Arabic Font Recognition Based on Templates

Ibrahim Abuhaiba
Department of Electrical and Computer Engineering, Islamic University of Gaza, Palestine

Abstract: *We present an algorithm for a priori Arabic optical Font Recognition (AFR). First, words in the training set of documents for each font are segmented into symbols that are rescaled. Next, templates are constructed, where every new training symbol that is not similar to existing templates is a new template. Templates are sharable between fonts. To classify the font of a word, its symbols are matched to the templates and the fonts of the best matching templates are retained. The most frequent font is the word font.*

Keywords: *Optical character recognition, optical font recognition, vertical normalization, template matching.*

Received January 29, 2003; accepted May 4, 2003

1. Introduction

OCR systems of machine-printed documents can be divided into three groups: Mono-font, Multi-font, and Omni-font. Mono-font OCR systems deal with documents written with one specific font; their accuracy is very high but they need a specific module for each font. Omni-font OCR systems allow the recognition of characters of any font, and for this reason their accuracy is typically lower. Finally, Multi-font OCR systems handle a subset of the existing fonts. Their accuracy is related to the number and the similarity of the fonts under consideration. Character recognition accuracy can be improved using an Optical Font Recognizer (OFR) to detect the font type and subsequently convert the multi-font problem into mono-font character recognition problem.

Optical font recognition can be addressed through two complementary approaches: the a priori approach, in which characters of the analyzed text are not yet known, and the a posteriori approach, where the content of the given text is used to recognize the font. To our knowledge, there has been no study of the Arabic Font Recognition (AFR) problem. Available studies deal with Latin fonts, which have different characteristics than Arabic fonts. Therefore, in this paper, we present a novel solution to the a priori AFR problem.

Often, the font style is not the same for a whole document; it is a word feature, rather than a document feature, and its detection can be used to discriminate between different regions of the document, such as title, figure caption, or normal text. Hence, in our approach, we try to find the font per word. The detection of the font style of a word can also be used to improve character recognition: we know that Mono-font OCR systems achieve better results that Multi-font ones, so the recognition of document can be done using first an OFR, and then a Mono-font OCR.

In [4], font recognition is developed to enhance the recognition accuracy of a text recognition system. Font information is extracted from two sources: one is the global page properties and the other is the graph matching result of recognized short words such as a, it, and of.

In [3], a multi-font OCR system to be used for document processing is presented. The system performs, at the same time, both character recognition and font-style detection of the digits belonging to a subset of the existing fonts. The detection of the font-style of the document words can guide a rough automatic classification of documents, and can also be used to improve character recognition. The system uses the tangent distance as a classification function in a nearest neighbor approach. The nearest neighbor approach is always able to recognize the digit, but the performance in font detection is not optimal. To improve the performance, they used a discriminate model, the TD-Neuron that is used to discriminate between two similar classes.

In [5], a texture analysis based approach is used for font recognition. Existing methods are typically based on local features that often require connected components analysis. In this work, the document is taken as an image containing some special textures, and font recognition as texture identification. The method is content independent and involves no local feature analysis. The well-established 2-D Gabor filtering technique is applied to extract such features and a weighted Euclidean distance classifier is used in the recognition task. The reported average recognition accuracy of 24 fonts consisting of over 6,000 samples is 98.6%.

In [6], a study of image degradations effects on the performance of a font recognition system is presented. The evaluation that has been carried out shows that the system is robust against natural degradations such as those introduced by scanning and photocopying, but its

performance decreases with very degraded document images. In order to avoid this weakness, a degradation modeling strategy has been adapted, allowing an automatic adaptation of the system to these degradations. The adaptation is derived from statistical analysis of features behavior against degradations and is performed by specific transformations applied to the system knowledge base.

In [7], a statistical approach based on global typographical features is proposed for font recognition. It aims at the identification of the typeface, weight, slope, and size of the text from an image block without any knowledge of the content of that text. The recognition is based on a multivariate Bayesian classifier and operates on a given set of known fonts. The effectiveness of the adopted approach has been experimented on a set of 280 fonts. Font recognition accuracies of about 97% are reached on high-quality images. Rates higher that 99.9% were obtained for weight and slope detection. Experiments have also shown the system robustness document language and text content and its sensitivity to text length.

