OCR for Arabic using SIFT Descriptors With Online Failure Prediction

Andrey Stolyarenko, Nachum Dershowitz
The Blavatnik School of Computer Science
Tel Aviv University
Tel Aviv, Israel
Email: stloyare@tau.ac.il, nachumd@tau.ac.il

Abstract—Character recognition for Arabic texts poses a twofold challenge, segmenting words into letters and identifying the individual letters. We propose a method that combines the two tasks, using a grid of SIFT descriptors as features for classification of letters. Each word is scanned with increasing window sizes; segmentation points are set where the classifier achieves maximal confidence. Using the fact that Arabic has four types of letters, isolated, initial, middle and final, we are also able to predict if a word is correctly segmented.

Performance of the algorithm applied to printed texts and computer fonts was evaluated on the PATS-A01 dataset. For fonts with non-overlapping letters, we achieve letter correctness of 87–96% and word correctness of 74–88%. For overlapping fonts, although the word correctness is low, only 14–23% are not predicted to be wrong.

We suggest several approaches for improved performance, with and without exploiting failure prediction.

Keywords-OCR, Optical, Character Recognition, Arabic, SIFT, Machine Learning, Predicting Correctness

I. Introduction

Printed Arabic texts present a difficult challenge for optical character recognition (OCR). The same Arabic 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). These multiple forms significantly increase the number of symbols a classifier needs to recognize to well over a hundred, besides ligatures and numerals. Even printed text is semi-cursive. The letters in a word are usually connected and cannot be easily separated, since finding the correct segmentation point is itself a challenge.

Recently, SIFT descriptors [1] have been suggested for use in OCR. They were used for Chinese character recognition in [3], [2], and were applied to degraded handwritten Latin manuscripts in [4]. The author of [5] uses hidden Markov models and a sliding window to segment and recognize Arabic scripts.

By using features extracted with a grid of SIFT descriptors and a sliding-window technique, we aim to jointly solve the segmentation and recognition problems for printed Arabic. We scan each connected component, and consider different segmentations of it into letters. For each possible segmentation, the classifier suggests a set of letters. The algorithm chooses those segmentation points for which the classifier

achieves its highest confidence in the recognized letters. The position of a letter within a word can be used to eliminate some misclassifications. In the experiments reported here, we did not make use of any language resources, such as a word list, which are normally used to improve performance.

In the next section, we explain how the classifier is constructed, and, in Section III, we show how it is combined with a sliding window to segment connected components into letters and recognize them. In Sections IV and V, we describe the benchmark we used and report on experimental performance results and their analysis. Finally, we conclude with suggestions of ways to improve our results.

II. Matching Based Classifier

Many Arabic letters are distinguished one from another only by diacritical dots or strokes, as in Figure 2. The main part of the letter is called the *ductus* (*rasm* in Arabic).

To classify a given letter, our algorithm, called *SOCR*, first aligns the image, then calculates descriptors and matches them against the table descriptors of a specific font. We assume that the Arabic text is an image with a white background and that baselines are horizontally aligned.

Font Table Descriptors: For each font of interest, a Word© document with all forms of all letters was exported as an image. Then a set of SIFT descriptors was computed for each form of each letter. For a set of 41 letters, we have 126 labeled forms and 1134 descriptors. See Figure 1 for the letter hā' in the Arial font. We call these *table descriptors*.

Alignment: The alignment process is very critical and directly affects the descriptors' values and, hence, recognition and segmentation quality. Each isolated letter is padded with white to the size of the smallest bounding *square*.

Scaling: To increase the variability of descriptors when classifying, we scale each aligned letter in the text under five magnifications, 1 through 5.

Calculating Descriptors: Descriptors are calculated for the aligned and scaled letters. The image of each letter is split into $N \times N$ identical squares covering the whole image. We calculate a SIFT descriptor in the middle of each square, as can be seen in Figure 2. In these experiments, $N = 3$.

