $C_3 = c_{@..}c_{..@}I_{@}(1 - I_{@})$
- For C_3 to be taken into account, it has to be significantly bigger than C_2, i.e. if $C_2/C_3 > 0.9$, then $C_3 = 0$. The value 0.9 was arbitrary.

Finally, based on the type confidences, we decide whether to return the isolated letter or letters that we found or to continue and scan for medial letters that are located between the initial and the final letter that we found:

1. If C_3 is the largest confidence, return $\sigma_{@..}, \sigma_{..@}$, the σs of the 2 isolated letters that were found.
2. If C_1 is the largest confidence and $\sigma_{@..} = \sigma_{..@}$, return $\sigma_{@..}$.
3. If C_1 is the largest confidence and $\sigma_{@..} \neq \sigma_{..@}$, choose the better isolated letter. Since the paw is a single isolated letter, taking the confidences of both possible isolated letters is not enough. It should be taken into account how many pixels were left outside the best window for each isolated letter. Based on that, a revised confidence for each isolated letter is calculated:
   (a) $c_{@..} = c_{@..}$ (size of the window that was classified to be $\sigma_{@..}$ and confidence $c_{@..}$)
   (b) $c_{..@} = c_{..@}$ (size of the window that was classified to be $\sigma_{..@}$ and confidence $c_{..@}$)

   If $c_{@..} > c_{..@}$, return $\sigma_{@..}$, otherwise return $\sigma_{..@}$.
4. If C_2 is the largest confidence continue scanning for medial letters as described in Sect. 2.7.

Scanning for Medial Letters We scan for medial letters if the paw was classified as type 2. At this point, we know the initial letter, σ_0, and it possible ending positions and the confidences for the letter to end at those positions, P_0. We also know the final letter that ends at the end of the paw, σ_*, and its possible starting positions and the confidences for it to start at those positions, P_*. We do the scanning for medial letters until we find a final letter as follows:

1. Let p_f be the first possible starting position of σ_*.
2. Let p_l be the last possible starting position of σ_*.
3. Let p_{pre} be the position where the previous letter has the highest confidence for ending. At the first iteration of this process, the previous letter is σ_0, while on the next iterations, this letter is the previous medial letter.

4. For windows ending before p_l, scan for σ_m and $P_m = \{(p_1, c_1), \ldots, (p_n, c_n)\}$, where σ_m is the σ of the best middle letter starting at p_{pre} and P_m in the set of possible ending positions and confidences of the letter to end at those positions. Let $(p_m, c_m) \in P_m$ be the ending position and the confidence for σ_m to end at this position, where c_m is the highest confidence among all possible ending positions.
5. For windows ending after p_f, scan for σ_{b*} and $P_{b*} = \{(p_1, c_1), \ldots, (p_n, c_n)\}$, where σ_{b*} is the σ of the best final letter starting at p_{pre} and P_{b*} is the possible ending positions and confidences of the letter to end at those positions. Let $(p_{b*}, c_{b*}) \in P_{b*}$ be the ending position and confidence for σ_{b*} to end at this position, where c_{b*} is the highest confidence among all possible ending positions.
6. Let the intersection ratio, I_{m*}, between the middle letter, σ_m, and the final letter, σ_*, be twice the number of pixels shared by the windows with the highest confidence of σ_m and σ_* divided by the sum of the window sizes. Update c_m to be $c_m\sqrt{1 - I_{m*}}$.
7. Let the intersection ratio, I_{**}, between the final letter, σ_{b*}, and the final letter, σ_*, be twice the number of pixels shared by the windows with the highest confidence of σ_{b*} and σ_* divided by the sum of the window sizes. Update c_{fl_b} to be $c_{fl_b}\sqrt{I_{**}}$.
8. If the SIFT confidence part, p_s, of the higher confidence between c_m and c_{b*} is less than 0.9 (the value 0.9 was chosen arbitrary without being optimized for any of the datasets tested in Sect. 3), retry and scan starting from all possible ending positions of the previous letter as follows:
   (a) Normalize the confidences of all ending points of the previous letter by dividing them by the value of the maximal confidence.
   (b) Repeat steps 3–7 to get σ_m, c_m and σ_{b*}, c_{b*}, the best middle and final letters and their confidences starting at all possible ending points of the previous letter.
   (c) For each possible ending point of the previous letter, multiple c_m and c_{b*} by the normalized confidence of the previous letter ending at this position.
   (d) Choose σ_m, $P_m = \{(p_1, c_1), \ldots, (p_n, c_n)\}$ and σ_{b*}, $P_{b*} = \{(p_1, c_1), \ldots, (p_n, c_n)\}$ to be the ones with the highest c_m and c_{fl_b} among all possible ending positions of the previous letters.
9. If $c_m > c_{fl_b}$, save σ_m as the next letter and scan, stating from step 3, for the next letter starting at the position, where the medial letter found has the highest confidence to end.
10. Otherwise if $c_m \leq c_{fl_b}$, return the initial letter σ_0, all medial letters and the final letter that has the higher confidence. If $c_{b*} > c_*$ return σ_{b*}; otherwise return σ_*.

