Font Group Identification Using Reconstructed Fonts

Michael P. Cutter[β], Joost van Beusekom[βγ], Faisal Shafait[βγ], Thomas M. Breuel[β]

[β]University of Kaiserslautern, Germany
[γ]German Research Center for Artificial Intelligence (DFKI)

ABSTRACT

Ideally, digital versions of scanned documents should be represented in a format that is searchable, compressed, highly readable, and faithful to the original. These goals can theoretically be achieved through OCR and font recognition, re-typesetting the document text with original fonts. However, OCR and font recognition remain hard problems, and many historical documents use fonts that are not available in digital forms. It is desirable to be able to reconstruct fonts with vector glyphs that approximate the shapes of the letters that form a font. In this work, we address the grouping of tokens in a token-compressed document into candidate fonts. This permits us to incorporate font information into token-compressed images even when the original fonts are unknown or unavailable in digital format. This paper extends previous work in font reconstruction by proposing and evaluating an algorithm to assign a font to every character within a document. This is a necessary step to represent a scanned document image with a reconstructed font. Through our evaluation method, we have measured a 98.4% accuracy for the assignment of letters to candidate fonts in multi-font documents.

Keywords: Font Reconstruction, Font Identification, Reconstructed Font, Token Matching

1. INTRODUCTION

Digitization of libraries and archives is proceeding at a rapid pace, motivated by making culturally and historically important information more widely accessible. The raw digitized page images need to be represented in a format that supports functions like reading, searching, and printing.

Existing solutions, like Adobe Acrobat®, achieve this via overlaying a compressed page image on top of (usually) errorful OCR output. In contrast, our approach yields PDF files that are compressed, scalable and have improved readability. The original font can be reconstructed and then repurposed to regenerate the document faithfully. Using a digitally reconstructed font instead of hidden text also means that the document's glyphs are scalable and reflowable. Someone can read the book on a constrained resolution device, such as a mobile phone. Going the other direction, a visually challenged individual can blow the document up on a large monitor. In both cases, the user sees a visually faithful version of the book regardless of the display size.

Our work builds on work in token based compression techniques such as DjVu[1–4] and Google Books.[5] However, standard token-based compression methods do not attempt font reconstruction. Adobe has realized the advantages of font reconstruction with the recent addition to Adobe Acrobat 9, Clear Scan®. A document compressed by font reconstruction is similar to a document compressed by Clear Scan. An Adobe blog[6] discusses the advantages of Clear Scan over traditional compression techniques.

Font reconstruction is related to the field of font identification, which has been explored with supervised and unsupervised techniques. A similar technique was developed by Serdar et al.[7] to cluster fonts in an unsupervised fashion. For evaluation, they prepared a set of 4420 character images from 65 fonts with bitmaps resized to 32x32. Multiple font distance metrics were explored. The final font cluster classification in their technique is addressed by randomly selecting characters from a region; the most common font among the selected characters is considered the dominate font of that region.

A method by Avilées-Cruz et al.[8] preprocesses the words of a document image to make sure they are monospaced. Next, their technique performs feature extraction on various sized texture windows. Then their technique relies on the expectation maximization algorithm to cluster the texture windows into font groups.

Supervised font identification methods make use of large databases of fonts. Solli et al.[9,10] compare each character to a database of over two thousand fonts. All of the edges of the character images are extracted by

Sobel filters, and then eigen-images are calculated. Their method ranks the likelihood of a character belonging to each font, and 99% of the time the correct font is one of the top five proposed. Khoubyaricviu et al.'s[11] font identification technique is similar, but instead of segmenting words into individual characters, their method compares frequent articles such as 'the' to a font database.

The objective of this work is the extension of our previous work in font reconstruction.[12] The previous work's focus was to find a representative candidate font for each font present in a document. Letters from the document are categorized as belonging to the same reconstructed font based on their co-location in the document's words. The candidate font is meant to contain an alphabet consisting of letters that come from the same original font with a high degree of accuracy. A complete candidate font will contain at most 52 tokens, since its objective is to find 26 lower case and 26 uppercase tokens, each of which represents clusters of alphabetical letters; this will leave many of the letters in the document without a font. This paper's new contribution is an algorithm to assign all the remaining letters to a candidate font. The algorithm utilizes both the location of the letters on the page and their shape to assign the letters to a candidate font. This work is being carried out because, in order for the document to be recreated with the reconstructed font, it must be known which font to print each letter with.

