Font clustering and cluster identification in document images

Serdar Öztürk
Bülent Sankur
Boğaziçi University
Department of Electrical-Electronic Engineering
TR-80815 Bebek
Istanbul, Turkey
E-mail: sankur@boun.edu.tr

A. Toygar Abak
Marmara Research Center
Information Technologies Research Institute
Gebze
Kocaeli, Turkey

Abstract. *In this work clustering and recognition problem of fonts in document images is addressed. Various font features and their clustering behavior are investigated. Font clustering is implemented both from shape similarity or from OCR performance points of view. A font recognition algorithm is developed that can identify the font group or the individual font from which a text was created. © 2001 SPIE and IS&T.* [DOI: 10.1117/1.1351820]

1 Introduction

Today, optical character recognition (OCR) machines can recognize most nonstylized fonts without having to maintain huge databases of specific font information. This is provided by omnifont technology. The database of an omnifont system will contain a description of each symbol class instead of the symbols themselves. This gives flexibility in automatic recognition of a variety of fonts. Although omnifont is the common term for these OCR systems, this should not be understood literally as the system being able to recognize characters well under all existing fonts. No OCR machine performs equally well on all the fonts used by modern typesetters.

Furthermore, new computer-generated fonts are being added to the existing fonts in the printing industry. Such increasing diversity of font styles is a factor limiting the performance of general-purpose OCR systems. The detection of font style of a text is an obvious way to improve the performance of optical recognition algorithms. A document may contain a number of font styles, but generally most of the document is printed using only one predominant font. After having determined the predominant font of the document, an OCR algorithm trained with respect to this font is applied.

Our goal in this study is to investigate the clustering behavior of an extensive set of fonts and to find the representative centroids of these clusters. One immediate application is a more robust OCR system, whereby once the font clusters are delineated, a bank of OCR algorithms can be trained with respect to each font cluster. The underlying scheme here is that the character recognition system will consist of a font cluster identifier unit followed by an OCR unit specific to each font group. A second application is the identification of each font when reproduction of a document in its original appearance is desired. In addition font recognition aids in the recognition of logical document structures. We conjecture also that for most of these tasks it suffices to determine the font cluster to which a text belongs, and that the cluster centroid can be used as the representing font.

The goals of the study can be itemized as follows:

- to find a minimal set of font clusters that provides adequate OCR performance across all fonts;
- to determine the representative (centroid) for each font cluster; and
- to develop an algorithm to detect the font type or font cluster in a document.

There have been few studies on font recognition. Notably Morris[1] has considered classification of digital typefaces using spectral signatures. Khoubyari and Hull[2] have considered a scheme based on the identification initially of such function words as ‘‘the,’’ ‘‘and,’’ ‘‘to’’ etc., and followed by template matching. This method, however, is too language specific. In Refs. 3 and 4 Zramdini and Ingold consider a font identification approach based on the projection profiles. The Bayesian classifier developed by the au-

Paper 99042 received July 29, 1999; revised manuscript received Oct. 4, 2000; accepted for publication Oct. 4, 2000.
1017-9909/2001/$15.00 © 2001 SPIE and IS&T.

|thors has a high performance rate. Zhu *et al.*[5] consider texture features (Gabor) to classify fonts in oriental characters. Jung *et al.*[6] address the multifont classification (seven fonts) using typographical attributes. Shi and Pavlidis[7] show the increase in OCR accuracy with font recognition.

The emphasis of our work is on the clustering behavior of fonts in order to simplify document analysis and OCR tasks. We show that the sensitivity of the OCR performance to font type can be mitigated significantly by font cluster recognition.

A font clustering scheme will be considered adequate if the span of the OCR performance differential within a cluster is small. In other words, within a cluster, when font i is used instead of j to recognize a text, there should not be an unacceptable drop in the correct recognition rate. If cluster-based recognition algorithms were not used, the performance differential is much greater when there is a mismatch between the fonts of the template and the test characters. In addition subjective plausibility based on the appearance of the grouped fonts in the cluster can be used.

This paper is organized in five sections. Preliminary information about font classes and document preprocessing are given in Sec. 2. Section 3 lists the various character and font features to be considered in clustering exercises. Font distance measures and clustering algorithms are presented in Sec. 4. The outcome of this section is the determination of the most suitable font features and font clustering method with a discussion on their comparative performance. OCR error performance within a pool of all fonts and within the pools of each font cluster, respectively, are discussed in Sec. 5. Section 6 presents clustering results when applied on fonts commonly used in daily newspapers. Finally Sec. 7 gives the concluding remarks.

2 Preliminaries: Fonts and Typefaces

A font can be modeled by the following attributes (see Table 1)[3,4]:

- The font family like ''Times,'' ''Helvetica,'' and ''Courier.''
- The size expressed in typographic points.
- The weight of the font, having one of the following values: *light*, *normal*, or *bold*.
- The slope indicating the orientation of the letters main strokes. A font could be *roman* or *italic*.
- The spacing mode specifying the pitch of the characters. A font may have a fixed pitch (monospaced) or a proportional one. The latter class may have *condensed*, *normal*, or *expanded* spacing mode.
- The presence of serif, that is serif and sans serif fonts. For example Bookman, New Century Schoolbook, Palatino, and Times have decorative serifs, while Courier has heavy square serif. Finally Avant Garde is an example of a font without serif (sans serif).

In this study the font attributes of spacing mode and size are normalized. More specifically spacing mode is uniformed by delineating their bounding boxes in the image preprocessing stage. Kopec[8] has analyzed such other features as character set widths, sidebearings, and pair spacing adjustments (kerning). The character bitmaps are then size normalized via bilinear interpolation to a standard size of 32×32. Notice that the character size normalization via bilinear interpolation causes some slight distortion in the character shapes, e.g., thickening of longitudinal characters as illustrated in Table 2. The 32×32 standard size was chosen as a compromise size for 300 dpi scanning density. At 300 dpi a 32×32 size mask would correspond to a character size of 2.7 mm, which corresponds approximately to a lower case of 18 point size. All fonts with point sizes bigger and smaller than this, are therefore reduced or enlarged, respectively, to this standard size. At this size one has a 1024-dimensional feature vector if bitmap values are used. The fonts considered belong to 28 different font families and together with weight, serif, and slope varieties their total number amounts to $F=65$ as listed in Table 1. The sans serif fonts are the ones having a (ss) mark next to them.

For each font we considered $K=68$ different characters, consisting of the lower case letters, uppercase letters, and numerals, as shown in Table 3. A total of $F\times K=68\times65=4420$ character images of different fonts were prepared using a computer graphics program. In the sequel we use the following notation: $\boldsymbol{c}_k$: kth character feature vector (e.g., its bitmap in lexicographic ordering) in a generic font; $k=1,\ldots,K$. The size of this vector will be denoted as m; $\boldsymbol{c}_{pk}$: kth character feature vector depicted in the pth font; $p=1,\ldots,F$, $\boldsymbol{c}^r_{pk}$: kth character feature vector depicted in the pth font which is assigned to the rth cluster; $r=1,\ldots,R$.

A specific feature component will be addressed as $c_k(i)$, $i=1\ldots m$ [and similarly, as $c_{pk}(i)$ and $c^r_{pk}(i)$]. The feature vector describing a font is indicated by ϕ, while the cluster centroid feature vector by γ. These concepts will be explained in the following sections.

3 Character and Font Features

There are many possible feature sets to be extracted from characters for recognition and clustering purposes. Several of these features such as bitmaps, zoning features, Fourier descriptors, Zernike moments, DCT coefficients, Fourier coefficients, Karhunen–Loeve features (eigencharacters), contour profiles, geometric moment invariants, projection histograms, and unitary image transforms have been assessed in Ref. 9. In this paper we tested the following prominent features from Ref. 9: bitmaps, DCT coefficients, eigencharacters, and Fourier descriptors.