All previous OFR studies deal with Latin fonts and there has been no similar studies on Arabic fonts, which have different characteristics than Latin fonts. The most impeding characteristic of Arabic OCR systems is the cursive nature of Arabic script, which makes basic symbols not ready for direct OCR or OFR. Instead, there should be a segmentation stage to extract some kind of basic symbols. In this paper, we present a novel contribution to the a priori Arabic Font Recognition (AFR) approach where we try to find the font per word.

The basic idea in our method to recognize fonts is to segment words into symbols that act as representatives of these fonts. These symbols are not necessarily characters and are connected with short parts. The information about font characteristics that such parts bear is too low that they can be cleared out. Clearing these parts divides the word into segments or symbols that are usually smaller than characters. At this stage of font recognition we don't care to successfully segment the word into characters. Figure 1 shows one line of Arabic text followed by the same line segmented into symbols and written in three different fonts, from top to bottom: Simplified Arabic, Traditional Arabic, and Tahoma. Clearing some irrelevant connecting parts produces the symbols shown in the figure. The remaining parts are the required symbols to distinguish between the fonts of the three lines.

Our approach is summarized in the following steps. First, a training set of documents is assembled for each font. Second, symbols in the training set are found and rescaled. Third, templates are constructed. Every new training symbol that is not similar to existing templates is a new template. Templates are sharable between fonts. To classify the font of a word, its symbols are matched to the templates and the fonts of the best matching templates are retained. The most frequent font is the word font.

In our AFR system, a font is identified by four attributes: typeface (Simplified Arabic, Traditional Arabic, and Tahoma), size expressed in typographic points, slant (Roman, Italic), and weight (regular, bold).

The rest of the paper is organized as follows. Font learning and recognition stages are described in Sections 2 and 3, respectively. Experimental results are reported in Section 4. Finally, the paper is concluded in Section 5.

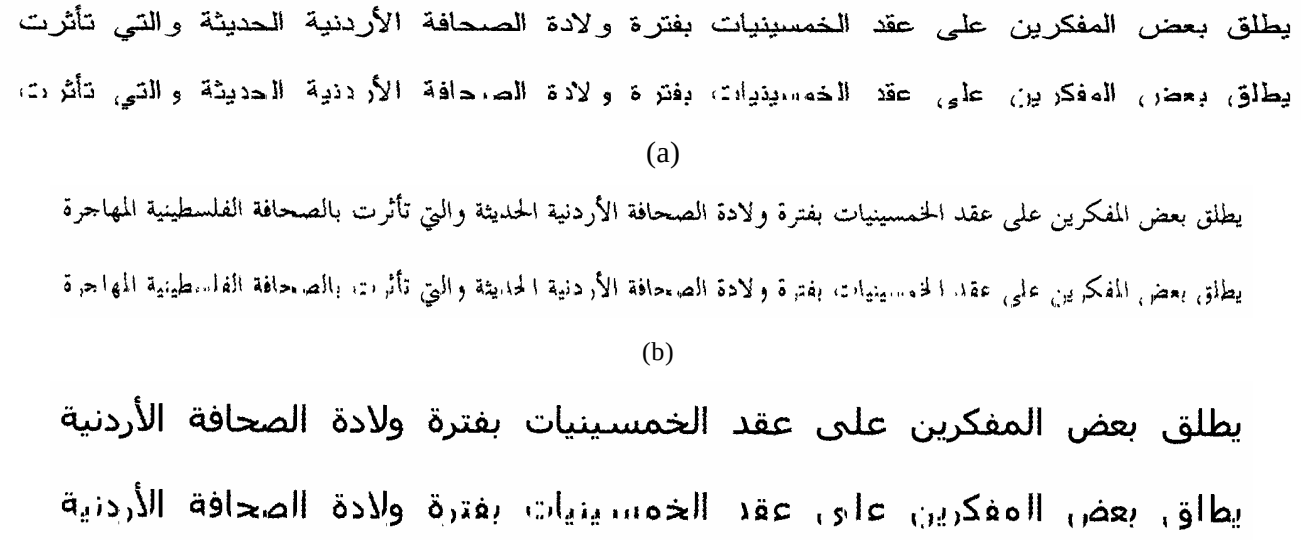

Figure 1. One line of Arabic text followed by the same line segmented into symbols and written in three different fonts, from top to bottom: (a) Simplified Arabic, (b) Traditional Arabic, and (c) Tahoma.