The rest of this paper is structured as follows: in Section 2 the previous unsupervised font reconstruction based on token co-occurrence technique and the new font identification algorithm are described, in Section 3 our method is evaluated for different values of parameters, the results are discussed in Section 4, and finally, in Section 5 conclusions and future work are presented.

2. METHOD

Before we introduce the method in detail, we will introduce some basic terminology so that this section will be easier to understand.

2.1 Terminology

Letter: A letter is a character segmented from its adjacent characters. In this work, the distinction between a character and a letter is that, a letter is segmented using a statistical language model using OCRopus[13–15] and, with a high degree of probability, only represents a single alphabetical letter.

Character: In this work a character is a connected component in the document.

Character class: The character class of a letter or character is which alphabetical letter a character or letter represents.

Token: A token is the result of merging similar images of letters together. Each token has a unique ID; all of the original locations of the letters that are represented by this token can be described by this token's ID. This token ID representation can be seen in the pipeline Figure 1(b).

Reconstructed Glyph: A reconstructed glyph is created by tracing the outline of the image of the token and adding necessary parameters so that it can be typeset on the page the same as it was found.

Token Co-occurrence Graph: The token co-occurrence graph is an abstraction of the layout of token IDs on the pages of a document. Each node represents a token ID and each edge is formed when two token IDs are in the same space delimited sequence of token IDs.

Reconstructed Candidate Font: A reconstructed candidate font is a collection of token IDs that have been clustered together from individual letter images. A reconstructed font represents a single alphabet including all the lower case and upper case letters, containing a maximum of 52 tokens. It is common, however, for a reconstructed candidate font to be missing several unlikely letters, such as a capital "Z", when composed from only a few pages. This does not influence the representation of the document because it is likely that since the document did not contain these glyphs to begin with, a reconstructed candidate font intended to reproduce that document does not need these characters to reproduce it faithfully.

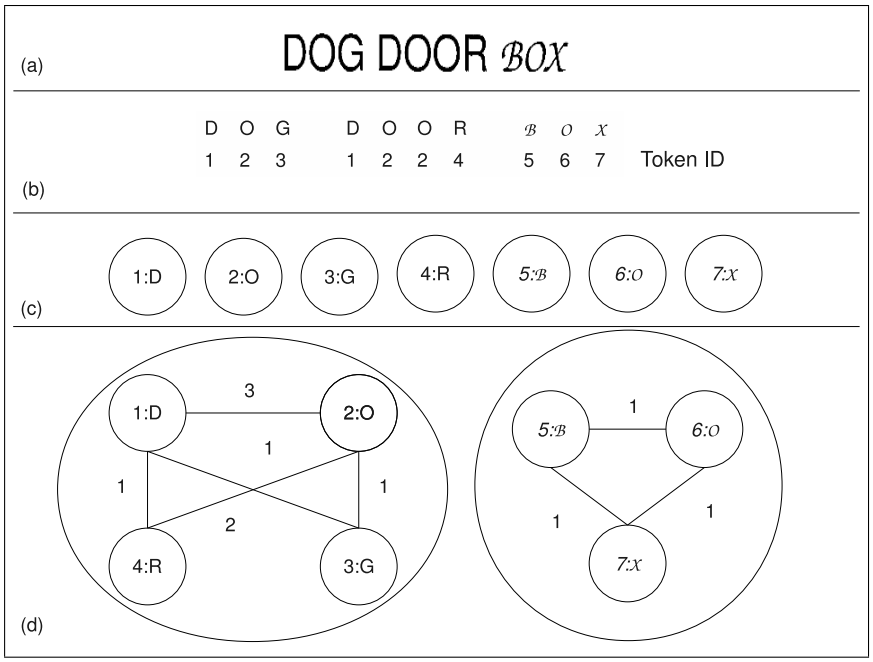


Figure 1. Font reconstruction pipeline. (a): Original text line. (b): Tokenization assigns an ID (number below each letter) to each connected component. (c): Each token ID is represented as a node. (d): The nodes connected into the co-occurence graph which is then partioned into candidate fonts.

