Electronic Letters on Computer Vision and Image Analysis 10(1):24-41, 2011

A Font Search Engine for Large Font Databases

Martin Solli and Reiner Lenz

ITN, Linköping University, SE-60174 Norrköping, Sweden

Received 15th Jun 2010; accepted 28th Dec 2010

Abstract

A search engine for font recognition is presented and evaluated. The intended usage is the search in very large font databases. The input to the search engine is an image of a text line, and the output is the name of the font used when rendering the text. After pre-processing and segmentation of the input image, a local approach is used, where features are calculated for individual characters. The method is based on eigenimages calculated from edge filtered character images, which enables compact feature vectors that can be computed rapidly. In this study the database contains 2763 different fonts for the English alphabet. To resemble a real life situation, the proposed method is evaluated with printed and scanned text lines and character images. Our evaluation shows that for 99 % of the queries, the correct font name can be found within the five best matches.

Key Words: Font Recognition, Font Retrieval, Eigenimages, Vision Application

1 Introduction

Choosing an appropriate font for a text can be a difficult problem since manual selection is very time consuming. One way to speed up the selection procedure is to be inspired by others. But if we find a text written with a font that we want to use, we have to find out if this font is available in some database. Examples of databases can be the database on our own personal computer, or a database owned by a company selling fonts, or a database with free fonts. In the following we describe our search engine for fonts. The user uploads an image of a text, and the search engine returns the names of the most similar fonts in the database. Using the retrieved images as queries, the search engine can be used for browsing through the database. The basic workflow of the search engine is illustrated in Figure 1. After pre-processing and segmentation of the input image, a local approach is used, where features are calculated for individual characters. In the current version, the user needs to assign letters to segmented input characters that should be included in the search. A similar approach has been utilized by for instance the commercial font search engine WhatTheFont. The proposed method can be applied in any font database. The intended users, however, are mainly people who are selecting fonts in their daily work (graphic designers etc.), where a fast selection procedure can save a lot of time and effort. We emphasize that there are major differences between font recognition in very large font databases, and the font recognition sometimes used as a pre-processor for Optical Character Recognition (OCR). The differences will be explained in upcoming sections.

Correspondence to: <martin.solli@liu.se>

Recommended for acceptance by <João Manuel Tavares>
ELCVIA ISSN:1577-5097
Published by Computer Vision Center / Universitat Autònoma de Barcelona, Barcelona, Spain

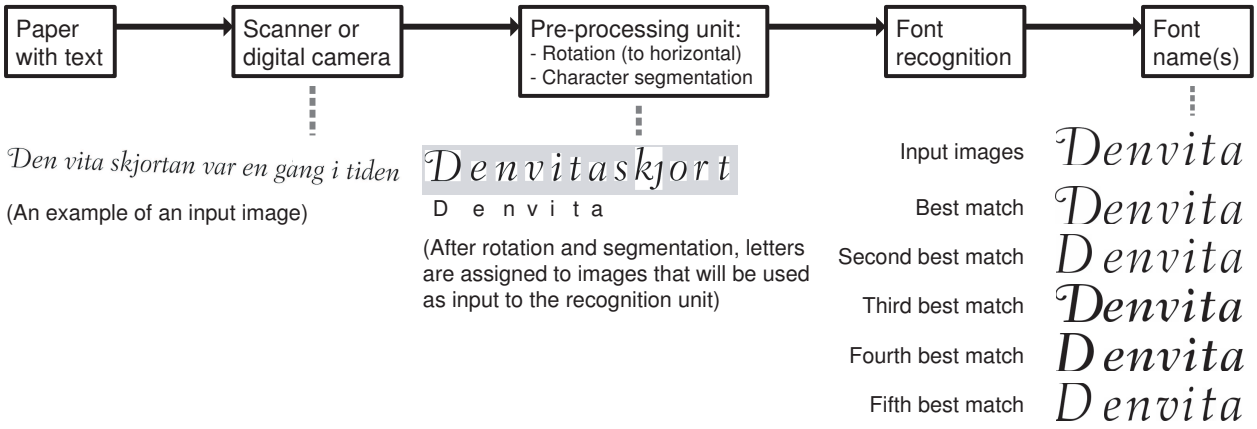


Figure 1: The workflow of the search engine. The input is an image of a text line, typically captured by a scanner or a digital camera. A pre-processing unit will rotate the text line to a horizontal position, and perform character segmentation. Then the user will assign letters to images that will be used as input to the recognition unit, and the recognition unit will display a list of the most similar fonts in the database.

The major contribution of this paper is to show how well known computer vision, or image analysis techniques, can be used in a font recognition application. We will focus our investigations on the recognition task, and leave pre-processing steps (such as character segmentation), and discussions about scalability, clustering, etc., for accompanying or upcoming publications. The main novelty of the presented approach is the use of a very large font database, multiple times larger than databases used in related research. Our solution, which we call Eigenfonts, is based on eigenimages calculated from edge filtered character images. Both the name and the method are inspired by the Eigenfaces method, used in the context of face recognition. Improvements suitable for font recognition, mainly in the pre-processing step, for the ordinary eigen-method are introduced and discussed. Although the implementation and usage of eigenimages is rather simple and straight forward, we will show that the method is highly suitable for font recognition in very large font databases. There are several advantages with the proposed method. It is, for instance, simple to implement, features can be computed rapidly, and descriptors can be saved in compact feature vectors. Other advantages are that the method shows robustness against various noise levels and image quality, and it can handle both overall shape and finer details. Moreover, if the initial training is conducted on a large dataset, new fonts can be added without re-building the system. The findings of our study are implemented in a publicly available search engine for free fonts.*

2 Background

There are two major application areas for font recognition or classification; as a tool for font selection, or as a pre-processor for OCR systems. A major difference between these areas is the typical size of the font database. In an OCR systems it is usually sufficient to distinguish among less than 50 different fonts (or font classes), whereas in a font selection task we can have several hundreds or thousands of fonts, which usually demands a higher recognition accuracy. As mentioned earlier, the database used in this study contains 2763 different fonts. To our knowledge this database is several times larger than any database used in previous work. The evaluation is made with the same database, making it harder to compare the result to other results. However, since the intended usage is in very large databases, we believe it is important to make the evaluation with a database of comparable size.

