FyFont: Find-your-Font in Large Font Databases

Martin Solli and Reiner Lenz

Dept. Sci & Tech., Linköping University, 601 74 Norrköping, Sweden
marso@itn.liu.se reile@itn.liu.se

Abstract. A search engine for font recognition in very large font databases is presented and evaluated. The search engine analyzes an image of a text line, and responds with the name of the font used when writing the text. After segmenting the input image into single characters, the recognition is mainly based on eigenimages calculated from edge filtered character images. We evaluate the system with printed and scanned text lines and character images. The database used contains 2763 different fonts from the English alphabet. Our evaluation shows that for 99.8 % of the queries, the correct font name is one of the five best matches. Apart from finding fonts in large databases, the search engine can also be used as a pre-processor for Optical Character Recognition.

1 Introduction

When selecting a font for a text one often has an idea about the desired appearance of the font, but the font name is unknown. Sometimes we may have a text written in a similar but unknown font. In that case we would like to find out if that font is contained in a given database. Examples of databases can be the collection on our own personal computer, a commercial database belonging to a company, or a set with free fonts. In this paper we describe a new font recognition approach called Eigenfonts, based on eigenvectors and eigenvalues of images. The approach is closely related to Eigenfaces, used for face recognition, see Turk and Pentland [1]. We also introduce some improvements, mainly in the pre-processing step. In our experiments we use three different databases. First the original database containing character images created directly from font files. A few examples of images from this collection can be seen in Fig. 1. The second database contains images from the original database that are printed and scanned. The third contains images from unknown fonts, also printed and scanned. Database two and three are used for evaluation. In the following chapters when we refer to 'testdb1' or 'testdb2', we use collections of images from the second database. More about the font databases can be found in section 4.2. The search engine can also be utilized as a pre-processor for Optical Character Recognition, or as a font browser if retrieved images are used as queries.

The paper is organized as follows: In the next chapter some previous attempts in font recognition are described. In chapter 3 we present the design of the search engine FyFont (**F**ind-**y**our-**Font**), a publicly available search engine for free fonts[1]. This includes a description of the Eigenfonts method together

[1] http://media-vibrance.itn.liu.se/fyfont/

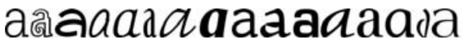


Fig. 1. Examples of character a

with the design parameters. In chapter 4, we summarize the search engine and evaluate the overall performance. This will also include a description of the font databases, and the dependence on properties of the search image. Then we draw conclusions in chapter 5.

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. A difference between these areas is the typical size of the font database. In a font selection task, we can have several hundreds or thousands of fonts, whereas in OCR systems it is usually sufficient to distinguish between less than 50 different fonts.

To our knowledge, only one search engine is available for font selection: WhatTheFont. The engine is commercially operated by MyFonts.com. An early description of the method can be found in [2]. Fonts are identified by comparing features obtained from a hierarchical abstraction of binary character images at different resolutions. Their database consisted of 1300 characters from 50 different fonts rendered at 100 pts/72 dpi. The best performance was achieved with a non-weighted kd-tree metric at 91% accuracy for a perfect hit. We are not aware of newer, publicly available, descriptions of improvements that are probably included in the commercial system.

Others used font classification as a pre-processor for OCR systems. There are mainly two approaches, a local approach based on features for individual characters or words, and a global approach using features for blocks of text. An example of the local approach describing font clustering and cluster identification in document images can be found in [3]. Four different methods (bitmaps, DCT coefficients, eigencharacters, and Fourier descriptors) are evaluated on 65 fonts. All result in adequate clustering performance, but the eigenfeatures result in the most compact representation. Another contribution to font recognition is [4], considering classification of typefaces using spectral signatures. A classifier capable of recognizing 100 typefaces is described in [5], resulting in significant improvements in OCR-systems. In [6] font attributes such as "serifness" and "boldness" where estimated in an OCR-system.