2.2 Unsupervised Font Reconstruction Based on Token Co-occurrence

Our previously published[12] unsupervised font reconstruction based on token co-occurrence method can be summarized graphically by Figure 1. The technique is explained in this section for the reader's ease in understanding the rest of the work.

The first step of the unsupervised font reconstruction based on token co-occurrence technique is to apply OCR to the document. The OCR system, OCRopus, outputs segmented and labeled letters, which become the input to a font reconstruction algorithm. Letter shapes with the same character label are compared using an edge sensitive weight metric, designed to only merge letters when they come from the same font and have the same character class. The following equations are used to find the token prototype in the same bin as the candidate image that has the lowest FinalError score.

$$\text{error} = \sum_{x=0}^{W}\sum_{y=0}^{H} M(x,y) \times \|T(x,y) - I(x,y)\| \quad (1)$$

$$\text{error}_I = \sum_{x=0}^{W}\sum_{y=0}^{H} M(x,y) \times I(x,y) \quad (2)$$

$$\text{error}_T = \sum_{x=0}^{W}\sum_{y=0}^{H} M(x,y) \times T(x,y) \quad (3)$$

$$\text{FinalError} = \min\left(\frac{\text{error}}{\text{error}_I}, \frac{\text{error}}{\text{error}_T}\right) \quad (4)$$

In the previous equations, T is the token prototype and I is the image being assessed as a possible match. I and T are first aligned using their centroids. A mask, M, of the outline of the letter's shape is created using

mathematical morphology. Letters considered matches after this comparison are represented by the same token prototype, which is a blend of the shape of the letters it represents.

The document can now be represented as a sequence of token IDs and spaces (see Figure 1 part (b)). The next step is to construct a token co-occurrence graph, in which the nodes are token IDs and each edge represents a co-occurrence of the token IDs in the same word. A word in this context is a string of token IDs that are delimited by recognized spaces. The token co-occurrence graph is segmented by a graph partitioning algorithm that finds the optimal set of token IDs for each candidate font to have the highest possible LinkScore.

$$\text{LinkScore}(F_i) = \sum_{t_k \in F_i} \sum_{t_j \in F_i} G(t_k, t_j) \times H(j,k) \tag{5}$$

$$H(j,k) = \begin{cases} 1 & : j \neq k \\ 0 & : j = k \end{cases} \tag{6}$$

In Equation 5, F_i is the candidate font and $G(x, y)$ is a function that gives the edge weight in the token co-occurrence graph between the node x and y. The graph partitioning algorithm is under the constraint that each candidate font, F_i, represents an alphabet, meaning that the same font can not have two tokens that represent 'a'. Additionally, the optimization of LinkScore is under the constraint that a token can only belong to one candidate font or none at all. The final constraint is that a candidate font is limited to only be a set of at most 52 tokens, 26 upper case plus 26 lower case tokens that represent letters.

The final output of the unsupervised font reconstruction based on token co-occurrence technique are candidate fonts, F_i, which contain representative letters of a reconstructed fonts alphabet. Since at most 52 tokens can be included in a reconstructed font, the remaining tokens are unfonted, meaning they are not part of any reconstructed candidate font.

2.3 Font Identification

The font identification algorithm described next is applied after the entire unsupervised font reconstruction based on token co-occurrence technique (See Figure 1) has been applied to the document. The goal of this font identification algorithm is to accurately label the font of all of the unfonted tokens, tokens that are not part of any candidate font.