*http://diameter.itn.liu.se/fyfont/

Figure 2: Examples of character a.

The methods used for font recognition can roughly be divided into two main categories: either local or global feature extraction. The global approach typically extracts features for a text line or a block of text. Different filters, for instance Gabor filters, are commonly used for extracting the features. The local approach sometimes operates on sentences, but more often on words or single characters. This approach can be further partitioned into two sub-categories: known or unknown content. Either we have a priori knowledge about the characters, or we don't know which characters the text is composed of. In this research we utilize a local approach, with known content.

Font recognition research published in international journals and proceedings was for many years dominated by methods focusing on the English or Latin alphabet (with minor contributions focusing on other alphabets, for instance the Arabic alphabet [14], and the South Asian script Sinhala [15]). In recent years, font recognition for Chinese characters has grown rapidly. However, there are major differences between those languages. In the English alphabet we have a rather limited number of characters (basically 26 upper case, and 26 lower case characters), but a huge number of fonts. There is no official counting, but for the English, or closely related alphabets, we can probably find close to 100 000 unique fonts (both commercial and non commercial). The huge number of fonts is likely due the limited number of characters, making it possible to create new fonts within feasible time. For Chinese characters, the number of existing fonts is much fewer, but in the same time, the number of possible characters is much, much larger. The effect is that for Chinese characters, font recognition based on a global approach is often more suitable than a local approach, since with a global approach you don't need to know the exact content of the text. For the English alphabet, one can take advantage of the smaller number of characters and use a local approach, especially if the content of the text is known. However, there is no standard solution for each alphabet. It's probably not totally fair to compare recognition accuracy for Chinese respectively English fonts, but results obtained are rather similar, indicating that none of the alphabets are much easier than the other. Moreover, research concerning completely different alphabets, like the Arabic or Persian alphabet, report similar results. We continue with a summary of past font recognition research. The focus will be on methods working with the English alphabet, but font recognition in other alphabets will be discussed also.

3 Related work

The research presented in this paper originates from a set of experiments presented in Larsson [10], where the goal was to investigate the possibility of using traditional shape descriptors for designing a font search engine. Focus was entirely on the English alphabet, and local feature extraction. Numerous traditional shape descriptors were used, but a majority of the them were eliminated early in the evaluation process. For instance, standalone use of simple shape descriptors, like perimeter, shape signature, bending energy, area, compactness, orientation, etc., were found inappropriate. Among more advanced contour based methods, Fourier descriptors where of highest interest. While Fourier descriptors have been of great use in optical character recognition and object recognition, they are too sensitive to noise to be able to capture finer details in a font silhouette. The same argument can be applied to chain coding. Experiments showed similar drawbacks as with Fourier descriptors. For printed and scanned character images, the noise sensitivity is too high. Another drawback with contour-based methods is that they have difficulties handling characters with more than one contour. In the second part of Larsson [10], the focus was shifted towards region-based methods.

The most promising approach involving region-based methods are originating from the findings of Sexton et al. [17][16], where geometric moments are calculated at different levels of spatial resolution, or for different

image regions. At lower levels of resolution, or in sub-regions, finer details can be captured, and the overall shape can be captured at higher levels of resolution. A common approach is some kind of tree-decomposition, where the image is iteratively decomposed into sub-regions. One can for instance split the image, or region, into four new regions of equal size (a quad-tree). Another approach is to split according to the centre of mass of the shape (known as a kd-tree decomposition), resulting in an equal number of object (character) pixels in each sub-region. In Larsson [10], the best result was found for a four level kd-tree decomposition. Each split decomposes the region into two new regions based on the centroid component. The decomposition alternates between a vertical and horizontal split, resulting in totally 16 sub-regions. For each sub-region, three second order normalized central moments are derived, together with the coordinate of the centroid normalized by the height or width of the sub-region. Also, the aspect ratio of the entire image is added to the feature vector, resulting in a feature vector of length 61. We will in a later section compare the retrieval accuracy of this best performing region-based method with our own approach using eigenimages.

To our knowledge, only two commercial search engines are publicly available for font selection or identification: The oldest and most established is WhatTheFont. The engine is operated by MyFonts[†], a company focusing on selling fonts mainly for the English alphabet. The starting point for WhatTheFont seems to be the local region-based method presented by Sexton et al. [17]. We are not aware of newer, publicly available, descriptions of improvements that are probably included in the commercial system. However, in an article about recognition of mathematical glyphs, Sexton et al. [16] describe and apply their previous work on font recognition. Recently, TypeDNA[‡] introduced various font managing products based on font recognition.

Another example of a local approach for the English alphabet is presented by Östürk et al. [20], describing font clustering and cluster identification in document images. They evaluated four different methods (bitmaps, DCT coefficients, eigencharacters, and Fourier descriptors), and found that they all result in adequate clustering performance, but the eigenfeatures result is the most parsimonious and compact representation. Their goal was not primarily to detect the exact font; instead fonts were classified into clusters. We also mention Hase et al. [7] as example of how eigenspaces can be used for recognizing inclined, rotated characters. Lee and Jung [11] proposed a method using non-negative matrix factorization (NMF) for font classification. They used a hierarchical clustering algorithm and Earth Mover Distance (EMD) as distance metric. Experiments are performed at character-level, and a classification accuracy of 98% was shown for 48 different fonts. They also compare NMF to Principal Component Analysis, with different combinations of EMD and the L_2 distance metric. Their findings show that the EMD metric is more suitable than the L_2-norm, otherwise NMF and PCA produce rather similar results, with a small advantage for NMF. The authors favor NMF since they believe characteristics of fonts are derived from parts of individual characters, compared to the PCA approach where captured characteristics to a larger extent describe the overall shape of the character. The combination of PCA and the L_2 metric has similarities with the approach presented in this paper, but here we apply the method on a much larger database, and use additional pre-filtering of character images. Interestingly, the results from our investigations, on a much larger database, favor the PCA approach ahead of methods using parts of individual characters, like NMF. Another local approach was proposed by Jung et al. [8]. They presented a technique for classifying seven different typefaces with different sizes commonly used in English documents. The classification system uses typographical attributes such as ascenders, descenders and serifs extracted from word images as input to a neural network classifier. Khoubyari and Hull [9] presented a method where clusters of words are generated from document images and then matched to a database of function words from the English alphabet, such as "and", "the" and "to". The intended usage is as a pre-processing step for document recognition algorithms. The method includes both local and global feature extraction. A method with similar approach, focusing on recognizing font families, is proposed by Shi and Pavlidis [18].

