TEL AVIV UNIVERSITY אוניברסיטת תל-אביב

The Raymond and Beverly Sackler Faculty of Exact Sciences

The Blavatnik School of Computer Science

Using SIFT Descriptors for OCR of Printed Arabic

This thesis is submitted in partial fulfilment

of the requirements for the degree of Master of Science

by

Andrey Rosenberg

This work was carried out under the supervision of

Prof. Nachum Dershowitz

February 2012

Acknowledgements

I wish to thank my advisor, Prof. Nachum Dershowitz, for his guidance and patience throughout the process of researching and writing this work.

I thank my parents for the inspiration they put in me to always strive for the very best.

I thank my beautiful wife for the mental support and patience.

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 propose a method that combines the two tasks, using multiple grids of SIFT descriptors as features. To construct a classifier, we don't 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" (paws), and each paw 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 paw. 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.

Contents

Chapter 1

Introduction

After more than forty years of research, *optical character recognition* (OCR) systems for machine-printed text show impressive performance [4]. 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 Arabic word* (paws). The letters inside a paw are connected 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. (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 Figure 1.1). (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

Figure 1.1: Initial, medial, final and isolated forms of the letter hā' in the Arial font.

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 doesn't 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 didn't use language models, morphological information, word lists or any other language resources that are commonly used to improve performance.

The remainder of this thesis is organized as follows. In Section 1.1, we review related work. In Chapter 2, we present the SOCR algorithm. In Chapter 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 Chapter 4.

1.1 Related Work

Recently, SIFT descriptors [7] have been suggested for use in OCR. They were used for Chinese character recognition in [14] and [5], and were applied to degraded handwritten Latin manuscripts in [3]. 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 [8] as centres of the SIFT descriptors. SIFT descriptors were also used to recognize the font of an Arabic printed text [15].

As been shown in a survey done more than a decade ago [13], Hidden Markov Model (HMM) serve as a base for most of the methods that perform recognition of cursive scripts. The author of PATS dataset [2], the first dataset we used to test our algorithm, uses HMM and a sliding window to segment and recognize Arabic scripts. In a competition [12] recently conducted by the authors of APTI dataset [11], the second data set we used to test our algorithm, participated two HMM based systems. The system that was suggested by the authors themselves didn't 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.

Chapter 2

SOCR Algorithm

Our algorithm, called *SOCR* for "SIFT-based OCR", that segments and recognizes the letters of an image containing a single paw is described in Section 2.6. Using a sliding-window technique a candidate letter is isolated inside each window (see Section 2.5) and classified using the appropriate classifiers (see Section 2.3). 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 Section 2.2). 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 Section 2.4, 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 Section 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 Section 3.2. For each input image, we estimate the baseline as the row having the most black ink among all rows. The baseline is needed to correctly split the image to paws (see Section 2.4) and to isolate correctly a letter inside a window see Section 2.5). A more accurate baseline estimation that was designed for handwritten texts and considers the diacritical dots and marks of Arabic [16] was considered, but

not used in this work.

2.1 Feature Extraction

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 Section 2.2. Those features are later compared to the features extracted from the image of a letter that we are trying to classify as described in Section 2.3.

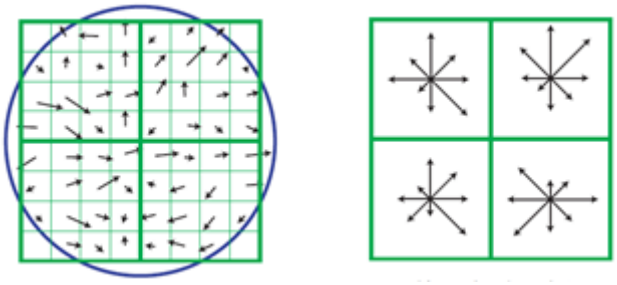
We use a multiple grids of SIFT descriptors as the main feature set. The structure and extraction process of a single descriptor are described in Section 2.1.1. To fine-tune the classification results produced by the use of SIFT descriptors, we use additional features. The structure and extraction process of those additional features are described in Section 2.1.2.

2.1.1 SIFT Descriptors

SIFT descriptors are local feature descriptors. They were designed for object-recognition purposes [7]. They were proven to be effective when extracted around points of interest [8]. A single sift descriptor is a 128-dimensional vector of bytes. A high level overview of the extraction process of a single SIFT descriptor is as follows:

1. Given a center, the gradients and their magnitudes are computed in an area around the center of the descriptor.
2. The magnitudes are weighted by a Gaussian window.
3. The area around the descriptor is split into 4×4 spatial bins. The size of each spatial bin (in pixels) is the *scale* of the descriptor.
4. For each spatial bin, a histogram of 8 orientation bins, weighted by the magnitudes is calculated. The area that is taken into account is defined by the scale of the descriptor and a *magnification factor*. This results in a $4 \times 4 \times 8 = 128$-dimensional descriptor.

Figure 2.1a shows the gradients of an area around the center of a descriptor that is split into 2×2 spatial bins. Figure 2.1b shows the 8 bin histogram calculated for each spatial bin resulting in a 32-dimensional SIFT descriptor.



(a) Gradients and magnitudes (arrows) weighted by a Gaussian window (blue circle)

(b) 8 bin orientations histogram (arrows) for each of the 4 spatial bins (green bold squares)

Figure 2.1: Extraction a 32-dimensional SIFT descriptor using 2 bins in each spatial direction and a histogram of 8 orientation bins.

2.1.1.1 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 $N_{Grid} \times N_{Grid}$ identical squares, where N_{Grid} 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_{Magnif} descriptors, where $M = \{m_1, \ldots, m_{N_{Magnif}}\}$ are the magnification factors.

5. Each extracted descriptor is identified by $D_{x,y,m}$, where x, y are the indices 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, $N_{Grid} = 3$ and $M = \{0.5, 1.0, 1.5\}$. Figure 2.2 shows an example of a grid of SIFT descriptors that were extracted without using magnification.

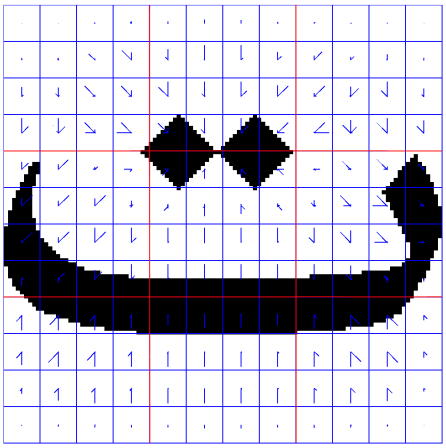


Figure 2.2: A grid of 3×3 SIFT descriptors of the letter tā' of isolated/final form. Each descriptor is separated by a red line.

2.1.1.2 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 [8]. While this method might perform well for complex images, it achieved very low performance on Arabic letters and had major disadvantages: a) The number of keypoints depends on the resolution. b) For some simple letters, the method didn't 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 Section 2.2.3.

2.1.2 Additional Features

Additional features are used to penalize the confidence of letters that are suggested by the SIFT classifier (see Section 2.3). 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.