A technique using typographical attributes such as ascenders, descenders and serifs from word images is presented in [7]. These attributes are used as input to a neural network classifier for classifying seven different typefaces with different sizes commonly used in English documents. Non-negative matrix factorization (NMF) for font classification was proposed in [8]. 48 fonts were used in a hierarchical clustering algorithm with the Earth Mover Distance (EMD) as distance metric. In [9] clusters of words are generated from document images and then matched to a database of function words such as "and", "the" and "to". Font

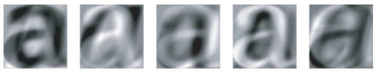


Fig. 2. First five eigenimages for character a (with normalized intensity values)

families are recognized in [10] through global page properties such as histograms and stroke slopes together with information from graph matching results of recognized short words such as "a", "it" and "of".

In a global approach [11], text blocks are considered as images containing specific textures, and Gabor filters are used for texture recognition. A recognition rate above 99% is achieved for 24 frequently used Chinese fonts and 32 frequently used English fonts. The authors conclude that their method is able to identify more global font attributes, such as weight and slope, but less appropriate to distinguish finer typographical attributes. Similar approaches with Gabor filters can be found in [12] and [13]. Also [14] describes an approach based on global texture analysis. Features are extracted using third and fourth order moments. 32 commonly used fonts in Spanish texts are investigated in the experiments.

Here we describe a new local approach based on the processing of character images.

3 Search Engine Design

In this chapter we introduce the eigenfonts method. We discuss important design parameters, like alignment and edge filtering of character images, and end with fine tuning the parameters of the system.

3.1 Eigenfonts Basics

We denote by $I(char, k)$ the k^{th} gray value image (font) of character $char$. For instance, $I(a, 100)$ is the 100^{th} font image of character 'a'. 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 of each character in the original database. The eigenvectors corresponding to the K highest eigenvalues describe a low-dimensional subspace on which the original images are projected. The coordinates in this low-dimensional space are stored as the new descriptors. The first five eigenimages for character 'a' can be seen in Fig. 2.

The method works as follows. Using the given images $I(char, k)$, for each character we calculate the mean image from different font images

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

After reshaping images to one column vectors, sets of images will be described by the matrix $I(char) = (I(char, 1), ..., I(char, N))$. From each set we subtract the mean image and get $\widehat{I}(char) = I(char) - m(char)$. The covariance matrix is then given by $C(char) = \widehat{I}(char)\widehat{I}(char)'/N$, and the eigenvectors u_k, corresponding to the K largest eigenvalues λ_k, are computed. The corresponding coefficients are used to describe the image. A query image is reshaped to a vector Q, and for $k = 1, ..., K$ its weights $\omega_1, ..., \omega_K$ are computed as: $\omega_k = u'_k(Q - m)$. The weights form a vector that describes the representation of the query image in the eigenfont basis. 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 (black letters on a white 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, and then gray values are scaled to fit a pre-defined range. 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. 3) Scaling: Character images are scaled to the same size.

3.2 Character Alignment and Edge Filtering

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: characters are scaled to fit the image frame exactly, or 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 we are dealing with black text on a white paper background. Based on this assumption, character images were filtered with different edge filters before calculating eigenimages. The images used are rather small and therefore we use only small filter kernels (max 3×3 pixels). Extensive experiments with different filter kernels (reported elsewhere) resulted in the following filters used in the final experiments (in Matlab notation): H=[1 2 1;0 0 0;-1 -2 -1]; V=[1 0 -1;2 0 -2;1 0 -1]; D1=[2 1 0;1 0 -1;0 -1 -2]; D2=[0 1 2;-1 0 1;-2 -1 0]; D3=[-1 0;0 1]; D4=[0 -1;1 0]; Four diagonal filters, one horizontal and one vertical edge filter. Retrieval results for character 'a', filtered with different filters can be seen in Table 1. PERFECT corresponds to the percentage of when the correct font is returned as the best match, and TOP5 when the correct font can be found within the five best matches. The same notation will be used in the rest of this paper. When several filters were used, we filtered the image with each filter separately, and then added the resulting images.

The table shows that the combination of one horizontal and two diagonal filters gives the best result. Some experiments with varying image sizes showed that images of size 25×25 pixels seem to be a good choice. Sizes below 15×15

Table 1. Retrieval accuracy for different filter combinations. (Character 'a' from testdb1, image size 40 × 40 pixels).