3 Experimental Results

To test the performance of SOCR, two different datasets were used. First, in Sect. 3.2, we describe how the PATS dataset was constructed [3] and compare the results of the algorithm suggested by the author of PATS to the results of SOCR. then, in Sect. 3.3, we describe how the APTI benchmark was constructed [12] and provide some initial tests results we did on parts of the dataset.

Common Parameter Configuration

Although all the parameters used in SOCR can be configured, the parameters that a priori cannot be automatically computed were the same for all datasets and fonts. The following list contains the common configuration of parameters:

- G – the number grid elements, SIFT descriptors, in each spatial direction extracted from each given image was set to 3.
- M – the set of descriptor magnification factors was set to $\{0.5, 1.0, 1.5\}$.
- E – the energy used to compute the optimal k for k-means clustering of the descriptors as described in Sect. 2.3 was set to 10^4.

3.1 Performance Metrics

The performance is reported in terms of character recognition rate (CRR) and word recognition rate (WRR). CRR was measured using the Levenshtein edit distance [7] between the predicted word and ground truth. WRR is the ratio of the number of words having all its letter recognized correctly to the total number of words. WRR is an important performance parameter for algorithms that take a language-specific approach, such as using word lists, for training or for improving the results of the classification process. Since we do not use a language-specific approach, WRR is less important for our algorithm and reported only in cases where we were able to split the input images into words.

3.2 PATS Dataset

The PATS dataset consists of 2766 text-line images that were selected from two standard classic Arabic books. The images were split into images that each contain one line. The dataset includes the fonts Arial, Tahoma, Andalus, Simplified Arabic, Akhbar, Thuluth, Naskh and Traditional Arabic. Lines written using Arial, Tahoma, Andalus and Simplified Arabic use 41 different letters and ligatures (see Table 1a). Akhbar uses three more symbols, while Thuluth, Naskh and Traditional Arabic use several more symbols that were not taken into consideration when creating the classifier (see Table 1b).

Using tables like 1a and 1b, a classifier was constructed for each font, as described in Sect. 2.3. Using the classifier, the SOCR algorithm was executed on all 2766 lines of each font. See Fig. 4 for a sample line of Tahoma, Andalus and Thuluth fonts.In Table 2 the performance, in terms of CRR and WRR, is reported for all 2766 lines and the last 266 lines, along side with the CRR of executing on the last 266 lines the algorithm suggested by the author of PATS. SOCR outperforms the algorithm suggested by the author of PATS on Tahoma, Arial, Andalus, Simplified Ararbic and Akhbar fonts. The classifier constructed

Table 1: Tables of unique symbols that are used to construct classifiers.