2. Learning

In our approach of font recognition, some image preprocessing is required in both the learning and testing stages. First, an image is skew-corrected using the algorithm of [1]. Next, horizontal and vertical solid lines are removed. Third, pepper noise is removed. The image is segmented into lines using horizontal white cuts, where each line consists of a sequence of words. Then, every line is segmented into words using vertical white cuts.

Every word is segmented into symbols as follows. We find all horizontal and vertical runs of 1's, assuming that pixels belonging to a word are assigned the value 1. The average, runave of these runs is calculated. Then, the word is scanned from right to left, where we sum the pixels along every column. If this sum is less than runave, then that column is cleared, i.e. all its pixels are changed to zero. The idea behind this step is that columns having a number of white pixels less than runave most probably belong to parts that connect adjacent characters in a word. We consider that the information about font characteristics that such parts bear is too low that they can be cleared out. Clearing these columns divides the word into segments that are not necessarily characters. At this stage of font recognition we don't care to successfully segment the word into characters.

All symbols in the training set are extracted. Symbols that don't satisfy certain size constraints are filtered out. For example, if the height or width of the symbol is less than some specified thresholds or is greater than some other thresholds, then the symbol is discarded, see Section 4 for values of these thresholds.

After all symbols in the training images are extracted, they are vertically normalized. In the context of document understanding, the normalization operation almost means to normalize in two directions: x and y. Actually, this can be problematic since information is lost due to this kind of two-dimensional normalization. However, performing a one-dimensional normalization preserves the height/width ratio. In Arabic, a word consists of characters some of which can be connected. The direction of writing follows horizontally from right to left. There is no limit to the number of characters that can be connected. Thus, for a specific font the word height has a limited variability while the word width is so variable that it is more informative to normalize in the vertical direction such that the height/width ratio is preserved, which results in normalized words that retain the relative geometrical attributes of the original un-normalized words. For more details on vertical normalization, see [2].

Let S represent the set of vertically normalized symbols. Every symbol $s \in S$ consists of the 2-tuple (Is, fs) where Is is the normalized symbol image, and fs is the symbol font. In our system, a font , f, is characterized by the 6- tuple (h, a, p, z, s, w), where h is the symbol height before normalization, a is the symbol area before normalization, p, z, s, w are the typeface, size, slant, and weight, respectively. Then, the set of global templates, T, is constructed as follows. Initially, T is empty. Every template $t \in T$ is a 2-tuple (It, F), where It is the vertically normalized template binary image, and F is the set of fonts that this template represents. Given a new training symbol s = (Is, fs) = (Is, (hs, as, ps, zs, ss, ws)) $\in$ S, the similarity, s(s, t) between this symbol and every template $t \in T$ is calculated, where T is the current set of templates. If the most similar template, t* = (It*, F*), yielded a similarity not less than a certain threshold, SMIN, then the symbol font fs is added to the list F* of t* such that there is no font f = (ht, at, pt, zt, st, wt) $\in$ F* that has |ht – hs| = HeightTolerance, |at – as| = AreaTolerance, pt = ps, zt = zs, st = ss, and wt = ws. Otherwise, the new symbol, s = (Is, fs), is added as a new template, t = (It = Is, F = { fs}), to the set of templates, T.

The symbols that are used early in the training phase don't see the templates constructed from later symbols, i.e., they aren't matched against each other. Thus, there can be a possibility that a symbol is matched to some template with certain similarity; then, later, a new template is generated which if matched against that symbol yields better similarity. Thus, after finding the templates, the sets of fonts, F's that templates represent are emptied and another scan is performed over the training set to match every symbol against the best template. The font of the symbol is added to the set of fonts that the best template represents. In this last phase, no new templates are generated.