3.1 *Character Bitmaps*

The binary pattern of a character in font k will be indicated by $c_k(i)$, $i=1,\ldots,1024$ in the lexicographic ordering of its 32×32 bitmap. The jitter, which is due to the finite resolution grid and interpolation, can be remedied by shifting the position of the template in four directions by one pixel: up, down, right, and left. Thus five templates are considered (itself and its four shifted versions), their match scores are calculated, and the maximum has been chosen for the subsequent analysis.

Table 1 List of the $F=65$ fonts as exemplified by character "*K*". The fonts will be referred to in the clustering experiments by their ID number. The sans serif characters are denoted by the (ss) symbol next to the font name. It is implied that the fonts without (ss) are all serif fonts.

ID	Font	
0	Aardvark Normal	K
1	Aardvark Normal Italic	*K*
2	Aardvark Bold	**K**
3	Aardvark Bold Italic	***K***
4	Arial Normal (ss)	K
5	Arial Normal Italic (ss)	*K*
6	Arial Bold (ss)	**K**
7	Arial Bold Italic (ss)	***K***
8	McLean Bold	**K**
9	McLean Bold Italic	***K***
10	Avantgarde BT Bold (ss)	**K**
11	Avantgarde BT Bold Italic (ss)	***K***
12	Korinna BT Bold	**K**
13	Korinna BT Bold Italic	***K***
14	Book Antiqua Bold	**K**
15	Victorian Normal	K
16	Victorian Normal Italic	*K*
17	Victorian Bold	**K**
18	Victorian Bold Italic	***K***
19	Windsor BT Normal	K
20	Windsor BT Normal Italic	*K*
21	Century Gothic Bold (ss)	**K**
22	Century Gothic Bold Italic (ss)	***K***
23	ZapfChan Md BT Bold	**K**
24	Century Schoolbook Bold	**K**
25	Century Schoolbook Bold Italic	***K***
26	Comic Sans MS Normal (ss)	K
27	Trebuchet MS Bold (ss)	**K**
28	Trebuchet MS Bold Italic (ss)	***K***
29	Garamond Bold	**K**
30	Avalon Bold (ss)	**K**
31	Avalon Bold Italic (ss)	***K***
32	Souvenir BT Bold	**K**
33	Georgia Bold	**K**
34	Georgia Bold Italic	***K***
35	Impact Normal (ss)	K
36	Impact Normal Italic (ss)	*K*
37	Impact Bold (ss)	**K**
38	Impact Bold Italic (ss)	***K***
39	Revue BT Normal (ss)	K
40	Revue BT Normal Italic (ss)	*K*
41	Revue BT Bold (ss)	**K**
42	Revue BT Bold Italic (ss)	***K***
43	Playbill BT Bold	**K**
44	City Bold	**K**
45	City Bold Italic	***K***
46	Square 721 BT Normal Italic (ss)	*K*
47	Square 721 BT Bold (ss)	**K**
48	Square 721 BT Bold Italic (ss)	***K***
49	Swiss 721 BT Normal (ss)	K
50	Swiss 721 BT Normal Italic (ss)	*K*
51	Swiss 721 BT Bold (ss)	**K**
52	Swiss 721 BT Bold Italic (ss)	***K***
53	Tahoma Normal (ss)	K
54	Tahoma Normal Italic (ss)	*K*
55	Tahoma Bold (ss)	**K**
56	Tahoma Bold Italic (ss)	***K***
57	News701 BT Normal	K
58	News701 BT Normal Italic	*K*
59	Times New Roman Bold	**K**
60	Mercurius Script MT Bold	***K***
61	Verdana Normal (ss)	K
62	Verdana Normal Italic (ss)	*K*
63	Verdana Bold (ss)	**K**
64	Verdana Bold Italic (ss)	***K***

3.2 *Fourier Descriptors*[9]

Fourier descriptors (FD) have been successfully used by many investigators for characterization of closed contours. A simple way to obtain Fourier features of a character is to calculate Fourier coefficients from the chain code of its closed contour. The chain code first described by Freeman approximates a continuous contour by a sequence of piecewise linear fits that consist of eight standardized line segments. In this study the cosinusoidal and sinusoidal components of the x and y projections of character chain codes have been used. For n number of harmonics the total feature set contains $m=4\times n$ features for each character.

3.3 *Eigencharacters*

While the bitmap of a character constitutes a 1024-element feature vector, using the Karhunen–Loeve method the sizeof these vectors can be reduced significantly, let's say to a size of $m=16$. The subspace features often prove to be more efficient in the classification and recognition tasks. On the other hand the subspace dimension $m=2$ avails us of the visualization potential of these multivariate data.

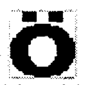
Let $\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \ldots, \boldsymbol{c}_F$ be the bitmap feature vectors of an $N\times N$ generic character image. The dimension of the original feature space (bitmap) is $d=N^2$. The F different realizations of the character bitmap may correspond to its various font realizations. The average of this set is defined by[10,4]

$$\Psi=\frac{1}{F}\sum_{n=1}^{F}\boldsymbol{c}_n \qquad (1)$$

and the corresponding covariance matrix is

$$\boldsymbol{C}=\frac{1}{F}\sum_{n=1}^{F}\boldsymbol{\Phi}_n\boldsymbol{\Phi}_n^T=\mathbf{A}\mathbf{A}^T, \qquad (2)$$

Table 2 Some character interpolation results.

Font	Before Interpolation	After Interpolation
Arial Bold	33 x 35	32 x 32
Georgia Bold Italic	29 x 36	32 x 32
Swiss 721 BT Normal	24 x 35	 32 x 32
Revue BT Normal Italic	21 x 36	32 x 32
Aardvark Bold Italic	33 x 36	32 x 32
Victorian Bold	11 x 28	32 x 32
ZapfChan Md BT Bold	16 x 39	32 x 32
McLean Bold Italic	40 x 34	32 x 32

where $\mathbf{A}=[\boldsymbol{\Phi}_1,\boldsymbol{\Phi}_2 \ldots \boldsymbol{\Phi}_F]$ and $\Phi_{\mathbf{i}}=\boldsymbol{c}_i-\Psi$. The eigenvectors of the covariance matrix $\boldsymbol{C}$ define an orthogonal linear space. These eigenvectors $\{\boldsymbol{u}_1,\boldsymbol{u}_2,\ldots,\boldsymbol{u}_d\}$ are ranked so that $\lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_d$. A linear subspace is created by selecting the set of m ranked eigenvectors $\boldsymbol{u}_1,\boldsymbol{u}_2,\ldots,\boldsymbol{u}_m$, where $\lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_{\boldsymbol{m}}$, which can also be called eigencharacters. The $m\times d$ matrix of transformation $\boldsymbol{H}_{\boldsymbol{m}}$ defined $\boldsymbol{H}_{\boldsymbol{m}}=[\boldsymbol{u}_1,\boldsymbol{u}_2 \ldots \boldsymbol{u}_m]^T$ can be used for feature extraction. In fact this matrix projects the d dimensional pattern space onto an m-dimensional ($m \ll d$) subspace whose axes are in the directions of the largest eigenvalues of $\boldsymbol{C}$ as follows:

$$\boldsymbol{y}_{\boldsymbol{i}}=\boldsymbol{H}_{\boldsymbol{m}}\Phi_l \quad \text{for } i=1,2,\ldots,F, \tag{3}$$

where the covariance of the projected patterns becomes a diagonal matrix. We will call the projections y_i as eigenfeatures. As is well known the sum of the first m eigenvalues shows the ‘‘power’’ retained in the new space while the sum of the d eigenvalues is the total variance in the original pattern space. For example for $m=2$ and $m=30$, respectively, the retained power has been found to be 33% and 90%, on the average, for all characters. It is interesting to note that less than the first 0.2% of the eigenfeatures represents one third of the total variance.

Since d is typically very large (e.g., 1024) the eigenvalue problem appears to be computationally complicated. Fortunately, we can solve for d-dimensional eigenvectors by first solving for the eigenvectors of an $F\times F$ matrix, where the number F of different fonts is an order of magnitude or more smaller than the dimensionality of the space d.