Filter	PERFECT	TOP5	Filter comb.	PERFECT	TOP5
H	73	95	H+D1+D2	**77**	**97**
V	57	83	H+D1	75	94
D1	69	97	H+D2	63	96
D2	66	94	H+V	73	98
D3	66	86			
D4	56	87			

Table 2. Retrieval accuracy for different filter combinations, for character 'd', 'j', 'l', 'o', 'q' and 's' from testdb2. (P=PERFECT, T5=TOP5).

	d		j		l		o		q		s	
Filter	P	T5	P	T5	P	T5	P	T5	P	T5	P	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

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 2. The retrieval results vary slightly between different characters, but usually filter combinations "H+D1+D2" and "H+D1" perform well. We choose the first combination, a horizontal Sobel filter together with two diagonal filters. The vertical filter does not improve the result, probably because many characters contain almost the same vertical lines.

3.3 Selection of Eigenimages

The selection of eigenimages is a tradeoff between accuracy and processing time and is critical for search performance. Increasing the number of eigenimages used leads first to an increased performance but the contribution of eigenimages with low eigenvalues is negligible. The number of eigenimages and retrieval performance for scanned and printed versions of character 'a' (size 24×24 pixels, edge filtered) can be seen in Fig. 3. The figure shows that 30 to 40 eigenimages are

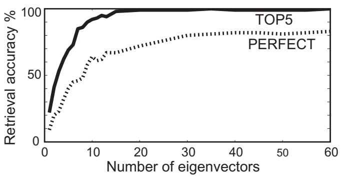


Fig. 3. Retrieval performance for different number of eigenvectors (eigenimages)

appropriate for character ’a’. Preliminary tests were carried out with other image sizes, and other characters, and most of them show that using 40 eigenimages is sufficient.

3.4 Additional Search Engine Components

We continue by analyzing the influence of image size and interpolation methods. As mentioned in chapter 3.2, a suitable image size for character ’a’ is 24×24 pixels. However, quadratic images is not suitable for characters like ’l’ and ’i’, therefore we use rectangular eigenimages for ”rectangular characters”. As an example, the perfect match accuracy for character ’b’ (considered as a ”rectangular character”) increased from 77 to 81 % when the image size changed from 24×24 to 24×20 pixels (rows$\times$columns). Scaling requires interpolation, and the influence of the interpolation method on the search performance was evaluated for three common interpolation techniques: Nearest neighbor, bilinear and bicubic interpolation. The result showed that the interpolation method is of minor importance as long as something more advanced than nearest neighbor is used.

We also evaluated if features not derived from eigenimages could improve retrieval performance. We evaluated the influence of the ratio between character height and width (before scaling), the ratio between the area of the character and the area of the surrounding box, and center of gravity (centroid) values. The only extra feature that resulted in significant improvements is the ratio between height and width. However, it is important that the value is weighted properly.

Similarity between feature vectors is calculated with the L_2 norm, or Euclidean distance. We also tested other distance measures (L_1, and Mahalanobis with different covariance matrices), but the L_2 norm 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.

4 Result

In this chapter the overall results are presented. We also describe our experiments investigating the effects of different pre-processing methods and noise sensitivity.

4.1 Final Configuration of the Search Engine

The components of the search engine and its internal parameters are as follows:

* **Image scaling:** Square images, size 24 × 24 pixels, for ”square characters” like a, e, and o. Rectangular images, for example 24 × 20 pixels, for ”rectangular characters” like l, i, and t. Align image borders for characters instead of center of gravity. Scaling with bilinear interpolation.
* **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)

4.2 Font Databases

The original database contains 2763 different fonts. Single characters are represented by images, typically around 100 pixels high. For evaluation, three test databases were created. The first contains images from the original 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 characters 'a' and 'b' from 100 randomly selected fonts were scanned (testdb1). For the second database (testdb2) the same 100 fonts were used, but this time all small characters were scanned, giving totally 2600 images. These images are used in the evaluation of the final search engine. The third test database also adopts the print and a scan procedure, as mentioned above, but with fonts that are not in the 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.