The first step is to apply a first pass, proximity based labeling algorithm. In the following equations j is the index of a letter in the chronological reading order list of letters within a document, $f_{\hat{i}}$ is the assigned font class, f_i is a font class and R_{f_i} is the set of all letter locations in a candidate font with the label f_i.

$$j_{f_{\hat{i}}} = \underset{f_i \in f}{\operatorname{argmax}}\, g(f_i, j) \tag{7}$$

$$g(f_i, j) = \sum_{v=(j-K)}^{(j+K)} \frac{1}{\|j - v\|} \times h(v, f_i) \tag{8}$$

The algorithm uses Equation 7 for index j of all the letters in the document in chronological reading order, to determine the candidate font to assign j to. Parameter K controls the reach of the first pass font labeling algorithm. The consequence for the decaying weight term(see Equation 8) is that a letter's font class assignment is weighted higher for the closest proximity neighboring font.

$$h(v, f_i) = \begin{cases} 1 & : x \in R_{f_i} \\ 0 & : x \notin R_{f_i} \end{cases} \tag{9}$$

The indicator function 9 ensures that only letters that are part of a candidate font can influence the font class assignment. If none of the K neighbors are part of a reconstructed font, the letter is given no label.

After the neighborhood proximity first pass labeling, the next step for the algorithm is to utilize the token's prototype shape to determine its reconstructed font class. Every token prototype in a document is resized to a size of 32 by 32, by using the PIL[16] BICUBIC resizing function. The two dimensional image is then converted to a one dimensional vector of length 1024 ($32 \times 32 = 1024$). For this application, a nearest neighbor approach using only pixel values is a reasonable choice. This representation allows the classifier to differentiate visually different representations of the same character, as obtained by the tokenization.[12] Improvements using a more sophisticated similarity measure may be possible, although the expected impact is low.

The images of the tokens that have been partitioned to form a candidate reconstructed font become the labeled training data for the nearest neighbor model, which is responsible for assigning an unfonted token to a candidate font. Every image of each unfonted token is compared to this classifier, which classifies the font group of the token by whatever candidate font group is the nearest neighbor to the query image. However, the nearest neighbor (n) to the candidate image (j) is subject to Euclidean distance $d(I_j, I_n) < \alpha$, or else the token of the candidate image remains without an assigned candidate font.

3. EVALUATION

As an initial experiment of the font identification technique, we experimented on a ten page document image. The document was synthetically created to consist of three fonts. The advantage of synthetically creating the document is knowing for certain the fonts involved. The digital document is exported as an image and then fed as the input to the unsupervised font reconstruction based on token co-occurrence pipeline (see Figure 1). Then the font identification algorithm discussed in section 2.3 is used to classify the font of each letter in the document.

We ran the experiment for different values of parameters α and K. Parameter α was tested with values between 0 and 5000 in intervals of 100. For each of these value of α, the parameter K was set successively from zero to four in intervals of one. With 50 values of α and 5 values of K, the algorithm was run a total of 250 times. We discuss the quality of our algorithm using the following metrics.

Coverage: the number of letters in the document whose distance satisfied: $d(I_j, I_n) < \alpha$, divided by the total number of letters.

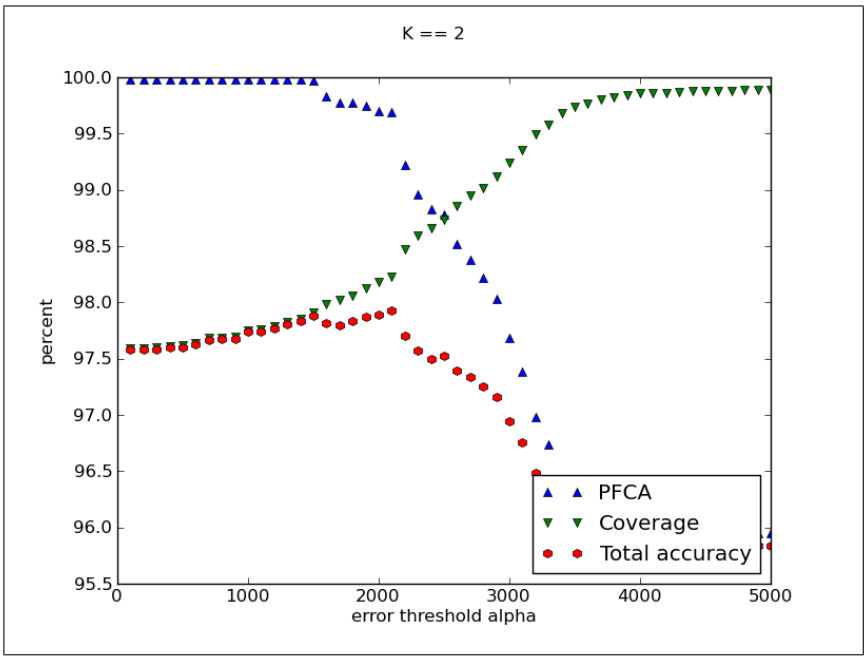
Predicted Font Classification Accuracy (PFCA): the measure of accuracy for the font predictions made for letter's whose distance from any font classifier is within α.