Consider the eigenvectors $\boldsymbol{\nu}_i$ of $\mathbf{A}^T\mathbf{A}$ (which is F by F) such that[10]

$$\mathbf{A}^T\mathbf{A}\boldsymbol{\nu}_i=\mu_i\boldsymbol{\nu}_i \quad \text{where } \mathbf{A}=[\boldsymbol{\Phi}_1\boldsymbol{\Phi}_2 \ldots \boldsymbol{\Phi}_F]. \tag{4a}$$

Premultiplying both sides by $\mathbf{A}$, we have

$$\mathbf{A}\mathbf{A}^T\mathbf{A}\boldsymbol{\nu}_i=\mu_i\mathbf{A}\boldsymbol{\nu}_i \tag{4b}$$

from which we see that $\mathbf{A}\boldsymbol{\nu}_{\mathbf{i}}$ are the eigenvectors of $\boldsymbol{C}=\mathbf{A}\mathbf{A}^T$.

Table 3 The $K=68$ characters considered in the experiments.

0	1	2	3	4	5	6	7	8	9
a	b	c	ç	d	e	f	g	h	i
j	k	l	m	n	o	ö	p	q	r
s	t	u	ü	v	w	x	y	z	A
B	C	Ç	D	E	F	G	H	I	J
K	L	M	N	O	Ö	P	Q	R	S
T	U	Ü	V	W	X	Y	Z		

3.4 *DCT Features*

The DCT transform is defined as

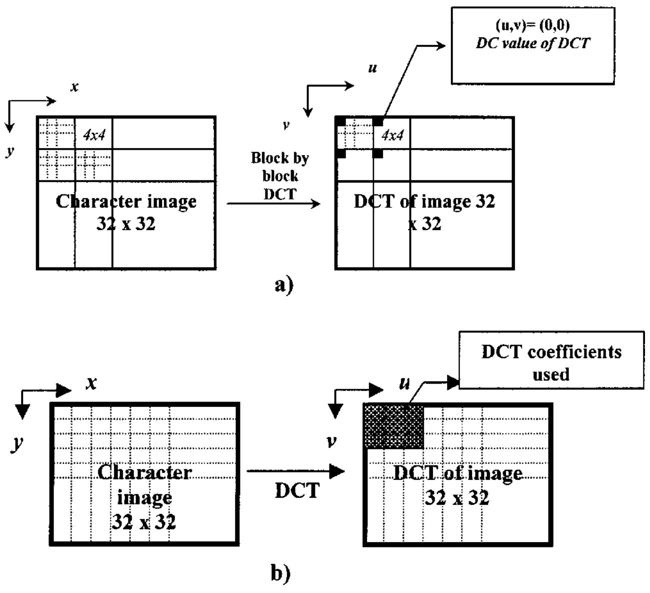


Fig. 1 Illustration of the DCT features of the character images: (a) DCT of 4×4 blocks where only DC values are chosen to form a 64-element feature vectors; (b) DCT of the whole 32×32 character image with the lowest nine coefficients chosen as features.

$$C(u,v)=\frac{2\beta(u)\beta(v)}{b}\sum_{t=0}^{b-1}\sum_{n=0}^{b-1}c(t,n)\cos\left(\frac{2t+1}{2b}u\pi\right)\times\cos\left(\frac{2n+1}{2b}v\pi\right)\quad u,v=0,1,\ldots,b-1 \tag{5}$$

and

$$\beta(k)=\frac{1}{\sqrt{2}}\quad\text{for}\quad k=0$$

and

$$\beta(k)=1\quad\text{otherwise,}$$

where b is the block size. Here, with some abuse of notation, we have used the double indexed $c(t,n)$ to denote the two-dimensional (2D) bitmap of features. The DCT feature vectors can be constructed in more than one way. Thus DCT can be executed on a block basis of size $b\times b$ where $b=4,8,16$ or over the whole character, i.e., $b=32$. In the latter case one encompasses the whole image. For $b<32$ one juxtaposes the individual DCT feature vectors of the blocks to form the whole character feature vector. In either case one selects the low order coefficients as features. For example, for $b=8$, if one selects the first four coefficients of each block DCT, one obtains a $m=4\times(32/8)^2=64$ element feature vector. Recall that for a block size $b\times b$, one ends up with a total number of $(32/b)\times(32/b)$ blocks. Figure 1 illustrates two choices with blocks of size $b=4$ and $b=32$.

3.5 *Font Features*

Font feature vectors are constructed by serial juxtaposition of the character feature vectors in some order. The dimensionality of the font feature vector adds up then to $D=K\times m$, where K is the number of characters and m is the number of features per character. Any ordering of character features can be made; in the case of alphanumeric ordering the font feature vector appears as $\phi=[\mathbf{a}\ldots\mathbf{z}\,\mathbf{A}\ldots\mathbf{Z}\,\mathbf{0}\ldots\mathbf{9}]^T$, where, e.g., **a** denotes the bitmap of the letter a and so forth. Thus for the bitmap features one has $D=K\times N^2=69\,632$ dimensional font feature vector, while for the DCT features with $b=32$ and $m=36$ coefficients per character one has $d=2448$.

Characters reconstructed using subspace features obtained via feature selection as in the case of DCT coefficients are displayed in Fig. 2(b). Characters reconstructed via eigencharacters, that is features extracted via the Karhunen–Loeve procedure, are shown in Fig. 2(a).

4 Clustering of Fonts

Cluster analysis is the process of classifying objects into subsets that have meaning in the context of a particular problem. The objects are thereby organized into an efficient representation that characterizes the population being sampled. Since there is no ''best'' clustering algorithm, we considered combinations of several data representation styles and clustering algorithms.

4.1 *Distance Functions*

Clustering algorithms depend critically upon the definition of the distance function between feature vectors. In the sequel we first explain the distance function for bitmap features. Let $c_{pk}(i,j)$ represent the bitmap at coordinates (i, j) of the kth character in the pth font style. The difference of the binary bitmaps of a character using two different font styles, p and q, are computed using the XOR operation:

$$X_{pq,k}(i,j)=c_{pk}(i,j)\oplus c_{qk}(i,j)\quad i,j=1,\ldots,32. \tag{6a}$$

In this 32×32 pattern we assigned a weight $w(i,j)$ to each bit in a contiguous groups of bits based on its number of neighbors to reflect the importance of mismatched pixels. This weighting is shown in Table 4. The weighted Hamming distance between any two character images k created in fonts p and q, respectively, is calculated as follows:

$$H(\mathbf{c}_{pk},\mathbf{c}_{qk})=\sum_{i,j}[w(i,j)X_{pq,k}(i,j)]. \tag{6b}$$

To avoid any sensitivity to the sampling grid and character jitter, the XOR pattern is obtained at five different displacements (center, one pixel up, down, left, and right) and the one showing the least difference is selected. After these comparisons, we obtain a proximity matrix for each character. For F fonts, the proximity matrix for each character will be a $F\times F$ symmetric matrix as illustrated in Table 5.