2.1.2.1 Center of Mass Feature

The *center of mass feature*, $\bar{f}_m = (x, y)$, 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 Figure 2.3. Given an image and a letter, where $\bar{f}'_m$ and

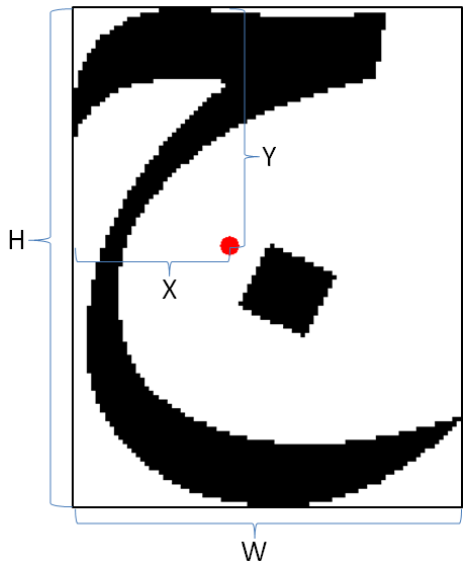


Figure 2.3: The center of mass of the letter jīm is marked by the red dot. The center of mass feature is $(X/W, Y/H)$.

$\bar{f}_m$ are their center of mass features, respectively, the *center of mass penalty* used is $p_m = 1/\left(1 + d(\bar{f}_m, \bar{f}'_m)^{\frac{1}{2}}\right)$. The exponent was arbitrary set to be 1/2 without being optimized for any of the datasets tested in Chapter 3.

2.1.2.2 Crosshair Feature

The *crosshair feature*, $\bar{f}_c = (x, y)$, 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 Figure 2.4. Given an image and a letter, where $\bar{f}'_c$ and $\bar{f}_c$ are their corsshair features, respectively, the *crosshair penalty* used is $p_c = 1/\left(1 + d(\bar{f}_c, \bar{f}'_c)^{\frac{1}{2}}\right)$. The exponent was arbitrary set to be 1/2 without being optimized for any of the datasets tested in Chapter 3.

2.1.2.3 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 f'_o and f_o are their ratio features, respectively, the *ratio penalty* used is $p_o = 1/\left(1 + (f'_o - f_o)^2\right)$. The exponent was arbitrary set to be 2 without being optimized for any of the datasets tested in Chapter 3.

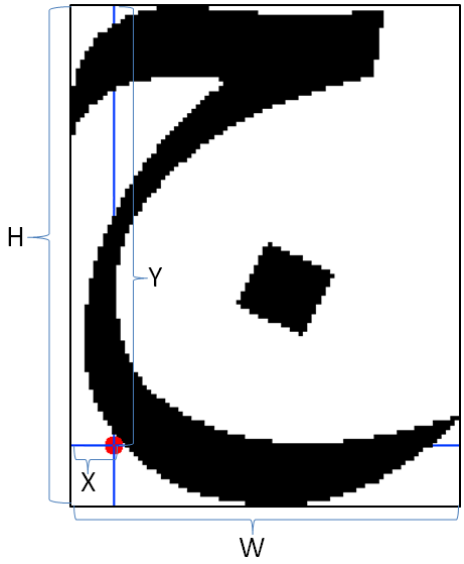


Figure 2.4: The crosshair of the letter jīm is marked by the red dot. The crosshair feature is $(X/W, Y/H)$.

2.1.2.4 Outline Features

Each image has four *outline features*, top, left, bottom and right. The top-outline feature, $\bar{f}_t = \left(f_t^{(1)}, \ldots, f_t^{(W)}\right)$, where W is the width of the bounding box of the black ink, is calculated as follows:

1. For i from 1 up to W, let $f_t^{(i)}$ be the distance from the top of the bounding box to the first occurrence of a black pixel on ith column of the image.
2. Let m be maximum$\bar{f}_t$.
3. For i from 1 up to W, let $f_t^{(i)}$ be $\left|f_t^{(i)} - m\right|$.
4. Let m be maximum$\bar{f}_t$.
5. For i from 1 up to W, let $f_t^{(i)}$ be $f_t^{(i)}/m$.

The left ($\bar{f}_l$), bottom ($\bar{f}_b$) and right ($\bar{f}_r$) outline features are calculated in a similar manner. The top-outline of the letter jīm is shown in Figure 2.5.

Given an image and a letter, where $\bar{f}'_t$ and $\bar{f}_t$ are its top-outline features, respectively, the *top-outline penalty*, p_t, is calculated as follows:

1. Let n be the size of $\bar{f}_t$ and n' be the size of $\bar{f}'_t$ (without loss of generality $n > n'$).
2. Let $\bar{f}''_t = \left(f_t''^{(1)}, \ldots, f_t''^{(n')}\right)$ be the downscaled version of $\bar{f}_t$ by taking n' elements from $\bar{f}_t$, where $f_t''^{(i)} = f_t^{([i \cdot n/n'])}$.

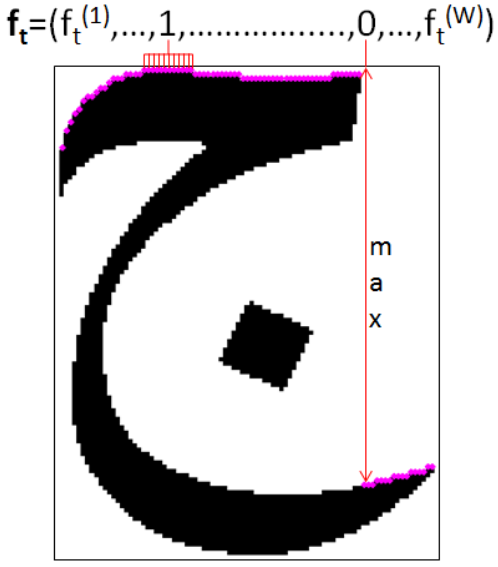


Figure 2.5: The top-outline of the letter jīm is marked by the purple dots. The top-outline feature is $\left(f_t^{(1)},\ldots,f_t^{(W)}\right)$.

3. Define $p_t = 1/\left(1+\frac{1}{n'}\sum_{i=1}^{n'}\left|f_t''^{(i)}-f_t'^{(i)}\right|\right)$

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

2.1.2.5 Black Ink Histogram Features

Each image has a horizontal *black ink histogram feature* and a vertical one. The horizontal black ink histogram feature, $\bar{f}_h = (f_h^{(1)},\ldots,f_h^{(H)})$, where H is the height of the bounding box of the black ink, is calculated as follows:

1. For i from 1 up to H, let $f_h^{(i)}$ be the number of black ink pixels in row i.
2. Let m be maximum $\bar{f}_h$.
3. For i from 1 up to H, let $f_h^{(i)}$ be $f_h^{(i)}/m$.

The vertical black ink histogram feature ($\bar{f}_v$) is calculated in a similar manner. The black ink histograms of the letter jīm are shown in Figure 2.6.

Given an image and a letter, where $\bar{f}_h'$ and $\bar{f}_h$ are its horizontal black ink histogram features, respectively, the horizontal *black ink histogram penalty*, p_h, is calculated as follows:

1. Let n be the size of $\bar{f}_h$ and n' be the size of $\bar{f}_h'$ (without loss of generality $n > n'$).

Figure 2.6: The vertical and horizontal black ink histograms of the letter jīm.

2. Let $\bar{f}''_h = \left(f''^{(1)}_h, \ldots, f''^{(n')}_h\right)$ be the downscaled version of $\bar{f}_h$ by taking n' elements from $\bar{f}_h$, where $f''^{(i)}_h = f^{([i \cdot n/n'])}_h$.

3. $p_h = 1/\left(1 + \frac{1}{n'} \cdot \sum_{i=1}^{n'} \left| f''^{(i)}_h - f'^{(i)}_h \right|\right)$.

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

2.2 Constructing a Classifier

On each classifier C for font F and an alphabet Σ, we execute a series of operations as described below. In Section 2.2.1, we explain how to generate high-resolution images, each one containing a unique symbol of the alphabet Σ written in font F. In Section 2.2.2, 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 Section 2.2.3, 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 Section 2.2.4, we compute the *base confidence* for each unique symbol.

2.2.1 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 don't 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 Chapter 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.