We continue with a few methods using a global approach. In Zramdini and Ingold [25], global typographical features are extracted from text images. The method aims at identifying the typeface, weight, slope and size of the text, without knowing the content of the text. Totally eight global features are combined in a Bayesian

[†]http://www.myfonts.com
[‡]http://www.typedna.com

classifier. The features are extracted from classification of connected components, and from various processing of horizontal and vertical projection profiles. They consider the minimum text length to be about ten characters. An early contribution to font recognition is Morris [13], who considered classification of typefaces using spectral signatures. Feature vectors are derived from the Fourier amplitude spectra of images containing a text line. The method aims at automatic typeface identification of OCR data. Among other approaches based on global texture analysis we mention Avilés-Cruz et al. [1] and the early attempt by Baird and Nagy [2].

As mentioned earlier, font recognition for Chinese characters is a rapidly growing research area. An approach using local features for individual characters is presented by Ding et al. [4], where they recognize the font from a single Chinese character, independent of the identity of the character. Wavelet features are extracted from character images, and after a Box-Cox transformation and LDA (Linear Discriminant Analysis) process, discriminating features for font recognition are extracted, and a MQDF (Modified Quadric Distance Function) classifier is employed to recognize the font. Evaluation is made with two databases, containing totally 35 fonts, and for both databases the recognition rates for single characters are above 90%. Another example of Chinese font recognition, including both local and global feature extraction, is described by Ha and Tian [5]. The authors claim that the method can recognize the font of every Chinese character. Gabor features are used for global texture analysis to recognize a pre-dominant font of a text block. The information about the pre-dominant font is then used for font recognition of single characters. Using four different fonts, a recognition accuracy of 99.3% can be achieved. A similar approach can be found in an earlier paper by Miao et al. [12]. In Yang et al. [23], Chinese fonts are recognized based on Empirical Mode Decomposition, or EMD (should not be confused with the distance metric with the same abbreviation). The proposed method is based on the definition of five basic strokes that are common in Chinese characters. These strokes are extracted from normalized text blocks, and so-called stroke feature sequences are calculated. By decomposing them with EMD, Intrinsic Mode Functions (IMFs) are produced. The first two, so-called stroke high frequency energies are combined with the five residuals, called stroke low frequency energies, to create a feature vector. A weighted Euclidean distance is used for searching in a database containing 24 Chinese fonts. Examples of global methods that are evaluated for both Chinese and English fonts are Sun [21], and the method presented by Zhu et al. [24], using Gabor filters. Similar approaches with Gabor filters can be found in Ha et al. [6] and Yang et al. [22].

As an example of Arabic font recognition we refer to Moussa et al. [14]. They present a non-conventional method using fractal geometry on global textures. For nine different fonts, they receive a recognition accuracy of 94.4%.

In conclusion, for Chinese characters (and similar scripts), the recognition is often based on a global approach using texture analysis. For the English alphabet (and similar alphabets), a local approach operating on single characters or words is more common, especially when a rough classification is not accurate enough.

4 Font databases

The font database contains 2763 different fonts for the English alphabet (all fonts are provided by a commercial font distributor, who wants to remain anonymous). Numerous font categories, like basic fonts, script fonts, etc., are included in the database. The originality of each font is purely based on the name of the font. With commercial fonts, however, it is very rare to find exact copies. Single characters are represented by rendered images, typically around 100 pixels high. Figure 2 shows some examples of character 'a' in different fonts. For evaluation, three test databases were created. The first contains images of characters (from the font database) that are printed in 400 dpi with an ordinary office laser printer, and then scanned in at 300 dpi with an ordinary desktop scanner (HP Scanjet 5590, default settings). As a first step 100 randomly selected characters of 'a' and 'b' were scanned, and saved in a database called **testdb1**. These 200 images are used in the initial experiments. For the second database, called **testdb2**, the same 100 fonts were used, but this time all lower case characters were scanned, giving totally 2600 test images. These images are used in the evaluation of the final search engine. A full evaluation with all fonts in the database would require us to print and scan more than 143K characters (counting both upper and lower case). For this initial study we used 100 fonts, which is believed

Name	Number of fonts	Characters	Size of the set (nr of images)	Generated through
original db	2763	all (a-z,A-Z)	143 676	Digital rendering
testdb1	100	a, b	200	Print and scan
testdb2	100	all lower case (a-z)	2600	Print and scan
notindb	7	a	7	Print and scan

Table 1: Overview of databases used in the experiments.

Figure 3: First five eigenimages for character a, reshaped to 2-D images (with normalized intensity values).

to be a reasonable trade-off between evaluation accuracy and the time spend on printing and scanning. The third test database, called **notindb**, also adopts the print and scan procedure, as mentioned above, but with fonts that are not in the original database. Only seven fonts were used, all of them downloaded from dafont (www.dafont.com). Both fonts with an "ordinary look", and fonts with unusual shapes were used. The font names of the downloaded fonts do not exist in our original database, but we can only assume that the appearance of each font differs from the original ones (in worst case, a font designer may have stolen the design from one of the commercial fonts). An overview of all databases can be seen in Table 1.

5 Basic search engine design

This section describes the theoretical background of the proposed search engine, starting with a description of the eigenfonts method, followed by important design parameters, like alignment and edge filtering of character images.

5.1 Eigenfonts basics

We denote by $I(char, k)$ the k^{th} image (font) of character $char$, reshaped to a column vector. Images of characters from different fonts are quite similar in general; therefore images can be described in a lower dimensional subspace. The principal component analysis (or Karhunen-Loeve expansion) reduces the number of dimensions, leaving dimensions with highest variance. Eigenvectors and eigenvalues are computed from the covariance matrix containing all fonts of each character in the original database. The eigenvectors corresponding to the K highest eigenvalues describe a low-dimensional subspace on which the original character images are projected. The coordinates in this subspace are stored as the new descriptors. The first five eigenimages for character 'a' can be seen in Figure 3.