Averaging font-to-font proximity matrices over each of the K characters, we obtain the $F\times F$ font proximity matrix:

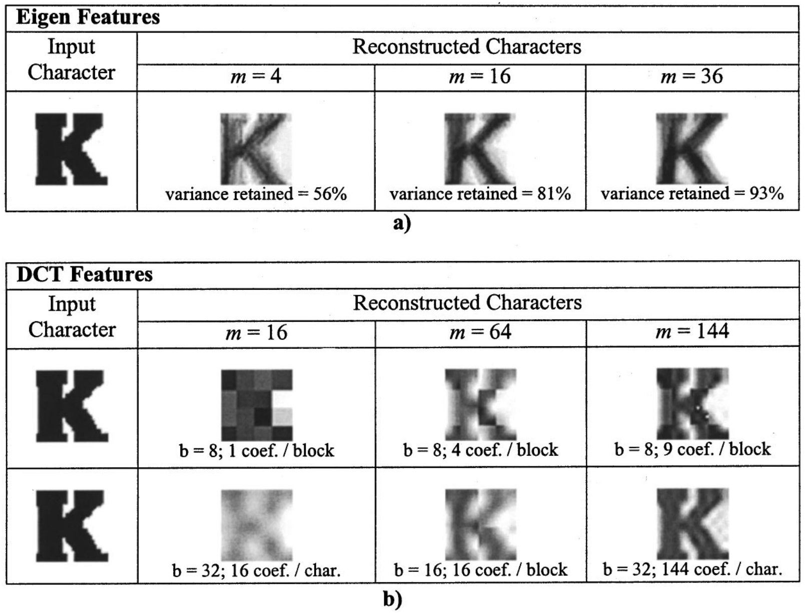


Fig. 2 Some character reconstruction results using: (a) eigencharacters and (b) DCT features.

$$D(\text{font}_p,\text{font}_q)=\sum_{k=1}^{K} p(\text{char}_k)H(\text{char}_k/\text{font}_p,\text{char}_k/\text{font}_q)$$
$$=\sum_{k=1}^{K} P(\boldsymbol{c}_k)H(\boldsymbol{c}_{pk},\boldsymbol{c}_{qk}), \tag{7}$$

where K is the number of characters used in each font and P is the character occurrence probability. When the character probabilities were taken according to their probability of occurrences in the English language, it was noticed that in this case characters occurring with low probability (such as z and j) have negligible effect in clustering. Instead clustering was dominated by the high probability characters like e. A more satisfactory font distance function was obtained by considering equiprobability characters.

The distance between clusters can be defined in terms of interfont distances. The three ways of obtaining cluster distances that we investigated are as follows:

- *Mean distance*

$$D_{\text{mean}}(\text{cluster}_r,\text{cluster}_s)$$
$$=\frac{1}{N_rN_s}\sum_{p=1}^{N_r}\sum_{q=1}^{N_s} D(\text{font}_p,\text{font}_q), \tag{8a}$$

where N_r and N_s are the number of fonts in clusters r and s, respectively.

- *Centroid distance*
  Let cent^r be the centroid font of the cluster r

$$D_{\text{center}}(\text{cluster}_r,\text{cluster}_s)=D(\text{cent}^r,\text{cent}^s). \tag{8b}$$

- *Nearest-neighbor distance*

$$D_{\text{nr}}(\text{cluster}_r,\text{cluster}_s)=D(\text{font}_p^*,\text{font}_q^*), \tag{8c}$$

where font_p^* and font_q^* are determined as $\arg\min_{p\in\text{cluster }r,q\in\text{cluster }s}D(\text{font}_p^*,\text{font}_q^*)$; that is the fonts in the classes r and s, respectively, with the minimum mutual distance. Alternatively the nearest neighbor distance can be defined between fonts in a cluster and the centroid of the other class

$$D_{\text{nr}}(\text{cluster}_r,\text{cluster}_s)$$
$$=\max\{\min_p D(\text{font}_p^r,\text{cent}^s),\min_q D(\text{font}_q^s,\text{cent}^r)\}.$$

For features other than bitmaps, i.e., Fourier descriptors, DCT coefficients, and Karhunen–Loeve (KL) features,

Table 4 Weighting used in the Hamming distance.

Number of neighbors	Weight $W(i,j)$
0	1
1	1
2	2
3	3
4	4

Table 5 Proximity matrix for generic character char_k.

Char_k	Font_1	Font_2	⋯	Font_n	⋯	Font_F
Font_1	0	$H(1,2)$	⋯	$H(1,n)$	⋯	$H(1,F)$
Font_2	⋯	0	⋯	$H(2,n)$	⋯	$H(2,F)$
⋯	⋯	⋯	0	⋯	⋯	⋯
Font_n	⋯	⋯	⋯	0	⋯	⋯
⋯	⋯	⋯	⋯	⋯	0	$H(F-1,F)$
Font_F	⋯	⋯	⋯	⋯	⋯	0

Eqs. (7) and (8) remain valid. For nonbinary features, Eqs. (6a) and (6b) should be interpreted as the Euclidean distances of character feature vectors, that is,

$$H(\boldsymbol{c}_{pk},\boldsymbol{c}_{qk})=\sum_{i=1}^{m}(c_{pk}(i)-c_{qk}(i))^2,$$

which is the Euclidean distance of the feature vectors of the kth character in fonts p and q, respectively, and where again m is the number of features considered.

4.2 *Cluster Centroids*

Based on the interfont distance the centroid of each cluster can be calculated. A cluster centroid has been shown in Eq. (8b) symbolically by cent^r, for some cluster r. The feature vector pertaining to this centroid is denoted by γ^r. The centroid of a font cluster can be chosen in one of the following manners:

Method 1: For the cluster r, using the font proximity matrix, the font which has the smallest summed distance to all other fonts is assigned as the centroid font. The feature vector of the centroid γ^r becomes simply the feature vector ϕ_p^r of the selected font

$$\text{cent}^r=\text{font}_p^r,$$
$$\text{where } p \text{ is selected as } \arg\min_p \sum_{\substack{p,q\\p\neq q}}^{N_r} D(\text{font}_p^r,\text{font}_q^r). \tag{9a}$$

Method 2: Per character centroid can be chosen. Thus one applies the algorithm above in Eq. (9a) not on the d-sized ($d=K\times m$) font feature vector, but on each of the m-sized character feature vectors for each character separately. In other words a centroid font for each character is identified

$$\text{cent}^r=[\text{cent}_1,\text{cent}_2,\ldots,\text{cent}_K], \tag{9b}$$

where the $\text{cent}_i=1,\ldots,K$, denote the character centroids in that class with N_r fonts. In this case the font centroid vector will consist of K centroid vectors, each of size m, one for each character, possibly coming from a different font.

Method 3: The centroid character of a given cluster r with N_r font styles cent_k^r is obtained by averaging and thresholding the bitmaps of the characters in different font styles in that cluster, that is

$$\text{cent}_k^r(i,j)=\begin{cases}1 & \text{if } \dfrac{1}{N_r}\sum_{p=1}^{N_r} c_{pk}^r(i,j)\geqslant 0.5\\ 0 & \text{else}\end{cases}\quad i,j=1,\ldots,N \text{ and } k=1,\ldots,K, \tag{9c}$$

where $c_{pk}^r(i,j)$ denotes the (i,j)th bit of the kth character with font style p, which is in the cluster r. Notice that this type of averaging of binary images is also called the Vorobev mean.[11] On the other hand the font created using the mean of the feature vectors corresponds to the minimum Euclidean distance case. In either case of method 3 a new centroid font is being created as opposed to methods 1 and 2 where fonts or characters are being picked up among existing fonts.

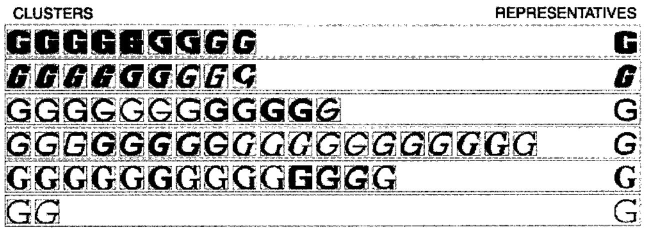


Fig. 3 Clustering result with minimum average distance [centroids by Eq. (9b)]. $\eta_{\text{eigen}}=2.292$.

In the sequel we describe a number of font clustering experiments. In these experiments we denote also the clustering performance figure η to be defined later in Eq. (14).