The way we calculate the similarity between symbols follows. Let s = (Is, f) be a vertically normalized symbol and t = (It, F) be a template. The template image, It, is already normalized since this is a task of the learning algorithm. Thus, both the symbol and the template have the same height. Let wmin and wmax be equal to the minimum and maximum of the widths of the symbol image, Is, and the template image, It, respectively. Let area be equal to wmax î normalized height. Initially, the similarity, s(s, t) = 1.0 – (wmax – wmin) normalized height / area. If this similarity is less than a specified threshold, SMIN, then there is no match between the symbol and the template. If the width of the symbol is less than or equal to that of the template then the Hamming distance, dH, between the symbol image, Is, and every consecutive wmin columns of the template image, It, is calculated. Or, if the width of template image is less than that of the symbol then the Hamming distance, dH, between the template image, It, and every consecutive wmin columns of the symbol image, Is, is calculated. The minimum distance, dHmin, is retained. A value equal to dHmin / area is subtracted from the remaining similarity to obtain the final similarity. If this similarity is not less than SMIN then the symbol is accepted, otherwise it is considered unmatched.

In our approach, it is worth mentioning that when a new template is created its image is set to that of a single

symbol. This means that extra symbols matched to the template are not used to modify the template's image. This is in contrast with some clustering algorithms that create clusters of symbols and calculates the cluster's centroid to form templates. We found out that our algorithm works very well without that averaging step. Algorithm 1, in the following text, is a formal description of the font learning algorithm.

Algorithm 1

Step 1: Preprocessing

a. *Correct the skew of the input image.*
b. *Remove horizontal and vertical solid lines.*
c. *Remove pepper noise.*
d. *Segment the image into lines using horizontal white cuts, where each line consists of a sequence of words.*
e. *Every line is segmented into words using vertical white cuts. At the end of Step 1 a set of words, W, is obtained.*

Step 2: Segmenting words into symbols

Assuming that pixels belonging to a word are assigned the value 1, for every word w ∈ W do

a. *Find all horizontal and vertical runs of 1's. Calculate the average, runave of these runs.*
b. *Scan the word from right to left to sum the pixels along every column. For any column, if this sum is less than runave, then that column is cleared.*
c. *Segment the word into symbols using vertical white cuts. Each symbol is defined by its minimum bounding rectangle.*
d. *Vertically, normalize the symbols that pass some tests. Here, only the image, Is, of every symbol is normalized, i.e., other values don't change. At the end of Step 2, a set of symbols, S, is obtained, where each symbol s ∈ S is a 2 tuple (Is, fs) and fs = (hs, as, ps, zs, ss, ws), where hs, as, ps, zs, ss, and ws are as defined before.*

Step 3: Global template construction

a. *Let the set of templates be T = f, the empty set.*
b. *For every training symbol s = (Is, fs = (hs, as, ps, zs, ss, ws)) ∈ S do {*

For every template t = (It, F) ∈ T do{

Find the similarity, s(s, t)

}

From the earlier computed similarities, let the most similar template be tbest = (It, F*) with similarity sbest;*

If sbest ≥ SMIN then{

If there is no font f = (ht, at, pt, zt, st, wt) ∈ F such that |ht – hs| = HeightTolerance, |at – as| = AreaTolerance, pt = ps, zt = zs, st = ss, and wt = ws, then update tbest by letting F* = F* ∪ fs*

}

else {

Create a new template t = (It = Is, F = { fs}),;

Let T = T ∪ t

}

}

Step 4: Template tuning

a. *For every template t = (It, F) ∈ T let F = f, the empty set;*
b. *For every training symbol s = (Is, fs = (hs, as, ps, zs, ss, ws)) ∈ S do {*

For every template t = (It, F) ∈ T do {

Find the similarity, s(s, t)

}

Let the template tbest = (It, F*) be the one that has the best similarity sbest among all similarities computed in the previous loop;*