2.2.2 Feature Extraction

Before we extract features, we assign to each $L_{i,r}$ a unique identification number, *id*. We extract SIFT descriptors and additional features as described in Sections 2.1.1 and 2.1.2, respectively. The number of extracted SIFT descriptors per symbol is $N_{Magnif} \cdot N_{Grid}^2$, where, as we said, N_{Magnif} is the amount of different descriptor magnifications used and N_{Grid} is the size of the grid of descriptors that were extracted. Each SIFT descriptor of the symbol *id* is denoted by $D_{x,y,m}^{(id)}$, where x,y and m are as described in Section 2.1.1.1. We group the descriptors into $4 \cdot N_{Magnif} \cdot N_{Grid}^2$ groups. A group for each combination of $g = (x, y, m, r)$, SD_g, contains all the descriptors $D_{x,y,m}^{(id)}$ of all *id*s, where *id* is the unique identification that was assigned to an image $L_{i,r}$.

2.2.3 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 Chapter 3. The process k-means is executed 1000 time 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 by SQD_g. Each quantized descriptor $QD \in SQD_g$ is assigned a unique identification number, *qid*. For each *qid*, we save a mapping, $MAP_g^{(qid)}$, to the *id*s of the descriptors that QD is their quantized descriptor; $id \in MAP_g^{(qid)}$ iff $QD_{qid} \in SQD_g$ is the center of the cluster to which $D_{id} \in SD_g$ belongs. We divide all SQD_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 $QD_{qid} \in SQD_g$, but since $|MAP_g^{(qid)}| \geq 1$, the descriptor can be matched to more than one symbol.

2.2.4 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 Section 2.3 on the image $L_{i,r}$ that the *id* 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 *id* is divided by its base confidence to create a more "comparable metric" between different symbols of the same form.

2.3 Single Letter Classification

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

First, SIFT descriptors and the additional features are extracted as described in Section 2.1.1 and Section 2.1.2, respectively. The grid size, N_{Grid}, 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 $x, y \in \{1, \ldots, N_{Grid}\}$ and $m \in M$; b) the additional features $\bar{f}_m, \bar{f}_c, f_o, \bar{f}_t, \bar{f}_r, \bar{f}_b, \bar{f}_l, \bar{f}_h, \bar{f}_v$. Next, we execute the following operations:

1. Let P' be an empty list that will hold the predicted *id*s. The *id*s in P' can repeat since two descriptors can be matched to the same *id*, as can be seen in the next step.

2. For each $D_{x,y,m} \in SD$, we execute the following:

(a) Find $QD_{qid} \in SQD_g$, where $g = (x, y, m, r)$, such that Euclidean distance between $D_{x,y,m}$ and QD_{qid} is smaller or equal to any other descriptor in SQD_g.

(b) Add all the ids of $MAP_g^{(qid)}$ to P'.

3. Let P be the set of unique values of P'.

4. For each $id \in P$ 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 Section 2.1.2.

(b) Let the SIFT confidence, p_s, be the number of occurrences of id in P' divided by $|P'|$.

(c) Let the confidence, c_{id}, of the classifier C in I being the symbol id, be $p_s \cdot p_m \cdot p_c \cdot p_o \cdot p_t \cdot p_l \cdot p_b \cdot p_r \cdot p_h \cdot p_v$/(base confidence of id).

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

2.4 Splitting Word into Pieces of Arabic Word

The classification process described in Section 2.6 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 doesn't 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

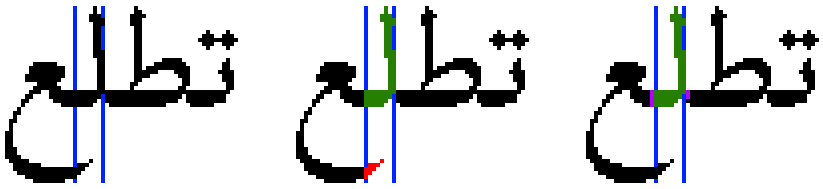
pixel, all *4-connectivity* CCs. At this point, each 4-connectivity CC has 2 labels, one 8-connectivity label and one 4-connectivity label. Next, all 4-connectivity CCs that are located on the *baseline* are marked as the anchors of each paw. For each anchor CC, we execute the following steps:

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 don't 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 arbitrary without being optimized for any of the datasets tested in Chapter 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%.

2.5 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. See Figure 2.7a which shows a paw with starting and ending window positions (inclusive) marked in blue. 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, as can be seen in Figure 2.7b, where the red color indicates the pixels of the next letter that are inside the window. 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, as can be seen in Figure 2.7c, where the black ink that we removed



(a) A window inside a paw marked by blue lines.

(b) A window inside a paw marked by blue lines. The green pixels are the pixels of the letter that we want to isolate. The red pixels are pixels of the next letter, but located inside the window.

(c) A window inside a paw marked by blue lines. The green pixels are the pixels of the letter that we isolated by cutting the baseline outside the window. The cuts that we made on the baseline are marked with purple.

Figure 2.7: A paw during the isolation process of a letter inside a window.

is marked in purple. 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 Section 2.4, 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.

2.6 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 don't 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 Section 2.4) 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 Section 2.6.2; step 5 is explained in detail in Section 2.6.3. The scanning procedure in steps 1, 2 and 5 is described in Section 2.6.1.

2.6.1 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 Section 2.5 and then classified using the classifier of form r as described in Section 2.3. Each window of size s is assigned with the result of the classifier on it, (id_s, c_s), and the following is executed:

1. Let s_b be the window size that has the highest classifier confidence c_{s_b}.
2. Let id_{s_b} be the id that was assigned by the classifier to the window of size s_b.
3. Return id_{s_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 id_{s_b} and c_s is the confidence of the classifier for the isolated letter inside the window to be id_{s_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.

2.6.2 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 id_{in} and $P_{in} = \{(p_1, c_1), \ldots, (p_n, c_n)\}$, where id_{in} is the id of the best initial letter starting at the beginning of the paw and P_{in} are the possible ending positions and confidences of the letter to end at those positions.

2. Scan for id_{fl_e} and $P_{fl_e} = \{(p_1, c_1), \ldots, (p_n, c_n)\}$, where id_{fl_e} is the id of the best final letter ending at the end of the paw and P_{fl_e} 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 id_{is_b} and $P_{is_b} = \{(p_1, c_1), \ldots, (p_n, c_n)\}$, where id_{is_b} is the id of the best isolated letter starting at the beginning of the paw and P_{is_b} are the possible ending positions and confidences of the letter to end at those positions.

2. Scan for id_{is_e} and $P_{is_e} = \{(p_1, c_1), \ldots, (p_n, c_n)\}$, where id_{is_e} is the id of the best isolated letter ending at the end of the paw and $P_{is_e b}$ 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, Ir_{is}, between the isolated letters, id_{is_b} and id_{is_e}, be twice the number of pixels shared by the windows with the highest confidence of id_{is_b} and id_{is_e} divided by the sum of the window sizes.

2. Let the intersection ratio, $Ir_{in fl}$, between the initial and final letters, be twice the number of pixels shared by the windows with the highest confidence of id_{in} and id_{fl_e} divided by the sum of the window sizes.

3. Let the "unclassified ratio", Ir_{isis} , be one minus the ratio of the number of pixels not covered by the windows with the highest confidence of id_{is_b} and id_{is_e} 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 $c_{t_1} = c_{is_b} \cdot c_{is_e} \cdot Ir_{is}$
- The confidence of a paw to be of type 2 is $c_{t_2} = c_{in} \cdot c_{fl_e} \cdot (1 - Ir_{infl})$
- The confidence of a paw to be of type 3 is $c_{t_3} = c_{is_b} \cdot c_{is_e} \cdot Ir_{isis} \cdot (1 - Ir_{is})$
- For c_{t_3} to be taken into account, it has to be significantly bigger than c_{t_2}, i.e. if $c_{t_2}/c_{t_3} > 0.9$, then $c_{t_3} = 0$. The value 0.9 was chosen arbitrary without being optimized for any of the datasets tested in Chapter 3.