4.3 *Clustering with a Linkage Algorithm*

Using the font proximity matrix, the font pairs having minimum average distance are merged and a new cluster is formed. After each merge step the proximity matrix in Table 5 is updated by defining the new distances between clusters and the number of rows and columns of the proximity matrix are decreased by one. As clusters become sizeable, the single linkage method suffers from false merges. To preclude such false growth, we have used the average distance function Eq. (8a) D_{mean} as the distance between clusters instead of the more conventional nearest-neighbor distance. The algorithm first starts by merging the closest pair of fonts; in the sequel all intercluster distances are updated using average distance function.

The results of this clustering algorithm for six clusters are shown in Fig. 3, where the representative characters (centroids) are computed using the centroid algorithm 2, Eq. (9b). The order of fonts in Fig. 3 from left to right in each cluster is the same as the sequel of successive merges. One can see that initially the fonts in each cluster are very close to each other, but as the sizes of clusters increases disparateness in the group increases.

4.4 *Clustering Using Mean-Square Distance*

The previous algorithm based on average cluster distance causes some unbalanced populations. We have resorted to the mean square Hamming distance between clusters defined as follows:

$$D(\text{cluster}_r,\text{cluster}_s)=\frac{N_rN_s}{N_r+N_s}\left|\frac{1}{K}\sum_{k=1}^{K}[D_{\text{mean}}(\boldsymbol{c}_k^r,\boldsymbol{c}_k^s)]^2\right|^{1/2}, \tag{10}$$

where the two clusters r and s contain N_r and N_s fonts, respectively. Finally, the two clusters r and s, which have minimum $D(\text{cluster}_r,\text{cluster}_s)$ distance are merged to form a new cluster t. The output of this algorithm for six classes is shown in Fig. 4. It can be seen that the mean square Hamming distance is effective in forming, subjectively, more balanced clusters, although its η performance figure was slightly higher.

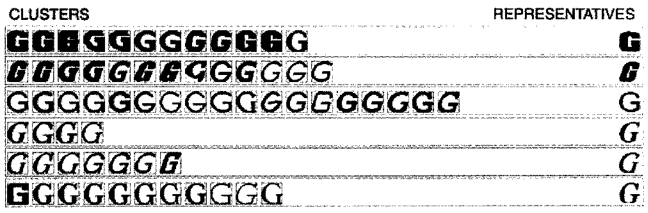


Fig. 4 Clustering with average mean square Hamming distance [centroids by Eq. (9b)]. $\eta_{\text{eigen}}=1.992$.

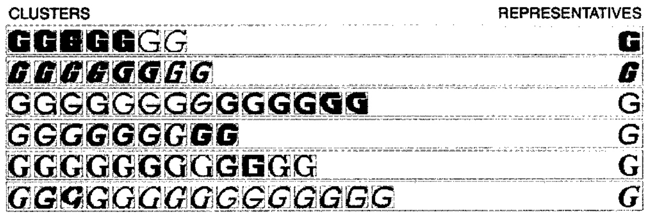


Fig. 6 Clustering with Ward's algorithm using 20 principal components of the bitmap features $\eta_{\text{eigen}}=1.628$.

4.5 *Clustering with Ward's Algorithm*

Recall that Ward's algorithm aims to minimize the within-cluster variation and this action is equivalent to maximizing the between-cluster variation.[12] Suppose that we have N_r patterns in a d-dimensional space and let ϕ^r_{pj} be the jth feature of the pth pattern (font) in cluster r and let N_r be the number of patterns in the cluster r, $r=1,\ldots,R$ and $j=1,\ldots,d$.

The centroid of cluster r, denoted $\gamma^r=[\gamma^r(1)\ldots\gamma^r(d)]$ over all the fonts is the average of the N_r patterns

$$\gamma^r(j)=\frac{1}{N_r}\sum_{p=1}^{N_r}\phi^r_p(j), \tag{11}$$

where ϕ^r_p was the font feature vector as defined in Sec. 3.5 consisting of stacked up character feature vectors.

The square error for cluster r is the sum for all patterns in cluster r of the squared distances to the centroid, and the square error for entire clustering, which contains R clusters, is the sum of the square errors (within-scatter) of the individual clusters

$$e_r^2=\sum_{p=1}^{N_r}\sum_{j=1}^{d}[\phi^r_p(j)-\gamma^r(j)]$$

and

$$E_R^2=\sum_{r=1}^{R}e_r^2. \tag{12}$$

Ward's method merges the pair of clusters that minimizes ΔE^2_{rs}, which is the change in E^2_R caused by the merger of clusters r and s into cluster t to form the next clustering. Since the square errors for all clusters except for the three clusters involved remain the same, ΔE^2_{rs} can be calculated as follows:

$$\Delta E^2_{rs}=e_t^2-e_r^2-e_s^2, \tag{13a}$$

which can be calculated using only the centroid information

$$\Delta E^2_{rs}=\frac{N_rN_s}{N_r+N_s}\sum_{j=1}^{d}[\gamma^r(j)-\gamma^s(j)]^2. \tag{13b}$$

The clusters r and s that are selected and merged are those two that minimizes this quantity ΔE^2_{rs}. The result of Ward's algorithm using full bitmap feature vectors; i.e., feature length $d=1024$, is given in Fig. 5.

The bitmap features reduced via the KL procedure are similarly clustered using Ward's algorithm as shown in Fig. 6. Figure 7 illustrates the organization of ten font clusters plotted in the plane formed from the first two eigenfeatures. We also tested Ward's algorithm on font feature vectors consisting of DCT coefficients. We experimented with DCT feature vectors obtained in various ways, e.g., block sizes of $b=4,8,16,32$ with different numbers of retained coefficients. For example for $b=4$ and maintaining only a DC value one ends up with 64 features per character. Conversely one can chose $b=32$ and consider the 64 lowest

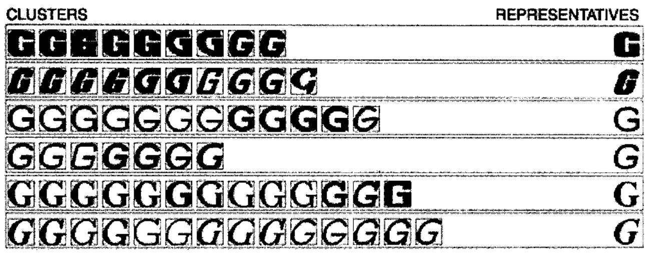


Fig. 5 Clustering with Ward's algorithm on 1024-dimensional bitmaps [centroids using Eq. (9b)] $\eta_{\text{eigen}}=1.627$.

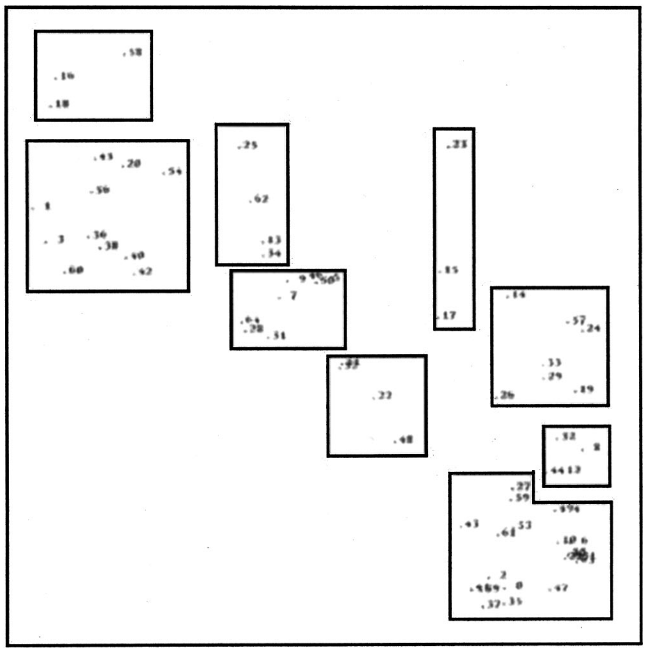


Fig. 7 Clustering of the 65 fonts using Ward's algorithm projected on the plane of the first two principal components.