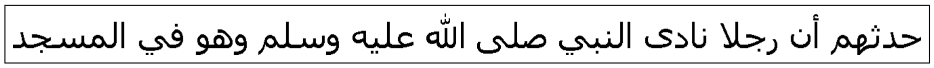
(a) Arial

		ﻮ	و	ﺸ	ﺷ	ﺶ	ش			ﺎ	ا
ﻴ	ﻳ	ﻲ	ي	ﺼ	ﺻ	ﺺ	ص			ﻰ	ى
		ﻼ	ﻻ	ﻀ	ﺿ	ﺾ	ض	ﺒ	ﺑ	ﺐ	ب
		ﺄ	أ	ﻄ	ﻃ	ﻂ	ط	ﺘ	ﺗ	ﺖ	ت
			ﷲ	ﻈ	ﻇ	ﻆ	ظ			ﺔ	ة
			ء	ﻌ	ﻋ	ﻊ	ع	ﺜ	ﺛ	ﺚ	ث
		ﺂ	آ	ﻐ	ﻏ	ﻎ	غ	ﺠ	ﺟ	ﺞ	ج
		ﺈ	إ	ﻔ	ﻓ	ﻒ	ف	ﺤ	ﺣ	ﺢ	ح
		ﺆ	ؤ	ﻘ	ﻗ	ﻖ	ق	ﺨ	ﺧ	ﺦ	خ
ﺌ	ﺋ	ﺊ	ئ	ﻜ	ﻛ	ﻚ	ك			ﺪ	د
		ﻶ	ﻵ	ﻠ	ﻟ	ﻞ	ل			ﺬ	ذ
		ﻸ	ﻷ	ﻤ	ﻣ	ﻢ	م			ﺮ	ر
		ﻺ	ﻹ	ﻨ	ﻧ	ﻦ	ن			ﺰ	ز
				ﻬ	ﻫ	ﻪ	ه	ﺴ	ﺳ	ﺲ	س

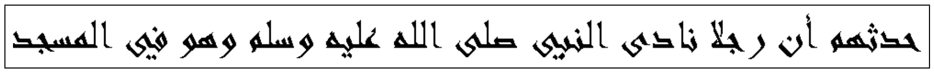
(b) Naskh

		ﻼ	ﻻ	ﺼ	ﺻ	ﺺ	ص			ﺎ	ا
		ﺄ	أ	ﻀ	ﺿ	ﺾ	ض			ﻰ	ى
			ﷲ	ﻄ	ﻃ	ﻂ	ط	ﺒ	ﺑ	ﺐ	ب
			ء	ﻈ	ﻇ	ﻆ	ظ	ﺘ	ﺗ	ﺖ	ت
		ﺂ	آ	ﻌ	ﻋ	ﻊ	ع			ﺔ	ة
		ﺈ	إ	ﻐ	ﻏ	ﻎ	غ	ﺜ	ﺛ	ﺚ	ث
		ﺆ	ؤ	ﻔ	ﻓ	ﻒ	ف	ﺠ	ﺟ	ﺞ	ج
ﺌ	ﺋ	ﺊ	ئ	ﻘ	ﻗ	ﻖ	ق	ﺤ	ﺣ	ﺢ	ح
		ﻶ	ﻵ	ﻜ	ﻛ	ﻚ	ك	ﺨ	ﺧ	ﺦ	خ
		ﻸ	ﻷ	ﻠ	ﻟ	ﻞ	ل			ﺪ	د
		ﻺ	ﻹ	ﻤ	ﻣ	ﻢ	م			ﺬ	ذ
			ﻟﺎ	ﻨ	ﻧ	ﻦ	ن			ﺮ	ر
			ﻟﻰ	ﻬ	ﻫ	ﻪ	ه			ﺰ	ز
			ﻓﻲ			ﻮ	و	ﺴ	ﺳ	ﺲ	س
				ﻴ	ﻳ	ﻲ	ي	ﺸ	ﺷ	ﺶ	ش

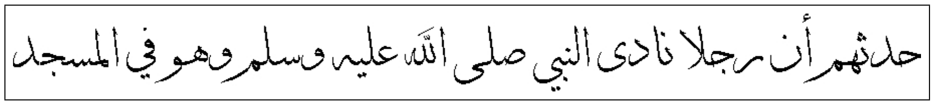
for fonts Thuluth, Naskh and Traditional Arabic was constructed using a table of 44 unique letters and combination of letters, but the lines in PATS dataset included some more combinations. This resulted in a constant failure of the classifier on those missing combinations and a poor CRR performance.


حدثهم أن رجلا نادى النبي صلى الله عليه وسلم وهو في المسجد

(a) Tahoma


حدثهم أن رجلا نادى النبي صلى الله عليه وسلم وهو في المسجد

(b) Andalus


حدثهم أن رجلا نادى النبي صلى الله عليه وسلم وهو في المسجد

(c) Thuluth

Fig. 4: Line 88 of various fonts in the PATS dataset.

Table 2: Performance in terms of CRR and WRR of SOCR and the algorithm suggested by the author of PATS executed on PATS dataset.