Matching and Labeling: Given a set of n text descriptors of a letter, we match it against the descriptors table of a given font. We say that a text descriptor X *matches* a

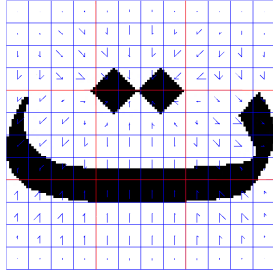Figure 1. Initial, medial, final and isolated forms of the letter hā'.

Figure 2. A grid of 3×3 SIFT descriptors for aligned letter tā' of isolated/final form.

table descriptor D if the Euclidean distance between the two is smaller than for any other table descriptor. This match is labeled with D's label. We assign to the letter a set of labels with confidence scores. The confidence in label L is m/n, where m is the number of matches that were assigned label L.

III. SOCR ALGORITHM

The SOCR algorithm receives as input a page in a form of an image that can contain one or more lines of printed text. We assume that the font used on the page belongs to a set of known fonts for which we have classifiers. Each line is split into connected components (of ink) that include ducti and diacritics. To make sure that letters do not lose their diacritics, connected components that vertically overlap are considered to be one component subword.

Figure 3. A word consisting of 3 subwords processed by SOCR (green lines are segmentation points; red line is the end of the sliding window.

Each such component subword is passed through a segmentation and classification process. We use a window of increasing size that starts at the beginning of a subword or at a previous segmentation point (See Figure 3). For each window size we execute the classifier. The classifier returns a list of predicted labels and their predicted confidence. Next, the labels are grouped by letters. Each letter can have up to four labels in its group, one for each form, isolated, initial, medial or final. The predicted confidence of each letter is the sum of confidences of its labels and confidence of each form is the confidence of the relevant label. At this point we have an array the elements of which represent the predicted letters and confidences for each window size. Theoretically,

at this point we can choose the segmentation point to be where we got the highest confidence for a specific letter. Practically we apply a series of steps that tend to improve the results significantly. The steps are as following:

A. Height to Width Ratio Based Rejection

For each window size we calculate the ratio R of height to width of the bounding box of the letter in the window. For each of the four letter forms, we intersect between the letters that receive a non-zero confidence in the window and the set of letters that have a ratio between $(1 - C)R$ and $(1 + C)R$. Small scale tests showed that setting $C = 0.15$ gives the best performance. This way, we get four arrays the elements of which represent the predicted letters and their confidences only for letters that match the ratio of the predicted letter. As a result of this process, letters that were predicted in the current window, but their ratio does not match, are rejected. We will see how this step affects recognition rates in Section V.

B. Additional Features

Various penalties can be applied to the confidences of each predicted letter by using additional features. We tested penalties based on two such features.

The first feature is the relative location (relative to the height and width of the window) of the center of mass of the window. The second feature is the relative locations of the vertical and horizontal slices with the largest proportion of black ink compared to white background. We call the latter, the *crosshairs*.

These features are extracted for each window size for all four possible letter forms and compared to the features of predicted letter of a relevant form. The comparison is done in terms of Euclidean distance. That distance d is transformed into a penalty $p = 1/(1 + \sqrt{d})$ and multiplied by the confidence to form a revised confidence score.

C. Form Confidence

For each of the four letter form vectors, we multiply the confidence of the letter by the confidence of the form. For some fonts, some of the letter forms are identical and there is a great chance that the confidence is spread among other forms of the classified letter. In that case, the multiplication of the letter confidence was done on the sum of confidences of forms that are identical.

D. Choosing Segmentation Point and Predicting Failure

At this point, we have four arrays with predicted letters for each form. For each form the best window size is chosen where a specific letter gets the highest confidence score. This way we obtain up to four letters, confidences and segmentation points as possible candidates. We choose the candidates with the highest letter confidence.

To predict a segmentation failure, we check if the chosen candidate letter has the form one would expect based on

its location in the component subword. If we are at the beginning of a component we expect to see an initial or an isolated form; if we are in the middle, we expect to see a medial form; while if we are at the end,we expect to see a final form. If the forms do not agree with the candidate assignment, we report a failure for the component. If the candidate having maximum confidence is different from the expected form, but the forms look identical, we do not report a failure.