The proposed method works as follows: The 2-D images are reshaped to column vectors, denoted by $I(char, k)$ (where $I(a, 100)$ is the 100^{th} font image of character 'a', as described above). For each character we calculate the mean over all font images in the database

$$m(char) = \frac{1}{N} \sum_{n=1}^{N} I(char, n) \tag{1}$$

where N is the number of fonts in the database. Sets of images will be described by the matrix

$$I(char) = (I(char, 1), ..., I(char, N)) \tag{2}$$

For each character in the database, the corresponding set (usually known as the training set) contains images of all fonts in the database. From each image in the set we subtract the mean and get

$$\widehat{I}(char,k) = I(char,k) - m(char) \tag{3}$$

The covariance matrix C, for character $char$, is then given by

$$C(char) = \frac{1}{N}\sum_{n=1}^{N} \widehat{I}(char,n)\widehat{I}(char,n)' = AA' \tag{4}$$

where $A = [\widehat{I}(char,1), \widehat{I}(char,2), ..., \widehat{I}(char,N)]$. Then the eigenvectors u_k, corresponding to the K largest eigenvalues λ_k, are computed. If it is clear from the context we will omit the $char$-notation. The obtained eigenfonts (eigenimages) are used to classify font images. A new query image, Q (containing character $char$), is transformed into its eigenfont components by

$$\omega_k = u_k'(Q - m) \tag{5}$$

for $k = 1, ..., K$. The weights, $\omega_1, ..., \omega_K$, form a vector that describes the representation of the query image in the eigenfont basis. The vector is later used to find which font in the database describes the query image best. Using the eigenfonts approach requires that all images of a certain character are of the same size and have the same orientation. We also assume that they have the same color or intensity (dark letters on a bright paper background is the most obvious choice). We therefore apply the following pre-processing steps before we compute the eigenfont coefficients: 1) Grey value adjustments: If the character image is a color image, color channels are merged (by adding the color channels in each pixel), and resulting gray values are scaled to fit a pre-defined range ($[0\ 1]$). 2) Orientation and segmentation: If character images are extracted from a text line, the text line is rotated to a horizontal position prior to character segmentation (described in [19]). 3) Scaling: Character images are scaled to a fixed size.

5.2 Character alignment and edge filtering

In the design process, the first thing to consider is the character alignment. Since we are using the eigenfonts method, images must have the same size, but the location of the character within each image can vary. We consider two choices: each character is scaled to fit the image frame exactly, or the original aspect ratio of each character is maintained, but characters are aligned according to their centroid value, leaving space at image borders. The later requires larger eigenfont images since the centroid value varies between characters, which will increase the computational cost. Experiments showed that frame alignment gives significantly better retrieval accuracy than centroid alignment.

Most of the information about the shape of a character can be found in the contour, especially in this case when shapes are described by black text on a bright background. Exceptions are noise due to printing and scanning, and gray values in the contour due to anti-aliasing effects when images are rendered. Based on this assumption, character images were filtered with different edge filters before calculating the eigenimages. The images used are rather small and therefore we use only small filter kernels (max 3×3 pixels). Experiments with many different filter kernels resulted in the following filters used in the final experiments (four diagonal filters, one horizontal, and one vertical edge filter):

$$H = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix},\ D_1 = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{pmatrix},\ D_3 = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$V = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix},\ D_2 = \begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix},\ D_4 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

Image size	Filter	PM	T5
40×40	H	73	95
40×40	V	57	83
40×40	D1	69	97
40×40	D2	66	94
40×40	D3	66	86
40×40	D4	56	87
40×40	H+D1+D2	77	97
40×40	H+D1	75	94
40×40	H+D2	63	96
40×40	H+V	73	98
25×25	H+D1+D2	82	98
20×20	H+D1+D2	81	98
15×15	H+D1+D2	80	96
10×10	H+D1+D2	53	78

Table 2: Retrieval accuracy for different filters and filter combinations. (Character 'a' from testdb1)

The mean retrieval result for character 'a', filtered with different filters can be seen in Table 2. The result in the column marked PM corresponds to the percentage of when the correct font is returned as the best match (Perfect Match), and the T5 column when the correct font can be found within the five best matches (Top 5). In other words, a figure of for instance 80 in the T5 column, means that the correct font can be found within the five best matches for 80% of the query fonts. The same notation will be used in the rest of this paper. Since the intended application is a dedicated search engine, we believe it is feasible to assume that the user finds it acceptable to examine at least five matches. When several filters are used, the character image is filtered with each filter separately, and then filter results are added to create the final result. The table shows that the combination of one horizontal and two diagonal filters gives the best result. Some of the experiments with varying image sizes are listed in the same table, showing that images of size 25×25 pixels seem to be a good choice. Reducing the image size without loosing retrieval accuracy is beneficial since the computational load will decrease. Sizes below 15×15 pixels decreased the retrieval accuracy significantly.

To verify the result from character 'a', a second test was carried out with characters 'd', 'j', 'l', 'o', 'q' and 's', from testdb2. The result for different combinations of filters can be seen in Table 3. The retrieval results vary slightly between different characters, but usually filter combinations "H+D1+D2" and "H+D1" perform well. The first combination, a horizontal Sobel filter together with two diagonal filters, is selected for the remaining experiments. The vertical filter does not improve the result, probably because characters from different fonts often contain very similar vertical lines.

5.3 Selection of eigenimages

The selection of the number of eigenvectors, here called eigenimages, has a strong influence on the search performance. The number of selected eigenvectors is a tradeoff between accuracy and processing time. However, increasing the number of eigenimages used leads first to an increased performance, but the contribution of eigenimages corresponding to low eigenvalues is usually negligible. The number of eigenimages compared to the mean retrieval accuracy for scanned and printed versions of character 'a', and the mean accuracy over all small characters, 'a-z', can be seen in Figure 4. Image size is 24×24 pixels, and images are pre-processed with edge filtering. The figure shows that 30 to 40 eigenimages are appropriate. Similar tests with other image sizes, and other individual characters, confirm that 40 eigenimages are sufficient.