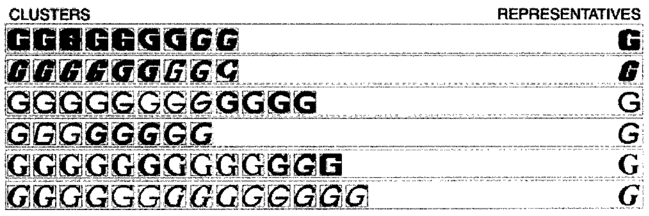


Fig. 8 Ward's algorithm on font feature vectors obtained by DCT using 32×32 block size and taking the first 36 coefficients (font feature vector length=2448). $\eta_{\text{eigen}}=1.605$.

indexed coefficients. A comparative study revealed that full block size, i.e., covering the whole character image, yields better clustering performance.

Figure 8 illustrates the outcome of Ward's algorithm using full block size ($b=32$) and 36 coefficients. This results in $36\times 68=2448$ dimensional features per font.

4.6 *Performance Comparisons of Clustering Algorithms*

An extensive set of experiments have been carried out using different clustering techniques and different feature sets. Some sample outcomes have been illustrated in Figs. 3–8. Among several options for cluster performance metrics we have opted for the well-established index[13] that compares the ratio of mean of within cluster variances to between cluster variances.

Let ϕ_p^r be the feature vector of font p in cluster r, γ^r be the centroid of cluster r, γ_0 be the centroid of all clusters, C be the total number of clusters, and N_r be the number of fonts in cluster r. Then we define our comparison index η as follows:

$$\eta=\frac{(1/C)\Sigma_{r=1}^{C}\Sigma_{p=1}^{N_r}(\phi_p^r-\gamma^r)^2}{\Sigma_{r=1}^{C}(\gamma^r-\gamma_0)^2}. \tag{14}$$

Obviously the smaller the η index, the better is the clustering performance. The comparison of the performance according to the η index is given in Table 6. The goal of these experiments is to establish the more favorable features and the method between linkage and Ward's algorithms. All the experiments on the centroid calculations are not reported in Table 6 to make it simpler. It was observed that among the three methods [Eqs. (9a), (9b), and (9c)] to calculate the centroid, the first and second ones always gave better results, with Eq. (9b) only slightly better than Eq. (9a).

In this table the column D denotes the size of feature vector used per font. Obviously this quantity is given by the product of the number of features per character and the number of characters, $D=m\times K$. To interpret these tables note that the indices η with subscript, i.e., η_{eigen}, η_{DCT}, η_{bitmap}, η_{FD} indicate the features with respect to which index was calculated. For example, in the first row of the table the clustering was carried out with respect to the 64 KL coefficients (eigencharacters). The performance was measured in η_{eigen} with respect to the 64 eigencoefficients, in η_{DCT} with respect to DCT coefficients ($b=8,16$ coefficients per block), in η_{bitmap} with respect to 1024 binary pixel values, and in $\dot{\eta}_{\text{FD}}$ with respect to 100 Fourier descriptors. The idea here is to crosscheck the features, that is to observe whether characters clustered with respect to one specific feature results in good index scores with respect to other features.

Table 6 Comparison results of clustering methods; (a) comparison of results for the linkage algorithm and (b) comparison of results for Ward's algorithm.

Feature	D	η_{eigen}	η_{DCT}	η_{bitmap}	η_{FD}
(a) 64 KL	64	0.382	1.217	0.751	0.793
20 KL	20	0.437	1.285	0.792	0.821
2 KL	2	0.478	1.288	0.836	0.856
DCT, $b=8$, 4 coef.	4352	0.454	1.168	0.708	0.676
DCT, $b=32$, 36 coef.	2448	0.423	1.150	0.691	0.657
DCT, $b=16$, 25 coef.	6800	0.448	1.152	0.698	0.702
BITMAP	69632	0.595	1.296	0.675	0.746
40 Fourier descriptors	2720	0.948	1.155	0.911	0.591
(b) 64 KL	64	0.324	1.275	0.698	0.692
20 KL	20	0.350	1.350	0.712	0.704
2 KL	2	0.484	1.367	0.832	0.747
DCT, $b=8$, 4 coef.	4352	0.422	1.204	0.754	0.728
DCT, $b=32$, 36 coef.	2448	0.401	1.173	0.729	0.706
DCT, $b=16$, 25 coef.	6800	0.418	1.194	0.748	0.728
BITMAP	69632	0.328	1.452	0.657	0.721
40 Fourier descriptors	2720	0.558	1.241	0.716	0.552

One conclusion that can be drawn from this table is that the Ward's algorithm results in a better performance as compared to the linkage algorithm. Furthermore clustering with respect to the eigenfeatures (KL features) results in the most compact representation.

We also determined whether clustering was sensitive to the serif–sans serif nature of the characters. The serif and sans serif characters almost always appeared in different clusters whichever feature was used (i.e., DCT, eigen, FD, bitmap). For example, using Ward's algorithm with DCT features there were only two occurrences out of 65 of a serif character with other sans serifs.

5 OCR Performance and Font Classification

5.1 *Inter- and Intracluster OCR Performance*

It is interesting to investigate how OCR performance varies across different fonts. The extent of this variation would provide a justification for font classification. To this purpose texts were created with font f_i and subjected to character recognition using templates from another, font f_j. Thus the performance differentials were measured using template matching on these texts. The reason why template matching was used is that it would be too costly to train other algorithms, e.g., artificial neural network based classifier for each and every font case. Furthermore template-based recognition is needed anyway as a front-end classifier for merged character detection (see Fig. 9).[14]

Using the ten clusters resulting from Ward's algorithm with full bitmap features, sample texts were prepared. More specifically text images were recreated with the font of each centroid and character recognition based on template

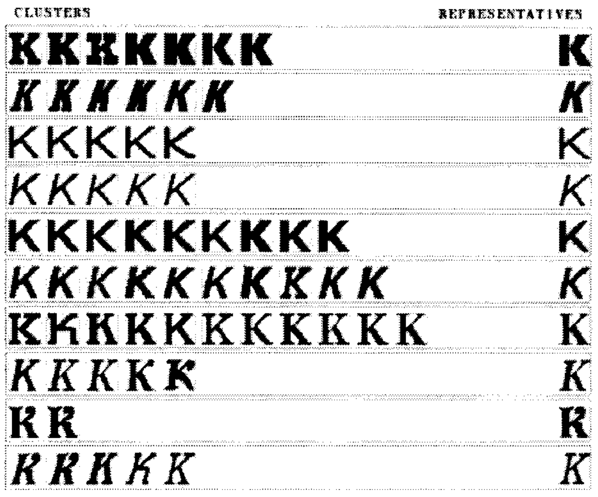


Fig. 9 Ten font clusters obtained using Ward's algorithm.

matching where the centroid font of all the other clusters was carried out. The recognition percentages are given in Table 7.

Notice that the scores on the diagonal are highest as the text and template fonts match. These scores are not all 100% because characters suffer some distortion during size normalization. One can observe that in the off-diagonal terms there is unacceptably large performance deterioration due to font mismatches. For example, the correct recognition performance in row 2 of Table 7 shows that the performance falls from 98.6% to anywhere between 68.8% and 20.6%. Such OCR performance losses prove the importance of font recognition.

A second question to be addressed is the intracluster variation of the OCR performance. In other words the correct recognition scores were observed with texts created with a font in that cluster and recognized with templates of another font also in that cluster. Consider for example cluster number 4 and its crossfont OCR performance in Table 8. As expected the performance differential across fonts in the same cluster is much more mitigated. The poorest performance across any row remains above 73.6, and usually much above. Recall that these results are obtained with the minimum effort, most simplistic character recognition, to put into evidence the relative importance of font cluster recognition.