If sbest ≥ SMIN then{

If there is no font f = (ht, at, pt, zt, st, wt) ∈ F such that |ht – hs| = HeightTolerance, |at – as| = AreaTolerance, pt = ps, zt = zs, st = ss, and wt = ws, then update tbest by letting F* = F* ∪ fs*

}

}

3. Recognition

The same preprocessing operations used in the learning stage are also used in the recognition stage. To identify the font of a new word, it is segmented into symbols. Symbols passing some tests are matched against templates. The template that yields the best similarity is recorded. If the final similarity is not less than a certain threshold, SMIN, then the symbol is considered accepted. For every accepted symbol, s = (Is, fs), where fs is unknown, the set of fonts associated with the template that best matched the symbol, such that the absolute differences between the heights of the symbol and the font and areas of the symbol and the font are not greater than certain thresholds, is retained. Thus, for every symbol, there will be a set of candidate fonts each represented by the 4 tuple (p, z, s, w), i.e., h and a are dropped. Now, for the set of accepted symbols in the word, we count how many times each font appears. The font that achieves the maximum count is the output font for the whole word. If no symbols of the word are accepted, then the word font is unknown. Algorithm 2, in the following text, is a formal description of the font recognition algorithm.

Algorithm 2

Steps 1 & 2: Preprocessing and segmenting a word into symbols
These are the same as Steps 1 & 2 of Algorithm 2, where, at the end, a set of vertically normalized symbols, S, is obtained.

Step 3: Symbol acceptance

a. *For every symbol s = (Is, fs = (hs, as, ps, zs, ss, ws)) ∈ S, define Qs to be its set of candidate fonts. Initially, Qs is empty.*
b. *Find the best matching template, t* = (I*, F*) with similarity s*. If s* < SMIN then the symbol font is unknown, otherwise, add every font f = (ht, at, pt, zt, st, wt) ∈ F* such that |ht – hs| = HeightTolerance, |at – as| = AreaTolerance to the list of fonts, Qs. If no such font is found, then the symbol font is unknown.*

Step 4: Word font selection

a. *For every word, w, consisting of a sequence of symbols s1, s2, …, sm, with corresponding sequence of sets of fonts Q1, Q2, …, Qm, concatenate these sets to form a list of fonts L.*
b. *The most frequent font appearing in L is selected as the recognized font of the word. If L is empty then the word font is unknown.*

4. Results

In the fonts used in the experimentation, we have three typefaces: Simplified Arabic, Traditional Arabic, and Tahoma. The slant is either Roman or italic. The weight is either regular or bold. The size is 12, 13, or 14 points. Thus, a total of 36 fonts were investigated. Table 1 summarizes the fonts used in our AFR system. Two files with different content were compiled to represent the learning and testing data sets. These files were printed using a laser jet printer once for every font in Table 1. The total number of printed pages is 380 and 390 A4 pages for the learning and testing data sets, respectively.

Table1. Arabic fonts used in our AFR system.

Typeface	Size	Slant	Weight
Simplified Arabic	12, 13, 14	Roman, Italic	Regular, Bold
Traditional Arabic	12, 13, 14	Roman, Italic	Regular, Bold
Tahoma	12, 13, 14	Roman, Italic	Regular, Bold

The minimum similarity, SMIN, used in the learning and testing phases was set to 0.90. In Steps 1 and 2 of Algorithms 1 and 2, a symbol that does not satisfy any of the following constraints is filtered out: the symbol width and height are at least 3 pixels each, the maximum width and maximum height are 600 and 200 pixels, respectively. These values were empirically determined and proved adequate to eliminate flecks and some non-textual content.

The learning phase produced 41,662 global templates. Some of these templates are shown in Figure 2. Notice that the template height is constant; i.e., only the template width varies, which is due to vertical normalization.