A question that arises is wether all 40 eigenimages are needed if we only want to perform classification? For

	d		j		l		o		q		s	
Filter	PM	T5	PM	T5	PM	T5	PM	T5	PM	T5	PM	T5
H	88	100	**86**	**99**	72	82	79	97	91	100	**92**	**100**
V	86	98	70	94	58	78	81	99	87	99	**91**	**100**
D1	**90**	**100**	82	98	64	85	81	97	91	100	**92**	**100**
D2	**91**	**100**	80	98	66	84	**84**	**99**	92	100	**91**	**100**
H+V	89	100	82	98	68	85	**85**	**99**	**95**	**100**	**91**	**100**
H+V+D1+D2	88	100	80	99	66	85	**82**	**98**	**93**	**100**	**91**	**100**
H+D1+D2	**90**	**100**	**85**	**98**	**72**	**88**	**83**	**98**	**93**	**100**	**91**	**100**
V+D1+D2	88	99	79	94	59	82	**84**	**98**	89	100	**91**	**100**
D1+D2	89	100	79	97	65	84	**85**	**98**	**93**	**100**	**92**	**100**
H+D1	89	100	**86**	**99**	**75**	**89**	80	97	**93**	**100**	**91**	**100**
H+D2	**90**	**100**	**85**	**99**	**72**	**88**	83	97	91	100	90	100

Table 3: Retrieval accuracy for different filter combinations, for character 'd', 'j', 'l', 'o', 'q' and 's'. The best results are printed in **bold**. (Characters from testdb2)

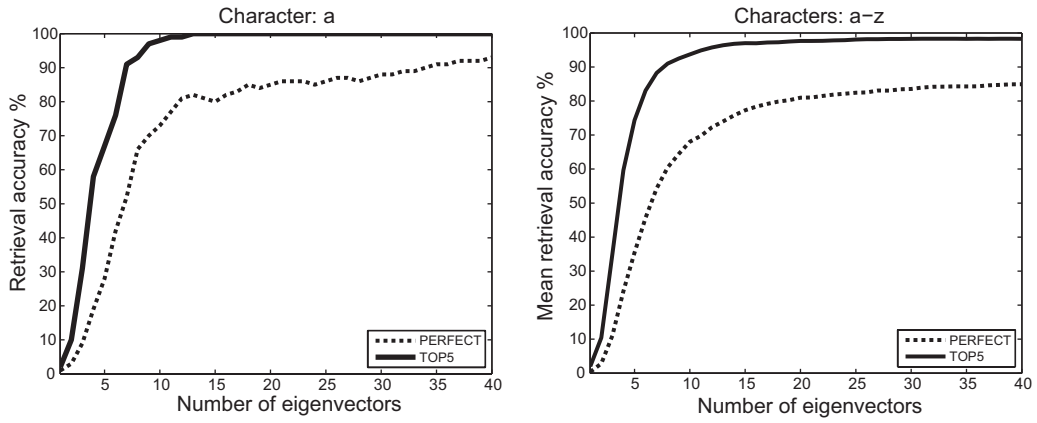
instance perform classification into different font styles, like Regular, **Bold** and *Italic*. Preliminary experiments indicate that classification or clustering of font styles can be achieved with only a few of the first eigenimages, corresponding to high eigenvalues. Eigenimages belonging to low eigenvalues are more or less useless for classification, but as can be seen in Figure 4, they are needed to obtain high accuracy in the recognition. They are probably more useful for distinguishing finer details and small variations. A visualization of some of the experimental results can be seen in Figure 5. In both sub-figures, character 'a' from the first 200 fonts in the database are plotted in a 2-D coordinate systems. The position of each character image is determined from the projection of the character image onto a subspace spanned by two eigenvectors. In Figure 5 (a), the coordinates are obtained from the second and third eigenimages (u_2 and u_3), both corresponding to high eigenvalues. Different font styles are clearly separated into different clusters, or regions, in this 2-D space. The eigenimage u_1 belonging to the highest eigenvalue, corresponds to a rough measure of the area of the character ("boldness"), and can also be a useful parameter in a classification task. In Figure 5 (b), the coordinates are obtained from eigenimages corresponding to the two lowest eigenvalues (u_{39} and u_{40}). Here the characters seem to be almost randomly distributed. Since our goal is recognition, we do not investigate the classification properties further, but for researchers working with OCR systems (where classification can be used as a pre-processor) it might be an interesting topic for further research.

6 Choosing search engine components

From the previous section, discussing basic search engine design, we continue with components for fine tuning the system, such as different techniques for image scaling and interpolation, the measure of similarity, etc.

Image scaling and interpolation: An initial idea was to use different image sizes for different characters. For instance character 'i' may benefit from using a rectangular image size, etc. Experimental results showed that by optimizing the image size for different characters, the overall retrieval accuracy will increase. We will, however, use a fixed image size (24×24 pixels) for all possible characters, since this removes the necessity to choose the window size which makes the method more general. Even with a fixed image size, we need to perform scaling, and scaling requires interpolation. The influence of the interpolation method on the search performance was evaluated for three common interpolation techniques: nearest neighbor, bilinear and bicubic interpolation. The experiments showed that the interpolation method is of minor importance as long as something more advanced than nearest neighbor is used.

Extra features: Eigenfonts alone results in high retrieval accuracy, but we also investigate if features not



(a) (b)

Figure 4: The retrieval performance for different number of eigenimages. The result for character 'a' from testdb2 are shown in (a), and the mean accuracy over all characters 'a-z' in testdb2 are shown in (b). The dashed line corresponds to a perfect match, and the solid line corresponds to a top 5 result.

Image size	Extra feature	PM	T5
24×24	none	70	97
24×24	Area/Box	71	98
24×24	Height/Width	80	99
24×24	Centroid	70	97
24×24	All three above	80	99

Table 4: Retrieval accuracy for features not derived from eigenimages (for character 'a' from testdb1).

derived from eigenimages can improve the performance. Three simple features are investigated: 1) The ratio between character height and width before scaling, 2) the ratio between the area of the character and the area of the surrounding box, and 3) center of gravity (centroid) values. The result is shown in Table 4. The only extra feature that resulted in significant improvements is the ratio between character height and width. However, in the final implementation, it is important that the value is weighted properly. In the final application, we use a weight value of 8, which is slightly higher than the maximum value usually obtained from the eigenfonts representation (in our implementation, coordinates in the eigenfont space usually lie within the range $[-5\ 5]$). Combinations of different extra features did not produce higher accuracy compared to the case when only the ratio between character height and width is used.