Finally, based on the type confidences, we decide weather 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_{t_3} is the largest confidence, return id_{is_b}, id_{is_e}, the ids of the 2 isolated letters that were found.
2. If c_{t_1} is the largest confidence and $id_{is_b} = id_{is_e}$, return id_{is_b}.
3. If c_{t_1} is the largest confidence and $id_{is_b} \neq id_{is_e}$, 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) $c_{is_b} = c_{is_b}$·(size of the window that was classified to be id_{is_b} and confidence c_{is_b})
   (b) $c_{is_e} = c_{is_e}$·(size of the window that was classified to be id_{is_e} and confidence c_{is_e})

   If $c_{is_b} > c_{is_e}$ return id_{is_b}, otherwise return id_{is_e}.
4. If c_{t_2} is the largest confidence continue scanning for medial letters as described in Section 2.6.3.

2.6.3 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, id_{in}, and it possible ending positions and the confidences for the letter

to end at those positions, P_{in}. We also know the final letter that ends at the end of the paw, id_{fl_e}, and its possible starting positions and the confidences for it to start at those positions, P_{fl_e}. 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 id_{fl_e}.

2. Let p_l be the last possible starting position of id_{fl_e}.

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 id_{in}, while on the next iterations, this letter is the previous medial letter.

4. For windows ending before p_l, scan for id_m and $P_m = \{(p_1, c_1), \ldots, (p_n, c_n)\}$, where id_m is the id 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 id_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 id_{fl_b} and $P_{fl_b} = \{(p_1, c_1), \ldots, (p_n, c_n)\}$, where id_{fl_b} is the id of the best final letter starting at p_{pre} and P_{fl_b} is the possible ending positions and confidences of the letter to end at those positions. Let $(p_{fl_b}, c_{fl_b}) \in P_{fl_b}$ be the ending position and confidence for id_{fl_b} to end at this position, where c_{fl_b} is the highest confidence among all possible ending positions.

6. Let the intersection ratio, Ir_{mfl}, between the middle letter, id_m, and the final letter, id_{fl_e}, be twice the number of pixels shared by the windows with the highest confidence of id_m and id_{fl_e} divided by the sum of the window sizes. Update c_m to be $c_m \cdot (1 - Ir_{mfl})^{\frac{1}{2}}$. The exponent was arbitrary set to be 1/2 without being optimized for any of the datasets tested in Chapter 3.

7. Let the intersection ratio, Ir_{flfl}, between the final letter, id_{fl_b}, and the final letter, id_{fl_e}, be twice the number of pixels shared by the windows with the highest confidence of id_{fl_b} and id_{fl_e} divided by the sum of the window sizes. Update c_{fl_b} to be $c_{fl_b} \cdot Ir_{flfl}^{\frac{1}{2}}$. The exponent was arbitrary set to be 1/2 without being optimized for any of the datasets tested in Chapter 3.

8. If the SIFT confidence part, p_s, of the higher confidence between c_m and c_{fl_b} is less than 0.9 (the value 0.9 was chosen arbitrary without being optimized for any of the datasets tested in Chapter 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 id_m, c_m and id_{fl_b}, c_{fl_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_{fl_b} by the normalized confidence of the previous letter ending at this position.

   (d) Choose id_m, $P_m = \{(p_1, c_1), \ldots, (p_n, c_n)\}$ and id_{fl_b}, $P_{fl_b} = \{(p_1, c_1), \ldots, (p_n, c_n)\}$ 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 id_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 id_{in}, all medial letters and the final letter that has the higher confidence. If $c_{fl_b} > c_{fl_e}$ return id_{fl_b}; otherwise return id_{fl_e}.

2.7 Multi-font Classification

A quick observation shows that when an image that contains a text written using font A is classified using a classifier constructed for font B, the average confidence of the classified letters is significantly lower than the average confidence of letters returned by the classifier of font A on the same image.

To classify an image containing a text written using one font from a group of fonts $F = \{F_1, \ldots, F_n\}$, a classifier for each font in F is constructed. Next, the image is independently classified using each classifier as described in Section 2.6. The result of the multi-font classification process is the set of letters returned by the classifier that had the highest average of letter confidence.

Chapter 3

Experimental Results

In order to test the performance of SOCR, two different datasets were used. In Section 3.2 we describe how the PATS dataset was constructed [2] and compare the results of the algorithm suggested by the author of PATS to the results of SOCR. In Section 3.3 we describe how the APTI benchmark was constructed [11] and provide some initial tests results we did on parts of the dataset. To demonstrate the robustness of SOCR, in Section 3.4, we show the result of processing an image of scanned text using SOCR.

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:

- N_{Grid} - 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 Section 2.2.3 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* [6]

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 don't 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. Word© document files with the same text were created, each with one of the fonts. Each file was printed on paper sheets and then the sheets were scanned to images representing the printed pages. The images were split into images that each contain one line. Ground truth information is given as a Unicode text file. 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 Figure 3.1). Akhbar uses three more unique symbols while Thuluth, Naskh and Traditional Arabic use several more symbols that were not taken into consideration when creating a classifier (see Figure 3.2).

Since each image in the dataset contains a line of text, in order to measure WRR, each line has to be first split into words. In Arabic, this task creates a challenge, since there is a need to differentiate between white spaces and spaces inside a word (a space between two letters in the same word where one of the letter doesn't have medial form). To achieve separation, the number of pixels separating each two components is measured and a k-means clustering executed on the number of pixels with $k = 2$. The cluster with the larger centroid value contains the white spaces. This approach proved to work almost perfectly on Arial, Tahoma, Andalush, Akhbar and Simplified Arabic, while failing on Thuluth, Naskh and Traditional Arabic. For that reason, spaces were omitted from the ground truth when measuring CRR of Thuluth, Naskh and Traditional Arabic and WRR is not reported.

Using the tables shown in Figure 3.1 and 3.2, a classifier for each font, as described