Font	SOCR CRR-266	PATS CRR-266	SOCR CRR-2766	SOCR WRR-2766
Tahoma	**100.0%**	99.68%	100.0%	100.0%
Arial	**99.98%**	99.90%	99.96%	99.90%
Andalus	**98.87%**	97.86%	98.58%	94.90%
Simplified Arabic	**99.72%**	99.70%	99.73%	99.00%
Akhbar	**99.83%**	99.34%	99.89%	99.72%
Thuluth	87.23%	**97.78%**	86.16%	N/A
Naskh	87.38%	**98.09%**	85.69%	N/A
Traditional Arabic	92.56%	**98.83%**	91.53%	N/A

3.3 APTI Dataset

The Arabic printed text image database (APTI) [12], was created to address the challenges of optical character recognition of printed Arabic text of multiple fonts, multiple font sizes and multiple font styles. APTI is designed for the evaluation of screen-based OCR systems. The authors suggested numerous OCR

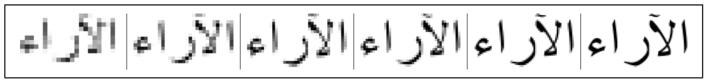


Fig. 5: A word in Arabic Transparent and sizes 6, 8, 10, 12, 18 and 24 (from left to right).

evaluation protocols and conducted a competition [13] to test the performance of state of the art OCR systems. The dataset is split in to six sets, where five of the six sets of the dataset were publicly available for research and training, while set number 6 was used for evaluating the submitted systems and known only to the creators of APTI.

Table 3 shows the performance comparison, in terms of CRR and WRR, of SOCR and the systems submitted to the competition running the first APTI protocol. The first protocol tests the ability of a system to recognize the images of words written using Arabic Transparent font (very similar to Arial) in six different sizes (6, 8, 10, 12, 18 and 24). See Fig. 5 for a sample image of a word in the dataset in Traditional Arabic and the six fonts sizes used. Each system is tested on all six sizes, while the size is known to the system. Three different systems, IPSAR, UPV-PRHLT and DIVA-REGIM, were evaluated in the competition using the first protocol on the unpublished set number 6, while the latter system was declared out of the competition since it was built by the creators of APTI and optimized for more than a year on the first five sets. The other two systems also used the first fives sets for training. Since set number 6 is not publicly available, SOCR was evaluated (on the randomly chosen) set number 4 [11, No. 221] in two different modes. The first mode considers the variations of the letter alif (no diacritic marks, hamza above, hamza below and tilda above) as different letters, while the second mode considers all the variation of alif as the same letter. The approach of the second mode, measuring the quality of Arabic OCR while considering the different variations of the letter alif as the same letter, was previously suggested in [10]. Since SOCR assumes that the input images are black ink on white background, each image was converted to a black and white image using a dynamic threshold. Assuming that the value 0 represents a black pixel and value 1 represented a white pixel, the dynamic threshold was calculated as follows:

- Let s be the standard deviation of the pixel values in the image.
- Let m be the mean value of all the pixels that are smaller than $1 - s$.
- Let $t = m + s$ be the dynamic threshold.

All pixels smaller than t are transformed into 0, while all pixels larger or equal to t are transformed into 1. The APTI dataset uses an alphabet of 43 uniquely represented letters and combination of letters for Arabic.

From Table 3 it can be seen that SOCR performs competitively with the other systems, mostly on the larger font sizes. It is important to mention that all the other three systems used the first five sets to train their classifiers, while SOCR did not perform any training using those sets. SOCR also used the same classifier for all font sizes. The difference in performance between the first and second mode shows that a training phase can be used to create an SOCR classifier for each font size to significantly improve the performance when running in the first mode. The training phase can fine tune the SIFT descriptors for each font size of the different variations of the letter alif.

Table 3: Performance in terms of CRR and WRR of SOCR and the algorithm suggested by the author of PATS executed on PATS dataset.

System/Size		6	8	10	12	18	24
SOCR	WRR	23.5%	61.9%	63.5%	71.2%	84.0%	97.0%
	CRR	64.7%	90.1%	92.7%	93.2%	97.1%	99.2%
SOCR	WRR	27.6%	78.9%	89.8%	94.0%	99.0%	98.5%
Ignore Alif Variation	CRR	68.2%	94.4%	97.5%	97.6%	99.8%	99.6%
IPSAR	WRR	5.7%	73.3%	75.0%	83.1%	77.1%	77.5%
	CRR	59.4%	94.2%	95.1%	96.9%	95.7%	96.8%
UPV-PRHLT	WRR	94.5%	97.4%	96.7%	92.5%	84.6%	84.4%
	CRR	99.0%	99.6%	99.4%	98.7%	96.9%	96.0%
DIVA-REGIM	WRR	86.9%	95.9%	95.7%	93.9%	97.9%	98.9%
	CRR	98.0%	99.2%	99.3%	98.8%	99.7%	99.7%