Measuring similarity: The similarity between feature vectors is calculated with the L_2 norm, or Euclidean distance, given by

$$d(x,y) = (\sum_{i=1}^{n} |x_i - y_i|^2)^{\frac{1}{2}} \tag{6}$$

where x and y are vectors of length n. Other distance measures were tested (L_1, and Mahalanobis distance with different covariance matrices), but the Euclidean distance gave the best result. This might be related to the fact that eigenimages are calculated with Principal Component Analysis, which is defined as the minimizer of the L_2 approximation error.

(a) (b)

Figure 5: Character 'a' from the first 200 fonts in the database plotted in the space spanned by (a) the eigenimages u_2 and u_3 corresponding to high eigenvalues, and (b) the eigenimages corresponding to the two lowest eigenvalues (u_{39} and u_{40}).

7 Search engine performance

In this section the overall results are presented together with experiments investigating the effects of different pre-processing methods and noise sensitivity. The current version of the search engine adopts a sequential search, where each query character is compared to all descriptors in the database belonging to that character. With the proposed descriptor, and the current size of the database, there is no need (in practice) to implement methods for maintaining scalability, such as clustering methods. The search time (for comparing and sorting descriptor vectors) for a single character, on an ordinary laptop with a 2.2 GHz processor, is 1.3 milliseconds, which is believed to be acceptable. Only if we try to include all available fonts (there are approximately close to 100 000 unique fonts i world), we might consider using methods for maintaining scalability. Since the presented recognition method is based on well known and frequently used standard tools, it should be possible to implement the search engine using almost any software.

7.1 Method composition

Below is an overview of the final combination of different methods and settings, used for both the training set (character images from the original font database) and query images:

* **Image size:** Square images, size 24×24 pixels. Characters are aligned and scaled to fill the whole image. Bilinear interpolation is used for scaling.

* **Edge filtering:** Three Sobel filters, one horizontal and two diagonal.

* **Number of eigenimages:** 40

* **Extra features:** Ratio between character height and width before scaling.

* **Distance metric:** L_2 norm (Euclidean distance)

Character	PM	T5	Character	PM	T5	Character	PM	T5	Character	PM	T5
a	94	100	h	88	100	o	83	98	v	89	99
b	89	100	i	81	94	p	88	100	w	91	99
c	89	99	j	85	99	q	93	100	x	90	100
d	89	100	k	86	100	r	87	100	y	87	100
e	91	100	l	70	88	s	91	100	z	90	100
f	90	100	m	91	100	t	87	99			
g	88	100	n	91	100	u	91	100			
									MEAN	88.0	99.0

Table 5: Search performance for different characters (from testdb2) for the proposed method using eigenfonts.

Character	PM	T5	Character	PM	T5	Character	PM	T5	Character	PM	T5
a	89	100	h	89	99	o	88	100	v	93	100
b	89	98	i	75	94	p	89	100	w	89	100
c	87	100	j	81	96	q	86	98	x	90	100
d	91	100	k	87	99	r	90	100	y	85	98
e	91	100	l	70	92	s	86	97	z	82	96
f	89	97	m	91	100	t	86	100			
g	91	100	n	88	100	u	87	100			
									MEAN	86.9	98.6

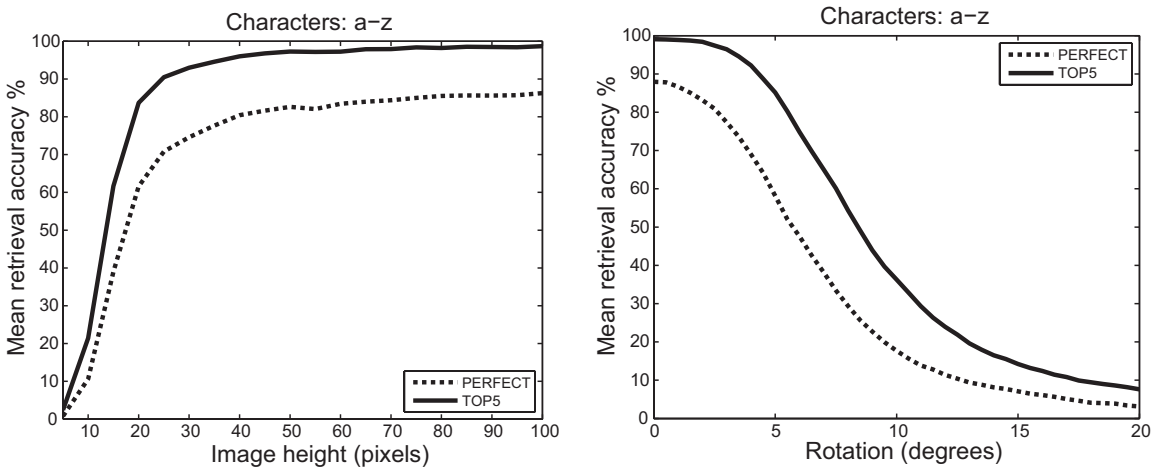
Table 6: Search performance for different characters (from testdb2) for the best performing region-based local method in Larsson [10].

In the final search engine, pre-processing is necessary before shape recognition. First the text line is rotated to a horizontal position (if needed). The rotation compensation is based on the Hough transform of an edge filtered text line image. Then the text line is segmented into single characters. The segmentation procedure is described in [19], but not given here since the focus of this paper is on the recognition task. However, a general survey of methods and strategies in character segmentation, also known as dissection, can be found in [3].

7.2 Overall results

The final search engine is evaluated with testdb2, containing 100 different fonts with 26 characters in each. In total 2600 characters, first printed and then scanned to resemble a real life situation. The search performance is measured as the mean performance of single character queries. For different characters the results are shown in Table 5. The mean values, over all characters, for a perfect match and a top 5 result, is 88.0% and 99.0% respectively. The mean values are significantly decreased due to the poor result for characters 'l' and 'i'. The reason is obvious; those characters contain relatively few lines and details that can be used for distinguishing fonts from each other. Without 'l' and 'i', the mean values increase to 89.1% and 99.7%. Since the input to the search engine is a text line, tricky characters like 'l' can be removed or weighted down, and the remaining characters will be sufficient for producing the result.