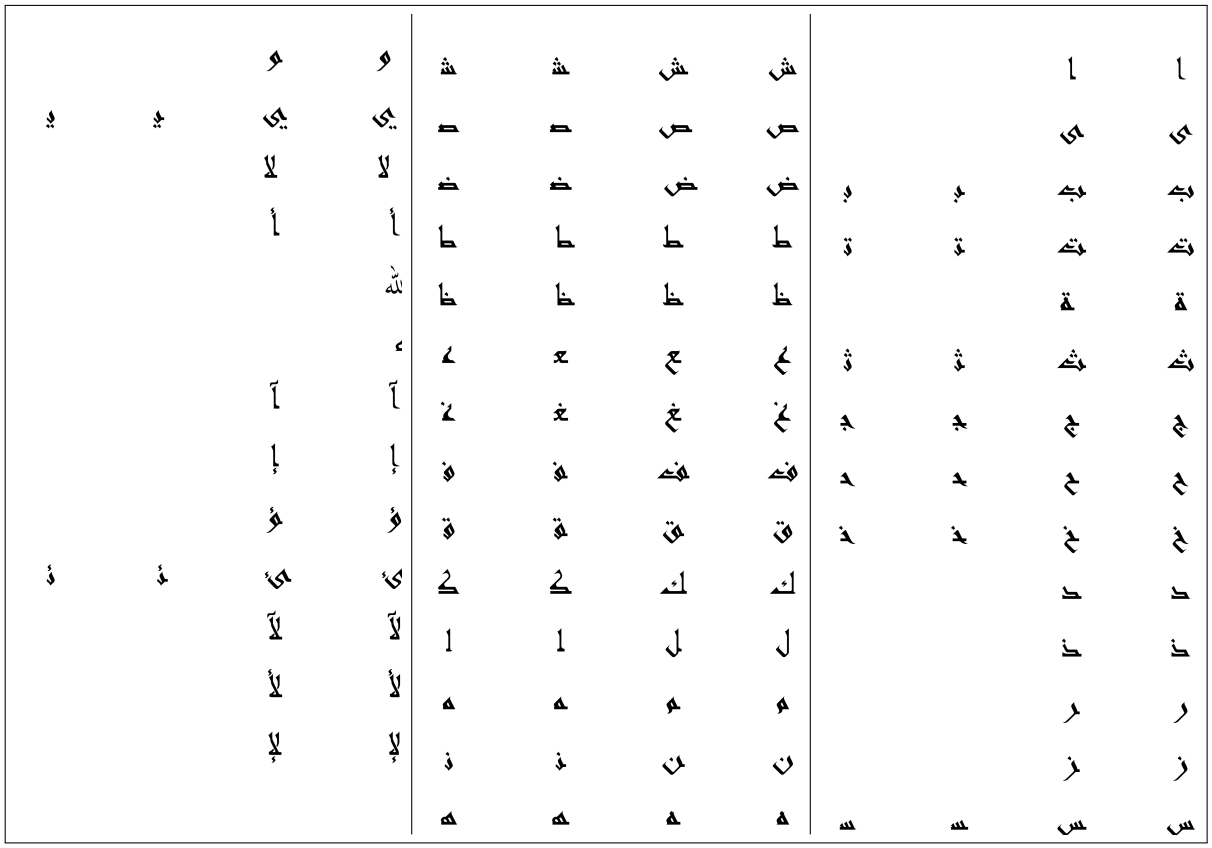
ا	ا			ش	ش	ش	شـ	و	و						
ى	ى			ص	ص	صـ	صـ	ي	ي	ـيـ	ـيـ				
ب	ب	بـ	بـ	ض	ض	ضـ	ضـ	لا	لا						
ت	ت	تـ	تـ	ط	ط	طـ	طـ	أ	أ						
ة	ة			ظ	ظ	ظـ	ظـ	لله							
ث	ث	ثـ	ثـ	ع	ع	عـ	عـ	ء							
ج	ج	جـ	جـ	غ	غ	غـ	غـ	آ	آ						
ح	ح	حـ	حـ	ف	ف	فـ	فـ	إ	إ						
خ	خ	خـ	خـ	ق	ق	قـ	قـ	ؤ	ؤ						
د	د			ك	ك	كـ	كـ	ئ	ئ	ـئـ	ـئـ				
ذ	ذ			ل	ل	لـ	لـ	لآ	لآ						
ر	ر			م	م	مـ	مـ	لأ	لأ						
ز	ز			ن	ن	نـ	نـ	لإ	لإ						
س	س	سـ	سـ	ه	ه	هـ	هـ								

(a) Arial

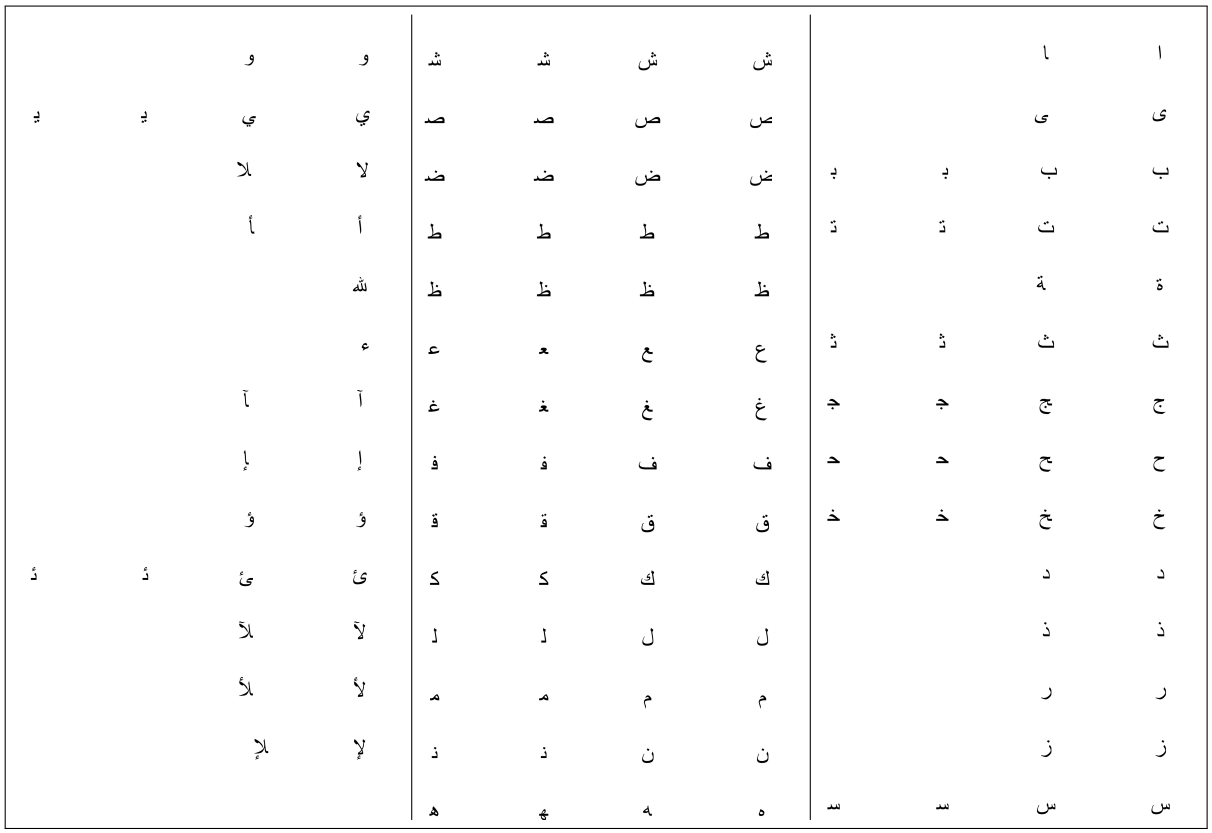
ا	ا			ش	ش	شـ	شـ	و	و						
ى	ى			ص	ص	صـ	صـ	ي	ي	ـيـ	ـيـ				
ب	ب	بـ	بـ	ض	ض	ضـ	ضـ	لا	لا						
ت	ت	تـ	تـ	ط	ط	طـ	طـ	أ	أ						
ة	ة			ظ	ظ	ظـ	ظـ	لله							
ث	ث	ثـ	ثـ	ع	ع	عـ	عـ	ء							
ج	ج	جـ	جـ	غ	غ	غـ	غـ	آ	آ						
ح	ح	حـ	حـ	ف	ف	فـ	فـ	إ	إ						
خ	خ	خـ	خـ	ق	ق	قـ	قـ	ؤ	ؤ						
د	د			ك	ك	كـ	كـ	ئ	ئ	ـئـ	ـئـ				
ذ	ذ			ل	ل	لـ	لـ	لآ	لآ						
ر	ر			م	م	مـ	مـ	لأ	لأ						
ز	ز			ن	ن	نـ	نـ	لإ	لإ						
س	س	سـ	سـ	ه	ه	هـ	هـ								

(b) Tahoma

Figure 3.1: A table of unique symbols used to construct the classifier of the first two fonts using 41 unique symbols.