Total Accuracy: The total accuracy is defined as the product of assignment prediction and coverage. Suppose there is a document with 1000 letters. If the algorithm is able to classify all the letters correctly, then the accuracy is 100%. If the algorithm is only able to make classifications for 900 letters, but each classification is perfect, then the algorithm has achieved 90% accuracy. If the algorithm is only able to make classifications for 90% of the letters, and of those letters 80% of the classifications are correct, then the total number of accurately labeled letters is 720.

The parameterized values that result in maximum total accuracy are α equal to 1500 and K equal to four. For these values, the algorithm's maximum total accuracy is 0.9852%, the Coverage is 0.9844% and the PFCA is 0.9994%. The maximum accuracy achieved in these experiments with parameter K equal to zero is 0.95%, using α equal to 2100. The results of this sequence of experiments can be seen graphically in Figure 2 and Figure 3.

4. DISCUSSION

The error rate of our algorithm can be attributed to sensitivity to imperfect OCR. Two types of OCR errors can have visible adverse consequences for a reconstructed font since our technique is applied after OCR. The first type of error is the assignment of a wrong character label by the OCR system. This technique is robust to this type of error due to the greedy initial alphabet selection. Tokens that represent the greatest amount of letters are given priority over tokens that represent fewer letters. If a token in a fonts alphabet is labeled incorrectly



(a) K=2

Figure 2. Trade off between coverage and predicted font classification accuracy (PFCA). Note that even though coverage steadily increases, PFCA does not fall below 100% until the match threshold α reaches 1500.