For comparison, we evaluate the best performing region-based method from Larsson [10], described in Section 3, on exactly the same test database. The result can be seen in Table 6. We notice that the proposed method using eigenfonts performs better than the region-based method. Another advantage with the eigenfonts method is that the length of the feature vector is about $2/3$ of the length obtained from the region-based method.



(a) (b)

Figure 6: The relationship between (a) query image size and search performance, and (b) query image rotation and search performance. The dashed line corresponds to a perfect match, and the solid line corresponds to a top 5 result.

7.3 Image quality influence

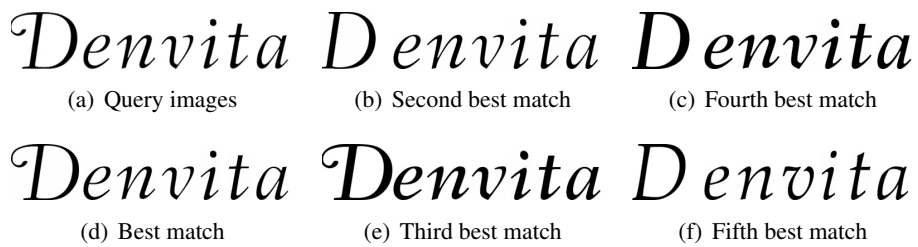
Experiments based on image resolution/scaling, JPEG compression and character rotation are presented in this section. Shown results are the mean performance over all characters 'a-z' in testdb2. Figure 6 (a) shows the relationship between query image size and search performance. To obtain a satisfying retrieval result, query images need to be at least 40-50 pixels in height. For images below 20 pixels, the performance decreases rapidly. Next, the correlation between different JPEG compression rates and search performance is examined. In JPEG compression, the quality can be set between 0 and 100, where 100 corresponds to the best quality (lowest compression). Results obtained indicate that the compression rate is of minor importance. Only when the quality is below 50 the retrieval performance is affected, but only slightly. Finally, the relationship between query image orientation (rotation) and search performance can be seen in Figure 6 (b). Here query images are rotated counter-clockwise. When characters are rotated more than 3-4 degrees, the performance declines sharply. However, an angle around or over 3-4 degrees is rarely encountered in real samples. After pre-processing, rotation angles are usually below 1 degree. We mention that recognizing rotated characters is always difficult since an italic font, rotated a few degrees counter-clockwise, can be mistaken for a regular font, and vice versa.

7.4 An example of a complete search

The complete process from input image to the font name output is illustrated. An example of an input image can be seen in the top left part of Figure 7. The first step is to ensure that the text line is horizontal. This is done using the Hough transform on an edge filtered image, and the result can be seen in the bottom left part of Figure 7.

Then the text line is segmented into sub images, each corresponding to one character. The first 12 sub images are shown in the right side of Figure 7. The user will assign letters to sub images that are to be used in the search. Since the segmentation algorithm did not manage to segment 'k' and 'j', the user should not assign a letter to this sub image. The character segmentation can be improved, but here we are primarily interested in font recognition and therefore we did not implement a more sophisticated segmentation method. Assigned images will be used as input to the search engine. Results from individual characters are weighted and combined

Figure 7: Top left: Example of an input image. Bottom left: Input image after rotation. Right: 12 first sub images after segmentation.



(a) Query images (b) Second best match (c) Fourth best match

(d) Best match (e) Third best match (f) Fifth best match

Figure 8: (a) Query images (b)-(f) The five most similar fonts in the database.

to a final result, presenting the most similar fonts in the database. In this initial study, we found it appropriate to weight the result from each character based on the squared retrieval accuracy obtained in Table 5 ($(PM/100)^2$). In other words, characters with a high individual accuracy will have a strong influence on the overall result, and vice versa. We then combine the result by letting each query character vote on five fonts in the database. The best match is given a vote value of 5 times the weight for that particular character. The second best match is given a value of 4 times the weight, etc. The font that obtained the overall highest vote value is considered the best match. Although the combination of different characters is very important for the final result, we leave this discussion for upcoming papers (see Future work). Figure 8 shows the five most similar fonts for the query image. In this case the first seven characters were assigned letters and used as input. Even when the correct font cannot be found in the database a match list will be presented to the user, showing the most similar fonts. The user can then continue the search by using a font from the match list as a new query. To see more examples, we encourage readers to visit our public implementation of the search engine.[§]

7.5 Fonts not in the database

It is always difficult to know how the search engine will respond when query images are created with fonts that are not in the database. The retrieval accuracy cannot be measured in the same way as in previous sections. We can only examine the results visually. Figure 9 shows seven queries with character 'a' from fonts belonging to the database notindb, together with the five best matches. The problem is that for some query images it is feasible to believe that we can not find any similar fonts at all in the database. For such situations, one can implement a reject policy (based on the similarity score) that will tell the user that no similar fonts were found. However, showing a match list is still beneficial, since a font from the match list can inspire a new query.

8 Conclusions

A search engine for very large font databases has been presented and evaluated. The input is an image with a text from which the search engine retrieves the name of the font used to render the text. The method is based on eigenimages calculated for different characters and fonts. Prior to calculating eigenimages, character images are scaled and filtered with one horizontal and two diagonal edge filters. Even for a very large font database,

[§]http://diameter.itn.liu.se/fyfont/

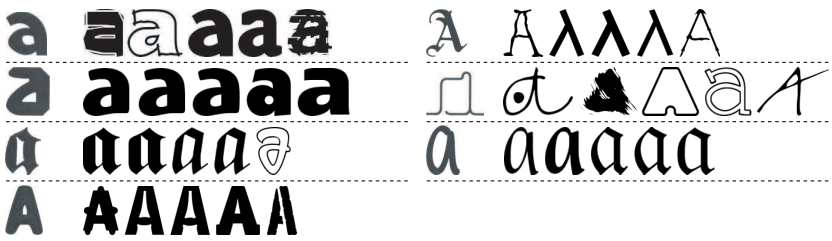


Figure 9: Search result for character 'a' from seven fonts not present in the database. The search image to the left, followed by the five best matches.

containing 2763 fonts, the retrieval accuracy is very high. For individual characters, the mean accuracy for a perfect match is 88.0%, and the probability to find the correct font name within the five best matches is 99.0%. These values are obtained for single character recognition. In practice, the overall accuracy will increase since the search engine will work with text lines, giving the opportunity to combine the result from many characters. To resemble a real life situation, retrieval experiments were made with printed and scanned text lines and character images from the original database. The method has proven to be robust against noise, like JPEG compression, and rotation/scaling of the input image. Since the database in this work is several times larger than what others have used, comparing retrieval accuracy is not straightforward. However, the proposed approach performs better than the best performing region-based local method presented in earlier research.