To further emphasize this point we calculated the average performance drop in Table 7 and in Table 8. Thus we have considered the average of the differences between the diagonal term in each row and its off-diagonal terms. Obviously these differences are due to font mismatches in the template matching. The average discrepancy was 30.8% without clustering within a group of ten fonts as in Table 7, and only 8.1 within a cluster as in Fig. 8 (cluster of ten fonts). When all 65 fonts were considered (that is only one cluster of 65 fonts) the average discrepancy was 43.6%. In conclusion it can be stated that the recognition of the font cluster before the template matching based character recognition is beneficial.[15]

When we tested whether the centroid also yields the highest OCR score we discovered that the centroid template and the font that would result in the best average character were generally the same. For example for cluster 4, the centroid (Ward's algorithm using bitmap features) was font number 6 and the best OCR performing template was also number 6. One should note that no special effort was made to determine the optimum character classifier. A simple template matching was used in order to determine the differences in recognition with and without font identification.

5.2 *OCR Based Clustering of Fonts*

Motivated by the above investigation has led us to consider OCR performance-based clustering. In other words fonts that result in minimal performance differentials should be grouped together. To this purpose we estimated a 65$\times$65 correct recognition score matrix using a corpus of texts. The resulting 65$\times$65 matrix is not symmetric, therefore the mutual OCR rate between fonts p and q (that is text with font p and template q and vice versa) is selected as the minimum of the two scores when font p is matched with font q and vice versa. In this clustering scheme one selects a seed, preferably the highest scoring font, and joins to its group all such fonts whose performance differential remains below a threshold ε. Obviously the more one lowers the threshold, the greater the number of clusters the data should split into. One thus obtains a cluster with smaller

Table 7 Correct recognition percentages with template based recognition on fonts different than the text font.

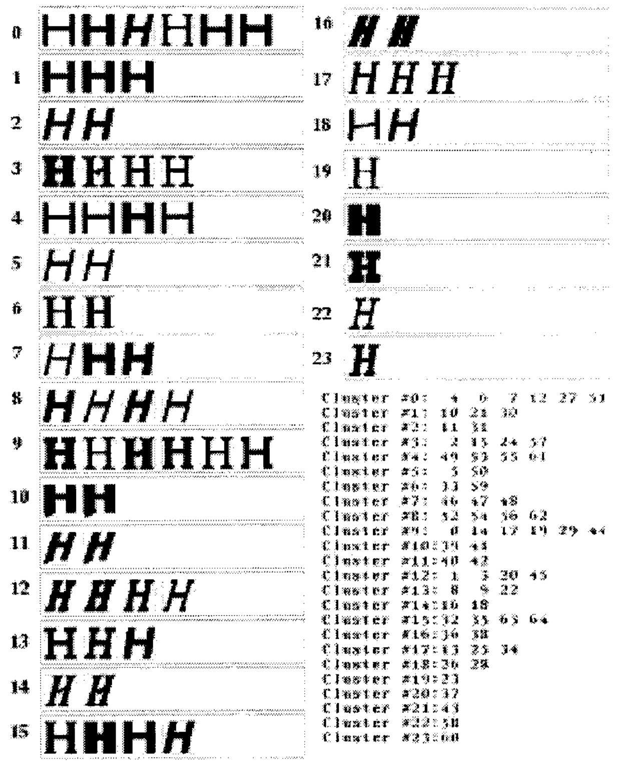
	Template font No. 0	Template font No. 1	Template font No. 2	Template font No. 3	Template font No. 4	Template font No. 5	Template font No. 6	Template font No. 7	Template font No. 8	Template font No. 9
Text font No. 0	**89.6**	55.6	71.8	78.6	35.8	81.8	76.4	65.2	79.6	26.8
Text font No. 1	64.4	**98.6**	20.6	58.2	38.8	46.0	58.8	50.4	47.4	68.8
Text font No. 2	69.0	39.2	**100.0**	77.2	94.8	72.0	73.6	52.2	53.8	14.4
Text font No. 3	74.4	76.2	83.4	**100.0**	78.8	88.4	76.8	67.0	70.8	60.8
Text font No. 4	93.2	52.0	94.8	81.4	**97.0**	96.0	79.4	69.0	86.8	41.0
Text font No. 5	64.6	88.0	74.8	91.6	87.0	**97.0**	71.6	67.2	83.0	82.8
Text font No. 6	84.6	64.2	64.4	81.2	78.6	83.2	**97.0**	88.4	73.8	24.4
Text font No. 7	64.4	71.6	37.2	76.2	50.0	72.6	68.0	**97.4**	66.0	56.6
Text font No. 8	59.4	34.6	58.8	73.8	77.0	77.0	80.6	76.6	**93.4**	48.0
Text font No. 9	40.6	70.4	41.0	64.4	43.8	51.4	36.8	59.8	66.8	**94.0**

Table 8 Intracluster OCR recognition rates for cluster No. 4. The cluster consists of fonts numbered 6, 51, 10, 21, 30, 27, 47, 57, and 63. The centroid font is 6, which is also used as a template.

	Font ID of the member font whose character set was used as templates								
Font ID's of the text images	**6**	**51**	**10**	**21**	**30**	**27**	**47**	**55**	**63**
6	97.0	97.0	78.2	86.6	86.6	94.0	88.4	94.8	94.8
51	96.2	97.0	86.0	83.4	83.4	89.4	78.8	77.8	77.8
10	86.0	78.6	97.0	97.0	97.0	81.4	78.2	77.8	77.8
21	82.0	74.4	91.6	92.4	92.4	81.2	73.6	76.8	77.2
30	86.0	78.8	97.0	93.4	97.0	81.4	77.8	85.6	85.6
27	96.6	96.8	86.2	77.0	84.0	100.0	95.0	96.2	96.2
47	87.0	88.2	69.4	80.0	69.6	86.2	89.4	76.2	85.2
55	95.6	94.8	85.4	87.8	86.4	94.2	94.4	100.0	100.0
63	95.6	95.6	85.6	78.0	86.4	94.2	94.2	100.0	100.0
Avg	**91.3**	89.02	86.26	86.18	86.98	89.11	85.53	87.24	88.29

populations and eventually each font becomes a ''separate cluster.'' The result of OCR based grouping for $\varepsilon = 10\%$ is shown in Fig. 10.

It is instructive to compare the OCR performance differentials with OCR-based clustering and with feature-based clustering. In both cases we considered 23 clusters, and in the feature-based technique we used Ward's method on KL features. They both had similar within-cluster performance drop averages of 8.47 and 8.84, respectively, for feature-based and OCR-based approaches. However in the feature-based case the individual performance differential scores varied between 0% and 33%, while in the OCR-based approach the worst case was limited by 10%.



```
Cluster #0:  4  6  7 12 27 31
Cluster #1: 10 21 30
Cluster #2: 11 31
Cluster #3:  2 15 24 57
Cluster #4: 49 53 55 61
Cluster #5:  5 50
Cluster #6: 33 59
Cluster #7: 46 47 48
Cluster #8: 52 54 56 62
Cluster #9:  0 14 17 19 29 44
Cluster #10:39 41
Cluster #11:40 42
Cluster #12: 1  3 20 45
Cluster #13: 8  9 22
Cluster #14:16 18
Cluster #15:32 35 63 64
Cluster #16:36 38
Cluster #17:13 25 34
Cluster #18:26 28
Cluster #19:23
Cluster #20:37
Cluster #21:43
Cluster #22:58
Cluster #23:60
```

Fig. 10 OCR-based clustering output for a performance differential threshold of $\varepsilon = 10\%$.