4.3 Overall Results

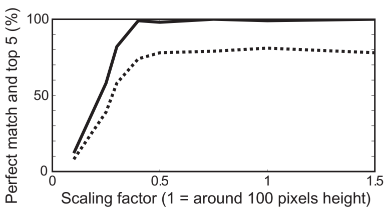
The search performance for different characters from testdb2 is shown in Table 3. The mean values for a perfect match and a top 5 result is 88,2 and 99,1 %. Characters like 'l' and 'i' that contain relatively few lines and details that can be used for distinguishing fonts from each other decrease the mean values rather significantly. Without 'l' and 'i', the mean values increase to 89,2 and 99,8 %. Since the input to the search engine is a text line, tricky characters like 'l' can be removed or weighted down, and usually the remaining characters will be sufficient for producing an accurate result.

4.4 Image Resolution and Rotation Influence

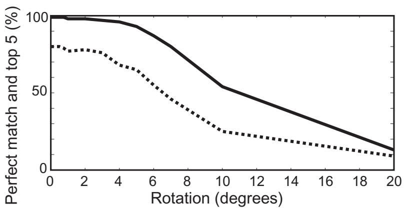
Experiments based on image resolution/scaling, JPEG compression and character rotation are presented in this section. Fig. 4 shows the relationship between query image size and search performance. In the figure we observe that a query image height below 40 pixels will decrease the retrieval performance significantly. We also evaluated the correlation between different JPEG compression rates and

Table 3. Search performance for different characters. (PE=PERFECT, T5=TOP5).

Character	PE	T5	Character	PE	T5	Character	PE	T5	Character	PE	T5
a	94	100	h	88	100	o	83	98	v	89	99
b	90	100	i	82	94	p	88	100	w	91	99
c	89	99	j	85	99	q	93	100	x	90	100
d	90	100	k	86	100	r	86	100	y	87	100
e	91	100	l	71	88	s	91	100	z	90	100
f	89	100	m	91	100	t	90	100			
g	88	100	n	91	100	u	91	100			
									MEAN	88.2	99.1



(a) The relationship between query image size and search performance.



(b) The relationship between query image rotation and search performance.

Fig. 4. Image resolution and rotation influence. Dashed line corresponds to perfect match, solid line corresponds to a top 5 result. (Character 'a' from testdb1).

search performance. In JPEG compression, the quality can be set between 0 and 100, where 100 corresponds to the best quality (lowest compression). The result showed that only when the quality is below 50 the retrieval performance is affected. Finally, the relationship between query image orientation (rotation) and search performance can be seen in Fig. 4. When input images are rotated more than 5 degrees, the performance declines sharply. However, an angle around or over 5 degrees is rarely encountered in real samples. In our experience rotation angles are below 1 degree after pre-processing.

4.5 An Example of a Complete Search

This section illustrates a complete search from input image to the font name output. An example of an input image can be seen in the top left part of Fig. 5. First edge filtering and the Hough transform are used to automatically rotate

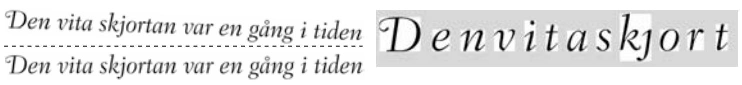
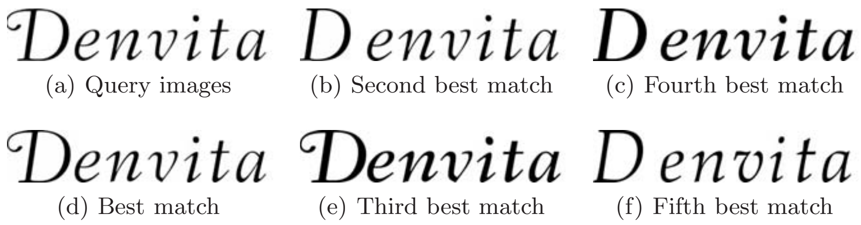

Den vita skjortan var en gång i tiden
Den vita skjortan var en gång i tiden
D e n v i t a s k j o r t

Fig. 5. 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

Fig. 6. (a) Query images (b)-(f) The five most similar fonts in the database

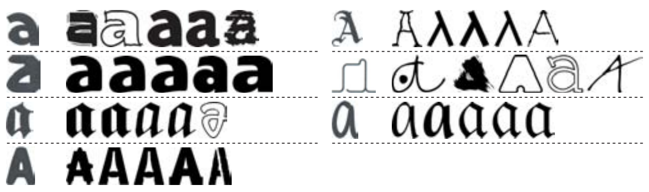


Fig. 7. Search result for character 'a' from seven fonts not present in the database. The search image to the left, followed by the five best matches.