(c) Andalus



(d) Simplified Arabic

Figure 3.1: A table of unique symbols used to construct the classifier of the last two fonts using 41 unique symbols.

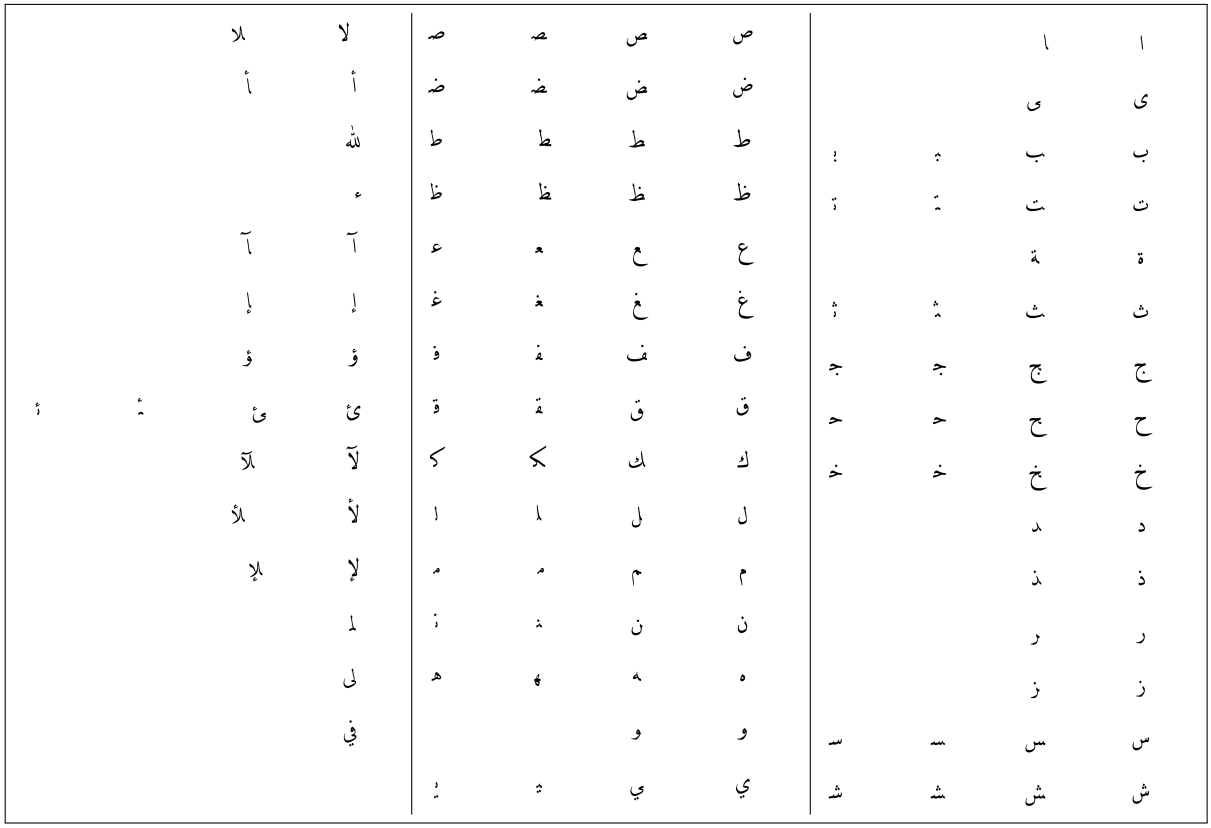
ا	ا			ص	ص	ص	لا	لا							
ى	ى			ض	ض	ض	أ	أ							
ب	ب	بـ	بـ	ط	ط	ط	لله								
ت	ت	تـ	تـ	ظ	ظ	ظ	ء								
ة	ة			ع	ع	ع	آ	آ							
ث	ث	ثـ	ثـ	غ	غ	غ	إ	إ							
ج	ج	جـ	جـ	ف	ف	ف	ؤ	ؤ							
ح	ح	حـ	حـ	ق	ق	ق	ئ	ئ	ئ	ئ	ئ				
خ	خ	خـ	خـ	ك	ك	ك	لآ	لآ							
د	د			ل	ل	ل	لأ	لأ							
ذ	ذ			م	م	م	لإ	لإ							
ر	ر			ن	ن	ن	لـ								
ز	ز			ه	ه	هـ	لى								
س	س	سـ	سـ	و	و		في								
ش	ش	شـ	شـ	ي	ي	يـ									

(a) Akhbar

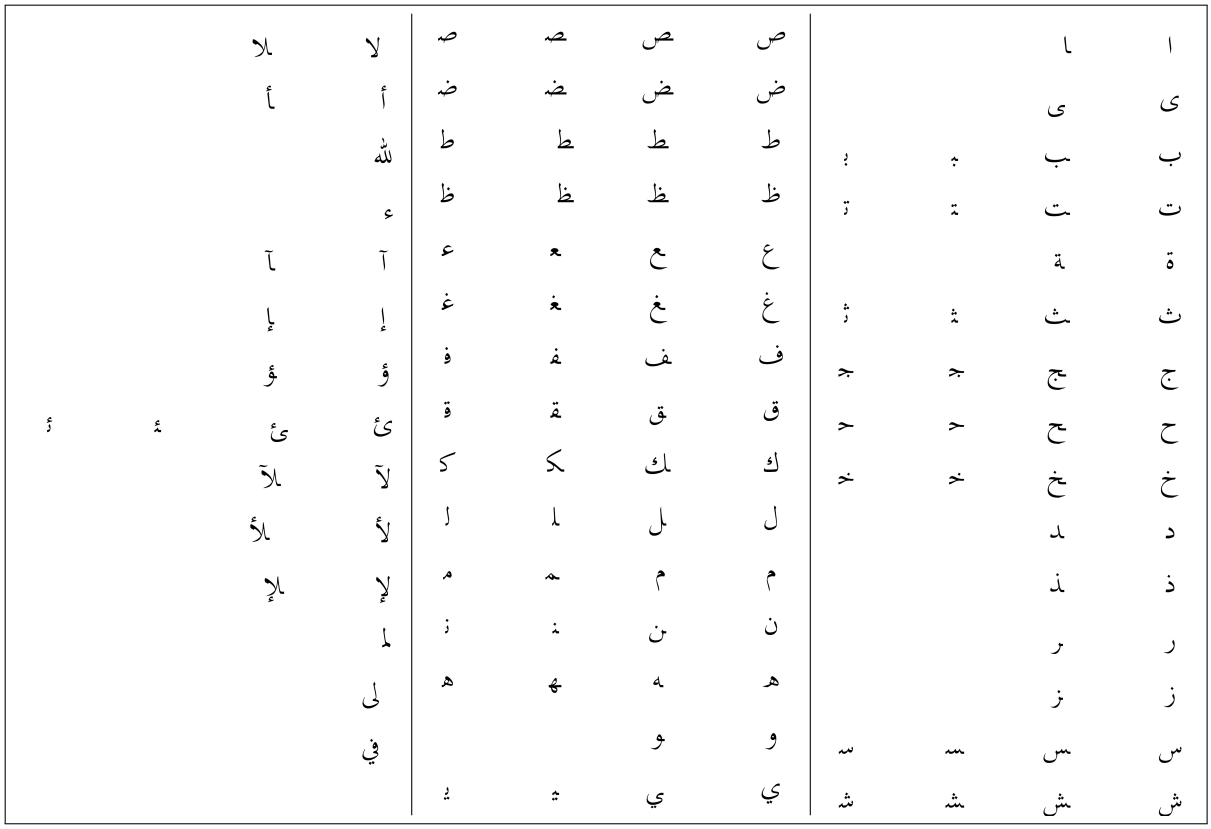
ا	ا			ص	ص	ص	لا	لا							
ى	ى			ض	ض	ض	أ	أ							
ب	ب	بـ	بـ	ط	ط	ط	لله								
ت	ت	تـ	تـ	ظ	ظ	ظ	ء								
ة	ة			ع	ع	ع	آ	آ							
ث	ث	ثـ	ثـ	غ	غ	غ	إ	إ							
ج	ج	جـ	جـ	ف	ف	ف	ؤ	ؤ							
ح	ح	حـ	حـ	ق	ق	ق	ئ	ئ	ئ	ئ	ئ				
خ	خ	خـ	خـ	ك	ك	ك	لآ	لآ							
د	د			ل	ل	ل	لأ	لأ							
ذ	ذ			م	م	م	لإ	لإ							
ر	ر			ن	ن	ن	لـ								
ز	ز			ه	ه	هـ	لى								
س	س	سـ	سـ	و	و		في								
ش	ش	شـ	شـ	ي	ي	يـ									

(b) Thuluth

Figure 3.2: A table of unique symbols used to construct the classifier of the first two fonts using 44 or more unique symbols.