6 Clustering Applied on Real Newspaper Images

6.1 *Determination of the Predominant Font Class in a Document*

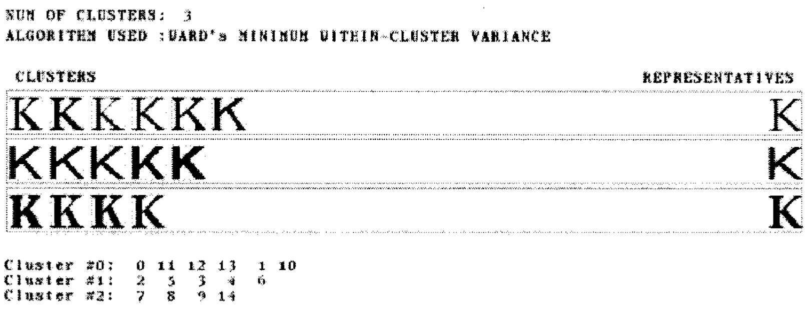
We assume that the document image has been segmented into text parts and within each text part characters are delineated by their bounding boxes. To determine which font class a text region belongs, we ran the template matching procedure. We selected characters randomly from the text and we incremented the score of the winning font by one, each time that font resulted in the highest match value. When the score of the highest scoring font exceeded that of its nearest competitor by a certain amount, we declared that font as the document font. In our work the score gap between the winner and its nearest competitor was chosen as 10 and the correct font was usually identified within 10–15 steps.

6.2 *Clustering of Press Fonts*

Documents like magazines and newspapers do not have a large variety of fonts. For example, the 15 fonts gleaned out of major Turkish newspapers are displayed in Table 9; and

Table 9 Fonts frequently encountered in newspapers and journals.

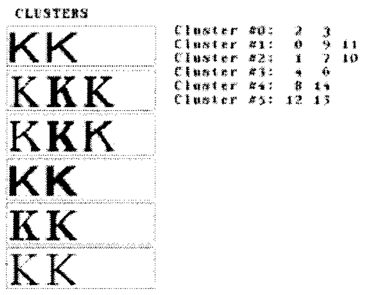
Font ID	Font		Font ID	Font	
0	Times New Roman	K	8	News701 BT	K
1	Souvenir Light BT	K	9	News701 BT Bold	K
2	Arial Normal	K	10	Korinna BT Normal	K
3	Verdana Normal	K	11	Georgia Normal	K
4	Trebuchet MS	K	12	Garamond Normal	K
5	Swiss 721 BT Normal	K	13	Book Antiqua Normal	K
6	Swiss 721 BT Bold	K	14	Book Antiqua Bold	K
7	Souvenir BT Bold	K			



```
NUM OF CLUSTERS:  3
ALGORITHM USED :WARD's MINIMUM WITHIN-CLUSTER VARIANCE
```

```
Cluster #0:  0 11 12 13  1 10
Cluster #1:  2  5  3  4  6
Cluster #2:  7  8  9 14
```

a)



```
Cluster #0:  2  3
Cluster #1:  0  9 11
Cluster #2:  1  7 10
Cluster #3:  4  6
Cluster #4:  8 14
Cluster #5: 12 13
```

b)

Fig. 11 (a) Three clusters of press fonts obtained with Ward's algorithm and DCT features; (b) OCR-based clustering of press fonts obtained with an error differential of $\varepsilon = 5\%$.

in Fig. 11 we show the clustering results both with Ward's algorithm on DCT features and the OCR-based clustering with $\varepsilon = 5\%$.

7 Conclusion

A font clustering and a font recognition algorithm have been developed and tested. A large selection of fonts have been clustered using various feature sets. Although all the tested features, i.e., DCT features, Fourier descriptors, bitmaps, and eigenfeatures result in adequate clustering performance, eigenfeatures result in the most parsimonious and compact representation.

Clustering of fonts has been effected with two different goals:

i. Clustering based on ''shape similarity'' using font features is needed when the problem domain is the recognition of logical document structures, document reproduction, document indexing, and information retrieval.

ii. Clustering based on ''differential error performance'' using character templates when the problem domain is merged character segmentation or enhancement of the recognition rate of the OCR system.

It has been shown that for all the fonts six to eight clusters prove adequate; on the other hand for the more limited set commonly used in daily newspapers three to four clusters were sufficient.

The font recognition algorithm can be used both to recognize font groups or individual fonts. In the latter case it is preferable to have a hierarchical approach, that is, to first recognize the font cluster and subsequently the specific font within that cluster.

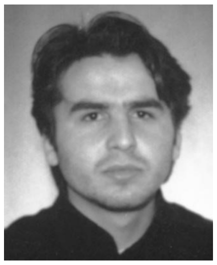
Acknowledgments

This project was supported by Boğaziçi University Research Fund Project 0234.

References

1. R. A. Morris, ''Classification of digital typefaces using spectral signatures,'' *Pattern Recogn.* **25**(8), 869–876 (1988).
2. S. Khoubyari and J. J. Hull, ''Font and function word identification in document recognition,'' *Comput. Vis. Image Underst.* **63**(1), 66–74 (1996).
3. A. Zramdini and R. Ingold, ''Optical font recognition using typographical features,'' *IEEE Trans. Pattern Anal. Mach. Intell.* **20**(8), 877–882 (1998).
4. A. Zramdini and R. Ingold, ''ApOFIS: an A priori optical font identification system,'' in *Proc. ICIAP'95, 8. Int. Conf Image Analysis Processing* (1995).
5. Y. Zhu, T. Tan, and Y. Wang, ''Font recognition based on global texture analysis,'' in *Proc. ICDAR'99*, pp. 349–354, Bangalore, India (1999).
6. M. C. Jung, Y. C. Shin, and S. N. Srihari, ''Multifont classification using typographical attributes,'' in *Proc. ICDAR'99*, pp. 353–356, Bangalore, India (1999).
7. H. Shi and T. Pavlidis, ''Font recognition and contextual processing for more accurate text recognition,'' in *Proc. ICDAR'97*, pp. 39–44, Ulm, Germany (1997).
8. G. E. Kopec, ''Least-squares font metric estimation from images,'' *IEEE Trans. Image Process.* **IM-2**(10), 510–518 (1993).
9. O. D. Trier, A. K. Jain, and T. Taxt, ''Feature extraction methods for character recognition,'' *Pattern Recogn.* **29**, 641–662 (1996).
10. M. Turk and A. Pentland, ''Eigenfaces for Recognition,'' *J. Cogn Neurosci.* **3**(1), 71–86 (19991).
11. T. Lewis, R. Owens, and A. Baddeley, ''Averaging feature maps,'' *Pattern Recogn.* **32**, 1615–1630 (1999).
12. A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, NJ (1988).
13. G. W. Milligan, ''Clustering validation: Results and implications for applied analysis,'' in *Clustering and Classification*, P. Arabie, L. J.

Hubert, and G. De Soete, eds., World Scientific, River Edge, NJ (1996).
14. R. G. Casey and E. Lecolinet, ''Survey of methods and strategies in character segmentation,'' *IEEE Trans.* **18**(7), 690–706 (1996).
15. N. Muller and B. Herbst, ''The use of eigenpictures for optical character recognition,'' in *Proc. 14th Int. Conf. Pattern Recognition*, pp. 1124–1126, Australia (1998).



Serdar Öztürk received his BS in electrical and electronic engineering from Boğaziçi University, Istanbul, Turkey in 1999 and is currently working toward his MSc in electrical engineering. He is also currently an engineer at Thomson-CSF, Istanbul, where his work deals with software for radar and air control systems. His main interests are image processing and multimedia security.

Bülent Sankur received his BS in electrical engineering at Robert College, Istanbul and completed his MSc and PhD degrees at Rensselaer Polytechnic Institute, New York. He has been teaching at Boğaziçi (Bosphorus) University in the Department of Electric and Electronic Engineering. His research interests are in the areas of digital signal processing, image and video compression, industrial applications of computer vision, and multimedia systems. Dr. Sankur has held visiting positions at University of Ottawa (Canada), Technical University of Delft (Holland), and ENST, France.

Toygar Abak received his BS in electronics and telecommunications engineering from Istanbul Technical University, Turkey in 1995, and his MS in electrical and electronics engineering from Boğaziçi University, Istanbul, Turkey in 1998. Since 1999, he has been a PhD student in computer engineering at Istanbul Technical University. He is currently a senior researcher at Information Technologies Research Institute, Marmara Research Center, TUBITAK, Turkey.