After the best segmentation point is chosen, we repeat the process from the beginning. When we reach the end of a component, we decide whether to put a white space after it and move on to the next component.

E. White Space

A component subword can either be a whole word or a part of a word. To decide if to put a white space after it, we need to separate between white spaces and spaces inside a word (a space between two letters in the same word that one of them does not have a medial form). To achieve separation, we measure the number of pixels separating each two components and executed a k-means clustering on the number of pixels with $k = 2$. The cluster with the larger centroid value contains the white spaces.

F. Font Identification

Failure detection allows us to identify the font that is used in the input page. We repeat the recognition process using all possible font classifiers and then choose the results of the classifier that predicts the fewest word failures.

IV. Benchmark

To measure the performance of our SOCR algorithm, we used the Arial, Tahoma, Andalus an Akhbar fonts from the PATS-01 dataset [5]. The PATS-A01 dataset consists of 2751 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 into images representing the printed pages. The images were split into images that contain one line. Ground truth information is given as a Unicode text file.

Each line that was recognized by SOCR was compared to the ground-truth line. Both lines were split into words and each word was compared using Levenshtein edit distance [6]). If the distance was greater then zero, the word recognition fails and the number of failed letters is the edit distance. We also measured the number of words that failed but were correctly predicted to be wrong. The results on this benchmark are given in the next section.

V. Experimental Results

We separated the fonts into two groups, fonts whose letters do not overlap and fonts whose letters do. Arial and Tahoma belong to the first group and Andalus and Akhbar to the second. For each font, we tested different configurations: with and without form confidence, as described in Section III-C; with and without the penalties mentioned in Section III-B. Altogether, we tested six configurations:

NN: without form confidence and with no penalty;
NC: without form confidence and with centroid penalty;
NH: without form confidence and with crosshair penalty;
FN: with form confidence and with no penalty;
FC: with form confidence and with centroid penalty;
FH: with form confidence and with crosshair penalty.

For each font, the results are displayed below in a table and bar chart. The first column of the table is the configuration name as listed above, the second column is the percentage of correctly identified letters, the third is the percentage of correctly identified words and the last is the percentage of words that were correctly predicted to be misread. Green represents correctly identified words; yellow represents the words that were correctly predicted to be wrong; and red represents words that were incorrectly identified but not detected to be so. No words were ever falsely predicted to be wrong.

A. Non-Overlapping Fonts

Table I
ARIAL – 775 LINES, 11,706 WORDS, 46,320 LETTERS

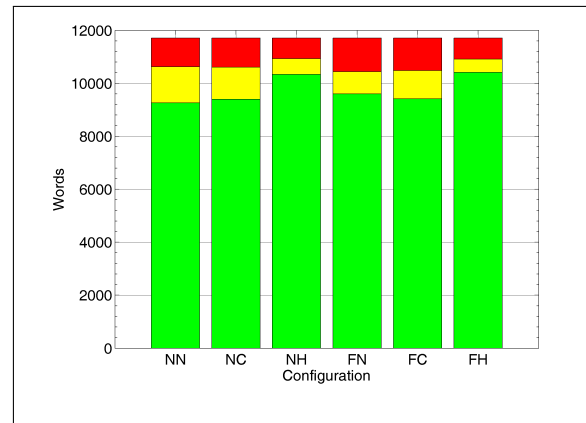| Config-uration	Correct Letters	Correct Words	Predicted Wrong
NN	92.0%	79.1%	11.6%
NC	91.7%	80.2%	10.4%
NH	95.9%	88.3%	5.1%
FN	92.4%	82.0%	7.1%
FC	91.9%	80.5%	9.1%
FH	**95.1%**	**88.9%**	**4.3%**

Figure 4. Arial font word performance (green denotes correct; yellow, detected mistakes; and red, undetected errors).