(c) Naskh



(d) Traditional Arabic

Figure 3.2: A table of unique symbols used to construct the classifier of the last two fonts using 44 or more unique symbols.

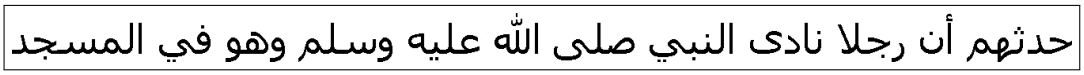
in Section 2.2, was constructed. Using the classifier, the SOCR algorithm was executed on all 2766 lines of each font. See Figure 3.3 for a sample line of each font. In Table 3.1 the performance, in terms of CRR and WRR, is reported for all 2766 lines and the last 266 lines, along side with the CRR of 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 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.

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

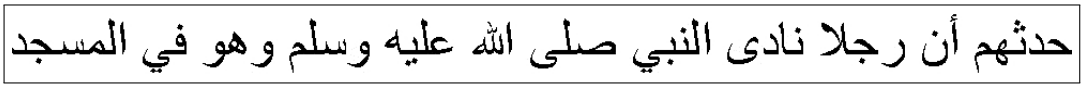
Table 3.1: Performance in terms of CRR and WRR of SOCR and the algorithm suggested by the author of PATS executed on PATS dataset.

3.3 APTI Dataset

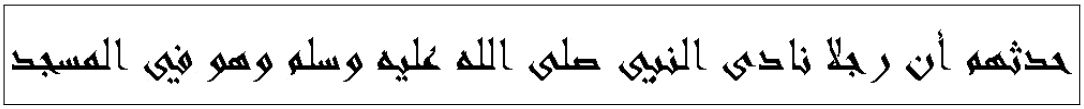
The Arabic printed text image database (APTI) [11], was created to address the challenges of optical character recognition of printed Arabic text of multiple fonts, multiple font sizes and multiple font styles. It consists of more than 100,000 different single words presented in 10 fonts (Andalus, Arabic Transparent, AdvertisingBold, Diwani Letter, DecoType Thuluth, Simplified Arabic, Tahoma, Traditional Aatbic, DecoType Naskh and M Unicode Sara), 10 font-sizes (6, 8, 10, 12, 14, 16, 18 and 24 points) and 4 font-styles (plain, italic, bold and combination of italic bold). APTI is designed for the evaluation of screen-based OCR systems. The images of the dataset are 72dpi resolution images of a single word, synthetically generated from a large corpus. The



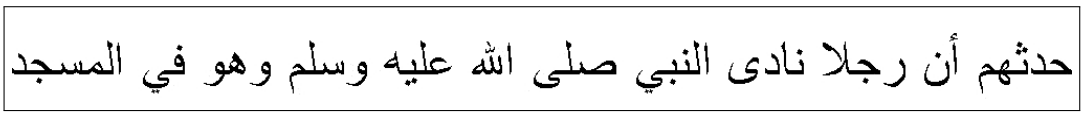
(a) Tahoma



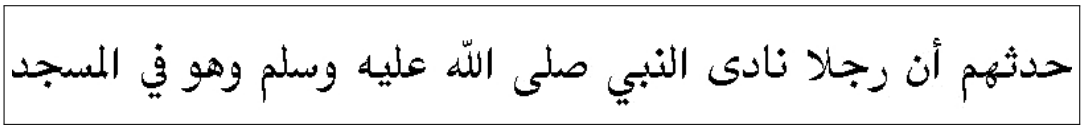
(b) Arial



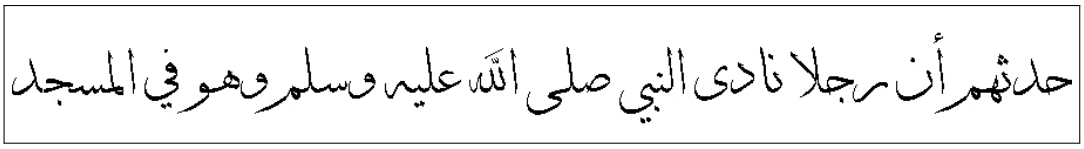
(c) Andalus



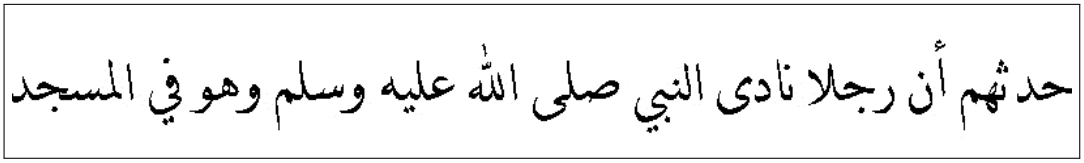
(d) Simplified Arabic



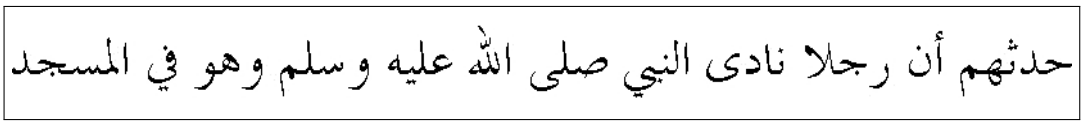
(e) Akhbar



(f) Thuluth



(g) Naskh



(h) Traditional Arabic

Figure 3.3: Line 88 of each font in the PATS dataset.

dataset is split into six sets, where the number appearances of each letter is evenly distributed between the sets. The authors suggested numerous OCR evaluation protocols and conducted a competition [12] to test the performance of state of the art OCR systems. 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. The ground truth is provided in XML files, and has a unique identifier for each combination of letter and letter form.

Table 3.2 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 Figure 3.4 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 [10] 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 [9]. 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. The SOCR classifier for it was constructed using the table shown in Figure 3.5.

From Table 3.2 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

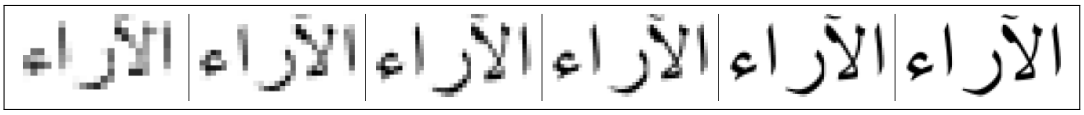


Figure 3.4: A word in Arabic Transparent and sizes 6, 8, 10, 12, 18 and 24 (from left to right).

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

Figure 3.5: A table of unique symbols used to construct the classifier for the Arabic Transparent font.

three systems used the first five sets to train their classifiers, while SOCR didn't 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.