4 Conclusion

We have seen how SIFT descriptors can be successfully used as features of individual letters to perform OCR of Arabic printed text. We overcame the challenge of printed Arabic being a cursive text by performing, jointly, segmentation into letters and their recognition. While enjoying the benefit of not having a need for a training set, our method performs competitively compared to other, recently purposed methods, which require large training sets. More work can be done to address the scenarios where the method showed a relatively high failure rate. In the situation where the method fails to distinguish between the variations of the letter alif, which differ only in diacritical marks, post-processing can be used to correct the misclassification by matching the recognized word against a pre-defined list of words. In the situation where the method fails due to a failure to split two paws, which happens mostly with low resolution fonts, this post-processing might not suffice, since the probability that most of the letters of the second paw will not be recognized correctly is high.

Since our classifier consists of four classifiers, one for each possible location of a letter inside a paw, creating one classifier for all forms can help overcome the failures that happen when a split fails. On top of that, introducing a learning phase can, potentially, improve performance by finding the best weights for penalties and their combination with the segmentation and recognition scores.

Also, image-based verification can be added. By generating an image of the predicted word or paw, we can measure its visual similarity to the word we are trying to recognize. Future work should focus on a larger variety of fonts and sizes and the algorithm should be extended to work well on texts containing multiple fonts.

References

1. Ahmad, R., Amin, S.H., Khan, M.A.: Scale and rotation invariant recognition of cursive Pashto script using SIFT features. 6th International Conference on Emerging Technologies (2010) 299–303
2. Ahmed, I., Mahmoud, S.A., Parvez, M.T.: Printed Arabic text recognition. In: Märgner, V., El Abed, H. (eds.): Guide to OCR for Arabic Scripts. Springer, London (2012) 147–168
3. Al-Muhtaseb, H.A., Mahmoud, S.A., Qahwaji, R.S.R.: Recognition of off-line printed Arabic text using Hidden Markov Models. Signal Processing **88** (2008) 2902–2912
4. Diem, M., Sablatni, R.: Recognition of degraded handwritten characters using local features. International Conference on Document Analysis and Recognition (2009) 221–225
5. Fujisawa, H.: Forty years of research in character and document recognition-an industrial perspective. Pattern Recognition **41** (2008) 2435–2446
6. Gui, J.P., Zhou, Y., Lin, X.D., Chen, K., Guan, H.B.: Research on Chinese character recognition using bag of words. Applied Mechanics and Materials **20–23** (2010) 395–400
7. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady **10** (1966) 707–710
8. Lowe, D.G.: Object recognition from local scale-invariant features. In: Proceedings of the International Conference on Computer Vision. ICCV '99, Vol. 2. (1999) 1150–1
9. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision **42** (2004) 91–110
10. Magdy, W., Darwis, K.: Arabic OCR error correction using character segment correction, language modeling, and shallow morphology. In: Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing. (2006) 408–411
11. Munroe, R.: (Random number) xkcd.com/221.
12. Slimane, F., Ingold, R., Kanoun, S., Alimi, A., Hennebert, J.: A new Arabic printed text image database and evaluation protocols. In: International Conference on Document Analysis and Recognition. (2009) 946–950
13. Slimane, F., Kanoun, S., Abed, H.E., Alimi, A.M., Ingold, R., Hennebert, J.: Arabic recognition competition: Multi-font multi-size digitally represented text. In: Eleventh International Conference on Document Analysis and Recognition. IEEE (2011) 1449–1453
14. Steinherz, T., Rivlin, E., Intrator, N.: Offline cursive script word recognition – a survey. International Journal on Document Analysis and Recognition **2** (1999) 90–110
15. Wu, T., Qi, K., Zheng, Q., Chen, K., Chen, J., Guan, H.: An improved descriptor for Chinese character recognition. Third International Symposium on Intelligent Information Technology Application (2009) 400–403

16. Zahedi, M., Eslami, S.: Farsi/Arabic optical font recognition using SIFT features. Procedia Computer Science **3** (2011) 1055–1059
17. Ziaratban, M., Fae, K.: A novel two-stage algorithm for baseline estimation and correction in Farsi and Arabic handwritten text line. 19th International Conference on Pattern Recognition (2008)