In Table I and Figure 4, we can see the benchmark result for Arial. We see that, using the crosshair penalty, results

are significantly improved. The best classifier is the one with form confidence and the crosshair penalty. It achieves 95% letter correctness and 89% word correctness.

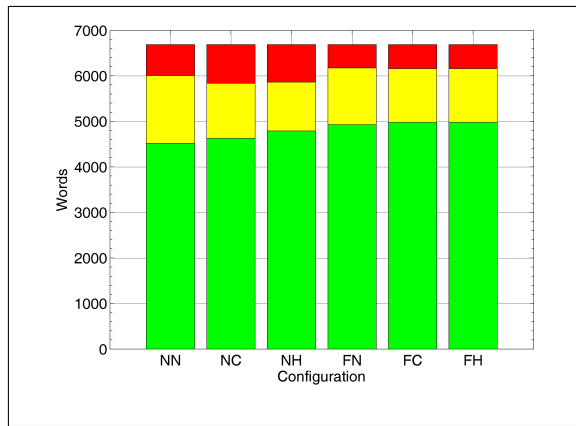| Config-uration	Correct Letters	Correct Words	Predicted Wrong
NN	83.4%	67.5%	22.3%
NC	85.7%	69.2%	18.1%
NH	85.9%	71.7%	16.0%
FN	84.9%	73.7%	18.7%
FC	86.8%	74.4%	17.7%
FH	**87.0%**	**74.4%**	**17.7%**

Figure 5.   Tahoma font word performance.

In Table II and Figure 5, we can see benchmark result for Tahoma. Using form confidence, results are significantly improved. The best classifier is the one that uses form confidence and the crosshair penalty. It achieves 87% letter correctness and 74% word correctness. Although the performance on Tahoma's dataset is worse then on Arial's, only 8% of the words are undetected errors.

B. Overlapping Fonts

Config-uration	Correct Letters	Correct Words	Predicted Wrong
NN	68.8%	42.8%	47.3%
NC	**71.0%**	**46.3%**	**39.0%**
NH	69.7%	41.8%	46.5%
FN	67.3%	41.6%	44.5%
FC	69.6%	43.6%	40.4%
FH	69.1%	41.1%	44.3%

Table III and Figure 6 give the benchmark results for Andalus. We see that when using the center-of-mass penalty results are improved. However, form confidence does not help for the overlapping font. All told, with this font, the
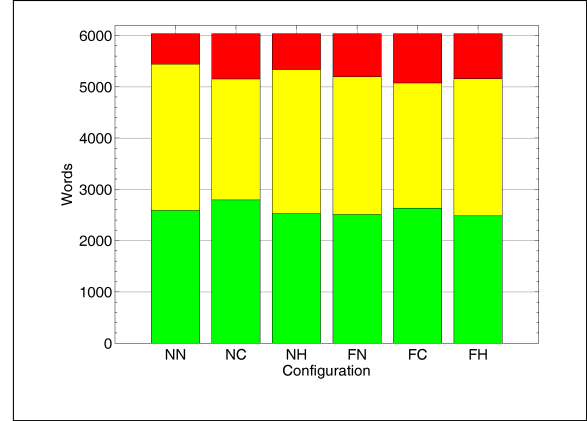

Figure 6.   Andalus font word performance.

algorithm achieves 71% letter correctness, but only 46% word correctness. Still, only 15% of the words are not predicted to be wrong by the algorithm.

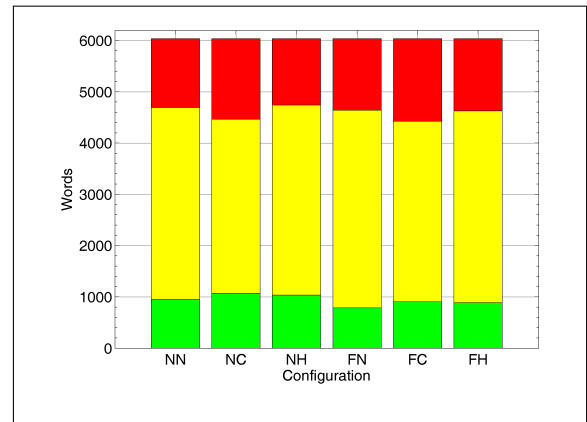| Config-uration	Correct Letters	Correct Words	Predicted Wrong
NN	55.8%	15.7%	62.0%
NC	**57.7%**	**17.7%**	**56.3%**
NH	58.6%	17.1%	61.4%
FN	51.1%	13.0%	63.8%
FC	53.6%	15.0%	58.3%
FH	53.8%	14.7%	61.9%

Figure 7.   Akhbar font word performance.