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%

Table 3.2: Performance in terms of CRR and WRR of SOCR and the algorithm suggested by the author of PATS executed on PATS dataset.

3.4 Scanned Document Example

SOCR was also tested on an image of a scanned page written using the Scheherazade font. The classifier for the font was constructed using an alphabet containing 44 graphically-unique letters and combination of letters, similar to the way that the classifier for the font Akhbar was constructed. The input image can be seen in Figure 3.6 and the result of SOCR can be seen in Figure 3.7. Since SOCR is not aware of punctuation marks and other symbols that are not in the alphabet that was used to construct the classifier, it mistakenly classifies those symbols as letters. Those letters are marked in yellow, while the letters SOCR misclassified are underlined and marked in red. Since separating white spaces and spaces inside a paw was not a goal of this work, on the few occasions when the separation algorithm mentioned in Section 3.2 failed, the spaces were inserted or removed manually.

ولدت الرغبة فى أعقاب نظرة مفعمة بالإثارة ، والسفينة تشق طريقها ضد التيار الهادئ القوى فى أواخر فصل الفيضان . بدأت الرحلة من مدينتنا سايس ماضية جنوبا إلى بانو بوليس لزيارة أختى التى استقر بها الزواج هناك . وذات أصيل مررنا بمدينة غريبة . مدينة تطل من أركانها عظمة غابرة ، ويزحف الفناء بنهم على جنباتها وأشيائها . مترامية بين النيل غربا ومحراب الجبل شرقا ، متعرية الأشجار ، خالية الطرقات ، مغلقة الأبواب والنوافذ كالجفون المسدلة ، لا تنبض بها حياة ولا تند عنها حركة ، يجثم فوقها الصَمت وتخيم عليها الكآبة وتلوح فى قسماتها أمارات الموت . أجلت فيها البصر فانقبض صدرى ، وهرعت إلى أبى حيث يسترخى على أريكة فوق المنصة مجللا بشيخوخته وسألته :

ـــ ما شأن هذه المدينة يا أبى ؟

فأجاب دون تأثر :

ـــ مدينة المارق ، المدينة الكافرة الملعونة ، يا مرى مون ..

فرجع البصر إليها بانفعال مضاعف وذكريات منثالة ثم سألت :

ـــ ألا يوجد بها حى ؟

فأجاب أبى باقتضاب :

ـــ ما زالت المرأة المارقة تتنفس فى قصرها أو سجنها وهو الأصح ، كما يوجد بعض الحراس بلا ريب ..

فغمغمت متذكرا :

Figure 3.6: An input image of a page written in the Scheherazade font.

ولدت الرغبة فى أعقاب نظرة مفعمة بالإثارة خآ والسفينة تشق طريقها
ضد التيار الهادئغ القوى فى أواخر فصل الفيضان هه بدأت الرحلة من مدينتنا
سايس ماضية جنوبا إلى بانو بوليس لزيارة أختى التى استقر بها الزواج
هناك ه ئف ذات أأصيل مررنابمدينةغريبة مدينة تطل من أركانهاعظمة
غابرة هم ويزحف أألفناء بنهم على جنباتها وأشيائها ه مترامية بين النيل
غربا ومحرأأب الجبل شرقا إ متعرية أألأشجار هآ خالية أألطرقات ه مغلقة
الأبواب والنوافذ كالجفون المسدلة ه لاتنبض بها حياة ولاتند عنها
حركة خآ يجثم فوقها الطسث وتخيم عليها الكآبة وتلوح فى قسماتها
أمارأأت أألموت ه أجلث فيها أألبصر فانقبض صدرى م وهرعث حإلى أبى
حيث يسترخى على أريكة فوق المنصة مجللا بشيخوخته وسألته أ
حسس ماشأن هذه المدينة ياأبى إآ
فأجاب دون تأثر نا
حست مدينة المارق ة المدينة الكافرة الملعونة مم يامرى مون هههء
فرجع البصبب إليها بانفعال مضاعف وقؤكريات منثالة ثم سألث إ
حسب ألايوجد بها حى ما
فأجاب أبى باقتضاب إ
حسب ما زالث المرأة المارة تتنفس فى قصرها أو سجنها وهو الأصح م
كما يوجد بعض الحراس بلاريب هه
فغمغمث متذكرا ن

Figure 3.7: The result of SOCR on the image in Figure 3.6. The punctuation marks that are classified as letters are marked in yellow and the misclassified letters are underlined and marked in red.

Chapter 4

Conclusion

We have shown 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. As suggested by Prof. Lior Wolf, 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 the variety of fonts and sizes available in the APTI dataset and the performance of SOCR should be evaluated when executed on multiple fonts.

Bibliography

[1] Riaz Ahmad, Syed H. Amin, and Mohammad A.U. Khan. Scale and rotation invariant recognition of cursive Pashto script using SIFT features. *6th International Conference on Emerging Technologies*, pages 299–303, 2010. 1.1, 2.1.1.2

[2] Husni A. Al-Muhtaseb, Sabri A. Mahmoud, and Rami S. R. Qahwaji. Recognition of off-line printed Arabic text using Hidden Markov Models. *Signal Processing*, 88. 1.1, 3

[3] Markus Diem and Robert Sablatni. Recognition of degraded handwritten characters using local features. *International Conference on Document Analysis and Recognition*, pages 221–225, 2009. 1.1

[4] Hiromichi Fujisawa. Forty years of research in character and document recognition-an industrial perspective. *Pattern Recognition*, 41:2435–2446, 2008. 1

[5] Jia Ping Gui, Yi Zhou, Xin Da Lin, Kai Chen, and Hai Bing Guan. Research on Chinese character recognition using bag of words. *Applied Mechanics and Materials*, 20–23:395–400, 2010. 1.1

[6] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–710, 1966. 3.1

[7] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision*, volume 2 of *ICCV '99*, pages 1150–1, 1999. 1.1, 2.1.1

[8] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 42:91–110, 2004. 1.1, 2.1.1, 2.1.1.2

[9] Walid Magdy and Kareem Darwis. 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*, pages 408–411, 2006. 3.3

[10] Randall Munroe. Random number. *xkcd.com*, page 221. 3.3

[11] Fouad Slimane, Rolf Ingold, Slim Kanoun, Adel Alimi, and Jean Hennebert. A new Arabic printed text image database and evaluation protocols. In *International Conference on Document Analysis and Recognition*, pages 946–950, 2009. 1.1, 3, 3.3

[12] Fouad Slimane, Slim Kanoun, Haikal El Abed, Adel M. Alimi, Rolf Ingold, and Jean Hennebert. Arabic recognition competition: Multi-font multi-size digitally

represented text. In *Eleventh International Conference on Document Analysis and Recognition*, pages 1449–1453. IEEE, 2011. 1.1, 3.3

[13] Tal Steinherz, Ehud Rivlin, and Nathan Intrator. Offline cursive script word recognition - a survey. *International Journal on Document Analysis and Recognition*, 2:90–110, 1999. 1.1

[14] Tong Wu, Kaiyue Qi, Qi Zheng, Kai Chen, Jianbo Chen, and Haibing Guan. An improved descriptor for Chinese character recognition. *Third International Symposium on Intelligent Information Technology Application*, pages 400–403, 2009. 1.1

[15] Morteza Zahedi and Saeideh Eslami. Farsi/Arabic optical font recognition using SIFT features. *Procedia Computer Science*, 3:1055–1059, 2011. 1.1

[16] Majid Ziaratban and Karim Fae. A novel two-stage algorithm for baseline estimation and correction in Farsi and Arabic handwritten text line. *19th International Conference on Pattern Recognition*, 2008. 2