9 Future work

Performance can probably be improved by combining single character matchings. The main open problem here is how the matching results from different characters are to be weighted. This may not only depend on the search performance for different characters (see section 7.2), or the frequency of different characters in different languages, instead we believe it is even more important to consider what characters users prefer to search with, or what characters the users find important for the visual interpretation of the search result. We investigated the retrieval results for three extra features, but emphasize that additional features can be investigated in future implementations. Examples are features based on graphemes, or features linked to special signs belonging to different alphabets. In current version of the search engine we assume that the user assigns a correct letter to images that will be used as input. One way to improve the system would be to use character recognition for helping the user assigning images. However, that would be an overwhelming task if collected query images show that users mainly submit very strange looking fonts.

References

[1] C. Avilés-Cruz, R. Rangel-Kuoppa, M. Reyes-Ayala, A. Andrade-Gonzalez, and R. Escarela-Perez. High-order statistical texture analysis - font recognition applied. *Pattern Recogn. Lett.*, 26(2):135–145, 2005.

[2] H. S. Baird and G. Nagy. Self-correcting 100-font classifier. *Document Recognition*, 2181:106–115, 1994.

[3] R.G. Casey and E. Lecolinet. A survey of methods and strategies in character segmentation. *IEEE Trans Pattern Anal Mach Intell*, 18(7):690–706, 1996.

[4] X. Ding, L. Chen, and T. Wu. Character independent font recognition on a single chinese character. *IEEE Trans Pattern Anal Mach Intell*, 29(2):195–204, 2007.

[5] M.-H. Ha and X.-D. Tian. Optical font recognition of chinese characters based on texture features. In *4th Int Conf on Machi Learn and Cybern, ICMLC 2005*, 3930:1005–1013, Guangzhou, 2005.

[6] M.-H. Ha, X.-D. Tian, and Z.-R. Zhang. Optical font recognition based on gabor filter. In *Int Conf on Machi Learnand Cybern, ICMLC 2005*, pages 4864–4869, Guangzhou, 2005.

[7] H. Hase, T. Shinokawa, M. Yoneda, and C.Y. Suen. Recognition of Rotated Characters by Eigen-space. In *ICDAR '03, Proc of the 7th Int Conf on Document Analysis and Recognition*, 2003.

[8] M. Jung, Y. Shin, and S. Srihari. Multifont classification using typographical attributes. In *ICDAR '99: Proc of the 5th Int Conf on Document Analysis and Recognition*, 1999.

[9] S. Khoubyari and J.J. Hull. Font and function word identification in document recognition. *Comput Vision Image Undersanding*, 63(1):66–74, 1996.

[10] S. Larsson. Font search engine. Master's thesis, Linköping University, Dept of Sci and Techn, 2006.

[11] C.W. Lee, H. Kang, H.J. Kim, and K. Jung. Font classification using nmf with hierarchical clustering. *Int J Pattern Recognit Artif Intell*, 19(6):755–773, 2005.

[12] X.-F. Miao, X.-D. Tian, and B.-L. Guo. Individual character font recognition based on guidance font. In *Proc. of 2002 Int Conf on Machine Learning and Cybernetics*, 4:1715–1717, Beijing, 2002.

[13] R.A. Morris. Classification of digital typefaces using spectral signatures. *Patt. Rec.*, 25(8):869–876, 1992.

[14] S.B. Moussa, A. Zahour, M.A. Alimi, and A. Benabdelhafid. Can fractal dimension be used in font classification. In *8th Int Conf on Document Analysis and Recognition*, 2005:146–150, Seoul, 2005.

[15] H.L. Premaratne, E. Jrpe, and J. Bigun. Lexicon and hidden markov model-based optimisation of the recognised sinhala script. *Pattern Recogn. Lett.*, 27(6):696–705, 2006.

[16] A. Sexton and V. Sorge. Database-driven mathematical character recognition. In *6th Int Workshop on Graphics Recognition, GREC 2005*, LNCS 3926:218–230, Hong Kong, 2006.

[17] A. Sexton, A. Todman, and K. Woodward. Font recognition using shape-based quad-tree and kd-tree decomposition. *Proc. of the 5th Joint Conf on Inform Sci, JCIS 2000*, 5:212–215, Atlantic City, 2000.

[18] H. Shi and T. Pavlidis. Font recognition and contextual processing for more accurate text recognition. In *Proc. of the 1997 4th Int Conf on Document Analysis and Recogn, ICDAR'97*. 1:39–44, Ulm, 1997.

[19] M. Solli. Topics in content based image retrieval - fonts and color emotions. Licentiate thesis No. 1397, Linköping University, 2009. *Available at: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-16941*

[20] S. Östürk, B. Sankur, and A.T. Abak. Font clustering and cluster identification in document images. *J Electron Imaging*, 10(2):418–430, 2001.

[21] H.-M. Sun. Multi-linguistic optical font recognition using stroke templates. In *18th Int Conf on Pattern Recognition, ICPR 2006*, 2:889–892, Hong Kong, 2006.

[22] F. Yang, X.-D. Tian, and B.-L. Guo. An improved font recognition method based on texture analysis. In *Proc. of 2002 Int Conf on Machine Learning and Cybernetics*, 4:1726–1729, Beijing, 2002.

[23] Z. Yang, L. Yang, D. Qi, and C.Y. Suen. An emd-based recognition method for chinese fonts and styles. *Pattern Recogn. Lett.*, 27(14):1692–1701, 2006.

[24] Y. Zhu, T. Tan, and Y. Wang. Font recognition based on global texture analysis. *IEEE Trans Pattern Anal Mach Intell*, 23(10):1192–1200, 2001.

[25] A. Zramdini and R. Ingold. Optical font recognition using typographical features. *IEEE Trans Pattern Anal Mach Intell*, 20(8):877–882, 1998.