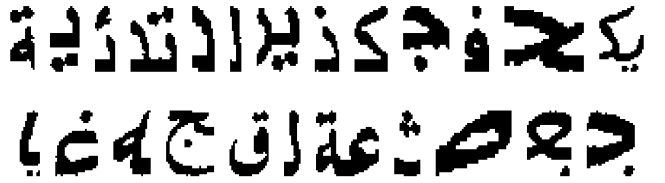

Figure 2. 24 sample template images extracted from the three Arabic fonts under study.

Table 2 shows font recognition results. The overall error, rejection, and success rates are 15.0%, 7.6%, and 77.4%, respectively. The high error rate is mainly due to errors in size recognition. Also, a high typeface error rate is noticed in some fonts. The rejection rate is high in some fonts, which can be reduced by learning more sample pages. The success rate can be increased by doing more learning and incorporating more discriminative features other than templates.

The algorithm was implemented and run on a Pentium III 866MHz PC with 128 MB RAM. The average time required to recognize the word font is approximately one second. The time can be reduced by using some programming optimization techniques and more powerful computers.

5. Conclusion

To our knowledge, there has been no study of the Arabic Font Recognition (AFR) problem. Available studies deal with Latin fonts, which have different characteristics than Arabic fonts. Therefore, in this paper, we presented a novel solution to the a priori AFR problem.

Often, the font style is not the same for a whole document; it is a word feature, rather than a document feature, and its detection can be used to discriminate between different regions of the document, such as title, figure caption, or normal text. Hence, in our approach, we find the font per word. The detection of the font style of a word can also be used to improve character recognition.

The overall font recognition rate was low, which can be increased by doing more learning and incorporating more discriminative features other than templates. The recognition time was high, however, it can be reduced by using some programming optimization techniques and more powerful computers.

Acknowledgment

The author would like to thank the referees for their valuable comments.

Table 2. Results of Arabic Font Recognition. In Slant column: R = Roman, I = Italic. In Weight column: R = Regular, B = Bold. Et = typeface error, Ez = size error, Es = slant error, Ew = weight error, ET = total error, Rej = rejection rate, Succ = success rate.