Lastly, in Table IV and Figure 7, we see the results for Akhbar. Again, when not using form confidence, the results are best. The best classifier does not use form confidence and does use the center-of-mass penalty. It achieves 58% letter correctness and only 18% word correctness. We also see that 26% of the words are not predicted to be wrong.

VI. CONCLUSIONS AND FUTURE WORK

We have seen how SIFT descriptors together with a sliding window can be used to successfully segment and identify Arabic printed words. We have also shown that by exploiting the fact that the same letter may take on different forms when located in different positions in the word, one can successfully predict the correctness of the segmentation quite frequently. Although the performance on fonts whose letters overlap is poor, the number of words that were not predicted to be wrong is reasonably small, so additional classification techniques can be applied on the words that are predicted to be mistakes.

The PATS-A01 data set is composed of reasonable quality scans. We plan to test the algorithm presented here also on a degraded images, where resolution is significantly smaller. To achieve good results in such cases and to improve results in general, we can suggest various improvements.

A. Penalties

The center-of-mass and crosshair penalties do in fact help improve recognition rates. The crosshair method always improves letter recognition; the center-of-mass method, almost always. We need to experiment with combinations of these penalties.

The letter-form heuristic needs to be modified to help word recognition with overlapping fonts.

B. Alignment

We have seen that with penalties based on additional features performance is improved. Those features should also be used to center the window in the alignment stage. Using an alignment based on these additional features should create a more robust segmentation that will be less dependent on "perfect" segmentation.

In these cases, modifying only the alignment process may not be enough. Letters that have the same ductus, but different diacritics, will be aligned to the same location in the window and will have similar set descriptors. In those cases, only few descriptors will be significantly different. To overcome this problem, quantization must be used. Increasing the grid size beyond 3 may also be needed when using different alignment.

C. Resegmentation on Failure Detection

If a failure is detected during the segmentation of a component subword, an attempt to choose a different segmentation point can be made. We can choose the starting point of the next window to overlap with the previous window. A different segmentation may be expected to give even better results when combined with a different alignment.

D. Separating Ductus from Diacritic

Separating the ductus from diacritics and classifying them separately can improve performance, since we expect to have greater variability between descriptors of different ducti and get stricter classifiers.

E. Dictionary Search

Suggested word readings can be tested against a dictionary, as is usually done in commercial systems. Words that do not appear in the dictionary can be replaced by the closest word (vis-à-vis edit distance) appearing in the dictionary. This is particularly important for words that are predicted to be erroneous.

REFERENCES

[1] D. Lowe, "Distinctive image features from scale-invariant keypoints", *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[2] J. Gui, Y. Zhou, X. Lin, K. Chen, and H. Guan, "Research on Chinese character recognition using bag of words", *Applied Mechanics and Materials*, vol. 20–23, pp. 395–400, 2010.

[3] T. Wu, K. Qi, Q. Zheng, K. Chen, J. Chen, and H. Guan, "An Improved descriptor for Chinese character recognition", *Third International Symposium on Intelligent Information Technology Application*, pp. 400–403, 2009.

[4] M. Diem and R. Sablatnig, "Recognition of degraded handwritten characters using local features", *International Conference on Document Analysis and Recognition*, pp. 221–225, 2009.

[5] H. Al-Muhtaseb, *Arabic text recognition of printed manuscripts*, PhD Thesis, University of Bradford, UK, 2010.

[6] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals", *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.