the text line to a horizontal position. The result can be seen in the bottom left part of Fig. 5. Then the text line is segmented into sub images. The first 12 sub images are shown in the right side of Fig. 5. The user will then assign letters to sub images that are to be used in the database 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 since we are primarily interested in font recognition we did not spend time on developing a more sophisticated segmentation method. See [15] for a survey of character segmentation. Assigned images will be used as input to the search engine. Results from individual characters are weighted and combined to a final result, presenting the most similar fonts in the database. Fig. 6 shows the five most similar fonts for the query image. In this case the first seven characters were assigned letters and used as input.

Ending this section we describe experiments with query images created with fonts not in the database. For these images the retrieval accuracy can't be measured, the result must be evaluated visually. Fig. 7 shows seven query images from fonts not present in the database, together with the five best matches.

5 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 retrieve the name of the font used to writing the text. Apart from font retrieval, the search engine can also be used as a pre-processor to OCR systems. In an OCR context, the method would be classified as a local approach since features are calculated for individual characters. The recognition is mainly based on eigenimages calculated from character images filtered with three edge filters. Even for the very large font database, containing 2763 fonts, the retrieval accuracy is very high. For individual characters, the mean accuracy for a perfect match is 89,2 %, and the probability to find the correct font name within the five best matches is 99,8 %. 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.

References

1. Turk, M., Pentland, A.: Eigenfaces for recognition. Journal of Cognitive Neuroscience 3(1), 71–86 (1991)
2. Sexton, A., Todman, A., Woodward, K.: Font recognition using shape-based quad-tree and kd-tree decomposition. In: JCIS 2000, pp. 2: 212–215 (2000)
3. Östürk, S., Sankur, B., Abak, A.T.: Font clustering and cluster identification in document images. Journal of Electronic Imaging 10(2), 418–430 (2001)
4. Morris, R.A.: Classification of digital typefaces using spectral signatures. Pattern Recognition 25(8), 869–876 (1992)
5. Baird, H.S., Nagy, G.: Self-correcting 100-font classifier. In: Proceedings of SPIE, vol. 2181, pp. 106–115 (1994)
6. Cooperman, R.: Producing good font attribute determination using error-prone information. In: SPIE, vol. 3027, pp. 50–57 (1997)
7. Jung, M., Shin, Y., Srihari, S.: Multifont classification using typographical attributes. In: ICDAR '99, p. 353. IEEE Computer Society Press, Los Alamitos (1999)
8. Lee, C.W., Jung, K.C.: NMF-based approach to font classification to printed english alphabets for document image understanding. Modeling Decisions for Artificial Intelligence, Proceedings 3558, 354–364 (2005)
9. Khoubyari, S., Hull, J.J.: Font and function word identification in document recognition. Computer Vision and Image Understanding 63(1), 66–74 (1996)
10. Shi, H., Pavlidis, T.: Font recognition and contextual processing for more accurate text recognition. In: ICDAR '97, pp. 39–44. IEEE Computer Society, Los Alamitos (1997)
11. Zhu, Y., Tan, T.N., Wang, Y.H.: Font recognition based on global texture analysis. IEEE T-PAMI 23(10), 1192–1200 (2001)
12. Ha, M.H., Tian, X.D., Zhang, Z.R.: Optical font recognition based on Gabor filter. In: Proc. 4th Int. Conf. on Machine Learning and Cybernetics (2005)
13. Yang, F., Tian, X.D., Guo, B.L: An improved font recognition method based on texture analysis. In: Proc. 1st Int. Conf. on Machine Learning and Cybernetics (2002)
14. Avilés-Cruz, C., Rangel-Kuoppa, R., Reyes-Ayala, M., Andrade-Gonzalez, A., Escarela-Perez, R.: High-order statistical texture analysisfont recognition applied. Pattern Recognition Letters 26(2), 135–145 (2005)
15. Casey, R.G., Lecolinet, E.: A survey of methods and strategies in character segmentation. IEEE T-PAMI 18(7), 690–706 (1996)