Font				Words	Et	Ez	Es	Ew	ET	Rej	Succ
Typeface	**Size**	**Slant**	**Weight**		**%**	**%**	**%**	**%**	**%**	**%**	**%**
Simplified Arabic	12	R	R	6242	0.7	0.6	0.3	0.4	1.3	4.7	94.0
Simplified Arabic	12	R	B	5249	0.3	0.3	0.3	15.4	15.8	5.8	78.4
Simplified Arabic	12	I	R	3756	0.7	0.6	1.4	0.6	2.4	4.2	93.3
Simplified Arabic	12	I	B	3410	0.7	0.8	1.4	4.6	5.8	4.5	89.7
Simplified Arabic	13	R	R	6119	0.4	3.4	0.6	2.0	3.6	5.4	91.0
Simplified Arabic	13	R	B	5459	0.3	2.3	0.6	10.2	12.1	4.7	83.1
Simplified Arabic	13	I	R	3874	0.4	4.2	1.1	1.7	4.7	3.8	91.5
Simplified Arabic	13	I	B	3600	0.6	3.4	1.4	3.4	6.4	5.4	88.2
Simplified Arabic	14	R	R	6542	0.1	21.1	0.4	8.1	21.2	3.3	75.5
Simplified Arabic	14	R	B	5153	0.3	7.3	0.6	6.4	13.4	4.2	82.4
Simplified Arabic	14	I	R	3948	0.2	11.3	1.6	2.8	11.7	3.4	84.9
Simplified Arabic	14	I	B	3477	0.8	11.2	1.2	2.4	12.6	4.1	83.3
Traditional Arabic	12	R	R	7056	2.5	3.4	1.0	1.3	6.4	8.3	85.4
Traditional Arabic	12	R	B	6792	4.8	2.5	0.9	12.4	14.3	6.8	78.8
Traditional Arabic	12	I	R	4523	6.9	7.7	2.7	1.9	13.1	5.5	81.4
Traditional Arabic	12	I	B	4405	5.5	4.4	1.8	6.9	11.1	5.1	83.8
Traditional Arabic	13	R	R	7043	4.6	22.6	0.9	2.3	23.2	5.3	71.5
Traditional Arabic	13	R	B	6845	6.0	25.7	0.9	10.2	31.1	6.5	62.3
Traditional Arabic	13	I	R	4498	6.8	22.7	2.6	5.0	23.8	4.5	71.7
Traditional Arabic	13	I	B	4378	12.1	23.3	1.2	7.1	28.0	3.9	68.1
Traditional Arabic	14	R	R	7065	11.1	31.9	1.2	2.6	32.3	6.5	61.2
Traditional Arabic	14	R	B	6889	14.6	24.6	0.8	14.8	29.2	7.9	63.0
Traditional Arabic	14	I	R	4516	10.0	29.1	2.4	4.9	29.9	5.5	64.6
Traditional Arabic	14	I	B	4415	14.7	23.1	1.6	10.6	26.9	5.1	68.0
Tahoma	12	R	R	5790	7.5	29.2	3.2	5.3	32.4	22.8	44.9
Tahoma	12	R	B	6660	1.0	21.5	2.4	0.6	22.8	39.2	38.0
Tahoma	12	I	R	3955	7.7	19.4	6.9	5.2	26.2	22.9	50.8
Tahoma	12	I	B	3957	2.4	19.9	8.4	1.2	27.3	31.4	41.2
Tahoma	13	R	R	5651	1.2	4.1	0.6	0.8	4.5	3.2	92.4
Tahoma	13	R	B	6469	0.1	0.6	0.2	0.0	0.7	2.6	96.7
Tahoma	13	I	R	3957	2.6	3.9	1.9	1.7	4.7	3.3	92.0
Tahoma	13	I	B	3884	0.9	2.3	1.3	0.7	2.7	2.4	94.9
Tahoma	14	R	R	5815	1.2	4.7	0.6	1.1	5.1	4.2	90.8
Tahoma	14	R	B	6600	0.0	5.1	0.0	0.1	5.1	2.5	92.4
Tahoma	14	I	R	3962	2.7	5.8	2.4	2.4	6.2	3.3	90.4
Tahoma	14	I	B	3885	0.2	3.4	1.9	0.2	4.0	4.1	91.9
All	All	All	All	185,839	3.8	11.9	1.5	4.6	15.0	7.6	77.4

References

[1] Abuhaiba I., "Skew Correction of Textual Documents," *accepted in King Saud University Journal, Computer and Information Sciences Division*.

[2] Abuhaiba I., "Discrete Script or Cursive Language Identification from Document Images," *accepted in Journal of King Saud University, Engineering Sciences Division*.

[3] Manna S. L., Colla A. M., and Sperduti A., "Optical Font Recognition for Multi-Font OCR and Document Processing," *10th International Workshop on Database & Expert Systems Applications*, Florence, Italy, pp. 549-553, 1999.

[4] Shi H. and Pavlidis T., "Font Recognition and Contextual Processing for More Accurate Text Recognition," *4th International Conference on Document Analysis and Recognition: (ICDAR'97*), Germany, pp. 39-44, 1997.

[5] Zhu Y., Tan T., and Wang Y., "Font Recognition Based on Global Texture Analysis," *Fifth International Conference on Document Analysis and Recognition: (ICDAR'99)*, Bangalore, India, pp. 349-352, 1999.

[6] Zramdini A. and Ingold R., "A Study of Document Image Degradation Effects on Font Recognition," *(ICDAR'95)*, Montreal, Canada, pp. 740-743, 1995.

[7] Zramdini A. and Ingold R., "Optical Font Recognition Using Typographical Features," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 877-882, 1998.

Ibrahim Abuhaiba is an assistant professor at the Department of Electrical and Computer Engineering, Islamic University of Gaza, Palestine. He obtained his Master of Philosophy and Doctorate of Philosophy from Britain in the field of document understanding and pattern recognition. His research interests include computer vision, image processing, document analysis and understanding, pattern recognition, artificial intelligence, and many other fields. Dr. Abuhaiba presented important theorems and more than twenty-five algorithms in text recognition. He published many original contributions in the field of document understanding in well-reputed international journals and conferences.