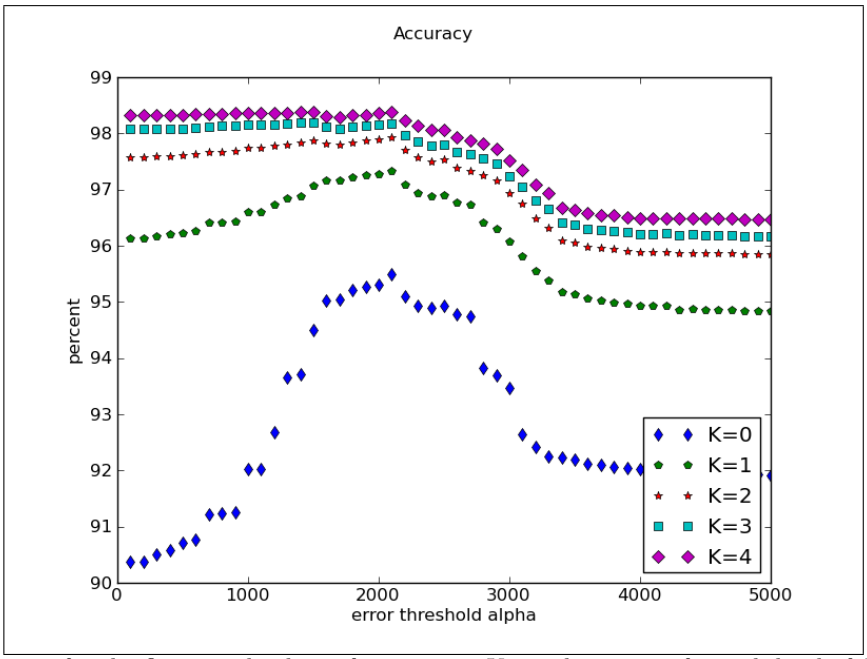


Figure 3. Total accuracy for the five tested values of parameter K, total accuracy for each level of K from Figure 2 can be seen on this graph. For these experiments, increasing K is strongly correlated with increases in total accuracy. This phenomena is explainable because parameter K controls the number of surrounding letters to a token in a reconstructed font's alphabet are also considered part of the same font. The effect on total accuracy is greatest when K increases from zero to one. Setting parameter K to a value greater than zero activates the proximity labeling algorithm. This ensures that letters next to a token known to be in a candidate font is considered part of the same candidate font. Since K controls the capability for letters to be turned into fonts based on proximity, the success of larger K values is based on how interspaced fonts are in the document. Therefore, further testing on larger documents will be used to determine an appropriate domain invariant value for K.

e.g. a ‘c’ is recognized as a ‘e’, then the reconstructed document will unfortunately contain this error for every ‘c’ in this font.

The second type of error is the recognition of extra, fictitious space. This error is much more damaging to the quality of a reconstructed font since the word level co-occurrence is how the original candidate font’s alphabet is selected. If every time a ‘c’ of a particular font is recognized with an extra space on either side of the letter, the reconstructed font representing the font of this ‘c’ will not include the ’c’ because the letter will not not co-occur in a word with any other letter. Unfortunately, with the current implementation, the token(s) representing this ‘c’ will not be classified as belonging to a reconstructed font because none of the reconstructed fonts will contain a similar enough image. A potential fix to this issue would be to modify the unsupervised font reconstruction based on token co-occurrence technique to consider single letters next to words as co-occurrences in the token co-occurrence graph.

It is important to point out that this technique is appropriate for documents with any amount of distinct fonts. If the fonts of the document are almost completely visually the same, then during the tokenization stage letters from different fonts may be merged into a single token. If this token becomes part of a candidate font, the error rate will greatly increase. However, this raises the question of how to evaluate these types of false mergings. If two fonts were almost the same to begin with the output of a font reconstructed document would not be harmed significantly. This slight reduction in visual accuracy would be misrepresented by a high candidate font assignment error rate.

It is desirable that in the future we test this technique on a real document that contains multiple distinct fonts. Ideally, the real document will be scanned at a high enough quality for OCR to work almost completely error free. The document‘s various regions containing different fonts must be hand labeled in order to evaluate the accuracy of our technique.

5. CONCLUSION

Font reconstruction is being developed for the Decapod[17] project, an open source system to capture books for digital libraries. Many people across the developing world pay a high premium for a shared Internet connection in order to become interconnected with the rest of the world. Often these connections are extremely low bandwidth, and therefore a requirement for globally accessible digital archives is a small file size for each digital novel.

The evaluation of our font identification algorithm has determined the highest accuracy achievable with the experimented parameter values to be 98.4%. The next step for font reconstruction is to parametrize glyphs so they lay on the page the same way as when they were extracted.[18] Glyphs can be automatically traced by using Potrace.[19] These glyphs can be represented by many different types of fonts. It might even be possible to automatically generate parametric fonts,[20] which could solve the problem of incomplete reconstructed fonts. Additionally, a challenge will be determining the appropriate kerning pairs without the existence of every possible combination of letter pairs.

The future work for font reconstruction is to combine reconstructed fonts with text image segmentation in order to produce full Mixed Raster Content[21] documents. By combining all of our previous work in font reconstruction, we can benchmark a completely font reconstructed document against a document produced by DjVu and Google Books. We hope that these font reconstructed digital documents meet the criteria for inclusion in a world class digital archive because they will be both highly compressed and high quality representations of the original documents.

ACKNOWLEDGMENTS

Parts of this project were financed by the Mellon Foundation’s DECAPOD project.

REFERENCES

[1] Haffner, P., Bottou, L., Howard, P. G., and Lecun, Y., "DjVu: Analyzing and Compressing Scanned Documents for Internet Distribution," in [*In Proceedings of the International Conference on Document Analysis and Recognition*], 625–628 (1999).

[2] Bottou, L., Haffner, P., Howard, P. G., Simard, P., Bengio, Y., and Le Cun, Y., "High Quality Document Image Compression with DjVu," *Journal of Electronic Imaging* **7**(3), 410–425 (1998).

[3] Mikheev, A., Vincent, L., Hawrylycz, M., and Bottou, L., "Electronic Document Publishing using DjVu," in [*Proceedings of the IAPR International Workshop on Document Analysis (DAS'02)*], (August 2002).

[4] Le Cun, Y., Bottou, L., Haffner, P., Triggs, J., Riemers, B., and Vincent, L., "Overview of the DjVu Document Compression Technology," in [*Proceedings of the Symposium on Document Image Understanding Technologies (SDIUT'01)*], 119–122 (April 2001).

[5] Langley, A. and Bloomberg, D. S., "Google Books: Making the public domain universally accessible," in [*Proceedings of SPIE Volume 6500, Document Recognition and Retrieval XIV*], 1–10 (2007).

[6] "Adobe's blog discussing Clear Scans advantages." http://blogs.adobe.com/acrolaw/2009/05/better_pdf_ocr_clearscan_is_smal.html/.

[7] Öztürk, S., Sankur, B., and Abak, A. T., "Font clustering and cluster identification in document images,"

[8] Avilés-Cruz, C., Villegas, J., and Escarela-Perez, R., "Unsupervised Font Clustering Using Stochastic Version of the EM Algorithm and Global Texture Analysis," in [*Progress in Pattern Recognition, Image Analysis and Applications*], 3–25 (2004).

[9] Martin Solli, R. L., "FyFont: Find-your-Font in Large Font Database," in [*Image Analysis*], 432–441 (2007).

[10] "FyFont, a visual search engine for fonts." http://media-vibrance.itn.liu.se/fyfont/.

[11] Khoubyari, S. and Hull, J. J., "Font and Function Word Identification in Document Recognition," *Computer Vision and Image Understanding* **63**(1), 66–74 (1996).

[12] Cutter, M. P., Beusekom, J. v., Shafait, F., and Breuel, T. M., "Unsupervised font reconstruction based on token co-occurrence," in [*Proceedings of the 10th ACM symposium on Document engineering*], *DocEng '10*, 143–150, ACM, New York, NY, USA (2010).

[13] "The OCRopus(tm) open source document analysis and OCR system." http://code.google.com/p/ocropus/.

[14] Breuel, T., "The OCRopus Open Source OCR System," in [*Proceedings of the Document and Retrival XV, IS&T/SPIE 20th Annual Symposium 2008*], Yanikoglou, B. and Berkner, K., eds., SPIE (2008).

[15] Breuel, T., "Recent progress on the OCRopus OCR system," in [*MOCR'09: Proceedings of the International Workshop on Multilingual OCR*], 1–10, ACM, New York, NY, USA (2009).

[16] "Python Imaging Library (PIL)." http://www.pythonware.com/products/pil/.

[17] "The Decapod project." http://sites.google.com/site/decapodproject/.

[18] Meyer-Lerbs, L., Schuldt, A., and Gottfried, B., "Glyph extraction from historic document images," in [*Proceedings of the 10th ACM symposium on Document engineering*], *DocEng '10*, 227–230, ACM, New York, NY, USA (2010).

[19] http://potrace.sourceforge.net/potrace.pdf, September (2003).

[20] Hassan, T., Hu, C., and Hersch, R. D., "Next generation typeface representations: revisiting parametric fonts," in [*Proceedings of the 10th ACM symposium on Document engineering*], *DocEng '10*, 181–184, ACM, New York, NY, USA (2010).

[21] Queiroz, R. D., Buckley, R., and Xu, M., "Mixed raster content (MRC) model for compound image compression," in [*Proc EI 99, VCIP, SPIE*], 1106–1117 (1999).