Conf-950869--1

LA-UR 95-0021

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

TITLE: **AUTOMATIC SCRIPT IDENTIFICATION FROM IMAGES USING CLUSTER-BASED TEMPLATES**

AUTHOR(S): J. Hochberg, L. Kerns, P. Kelly, T. Thomas

SUBMITTED TO: 3rd International Conference on Document, Analysis, & Recognition
Montreal, Canada
August 14-16, 1995

DISCLAIMER

MASTER

Los Alamos

Los Alamos National Laboratory
Los Alamos New Mexico 87545

Automatic script identification from images using cluster-based templates

Judith Hochberg, Lila Kerns, Patrick Kelly, and Timothy Thomas

Computer Research Group (CIC-3)

Los Alamos National Laboratory

Contact author: Judith Hochberg

Mail Stop B265

Los Alamos National Laboratory

Los Alamos, NM 87545

(505) 667-4679 (phone)

(505) 665-5220 (fax)

judithh@lanl.gov

Automatic script identification from images using cluster-based templates

Abstract

We have developed a technique for automatically identifying the script used to generate a document that is stored electronically in bit image form. Our approach differs from previous work in that the distinctions among scripts are discovered by an automatic learning procedure, without any hands-on analysis. We first develop a set of representative symbols (*templates*) for each script in our database (Cyrillic, Roman, etc.). We do this by identifying all textual symbols in a set of training documents, scaling each symbol to a fixed size, clustering similar symbols, pruning minor clusters, and finding each cluster's centroid. To identify a new document's script, we identify and scale a subset of symbols from the document and compare them to the templates for each script. We choose the script whose templates provide the best match. Our current system distinguishes among the Armenian, Burmese, Chinese, Cyrillic, Ethiopic, Greek, Hebrew, Japanese, Korean, Roman, and Thai scripts with over 90% accuracy.

Keywords

script identification, language identification, optical character recognition

1. Introduction

In order to make proper use of a document it is important to know which language it is written in. For example, documents may need to be sorted by language for referral to an appropriate human reader or translator, or to an appropriate piece of document analysis software. The traditional way to identify a document's language is for a skilled individual to examine the document, whether in hardcopy or in electronic form. The question of how to automate this process is an active one in the field of document processing. Automation would speed up the identification process and reduce its cost.

To identify the language of an electronic document in ASCII form, where each character (*A*, *?*, *5*, etc.) is directly represented by an eight-bit code, one can use an *n-gram* algorithm that looks for character sequences that are common in different languages (Church 1986). For example, an English document is likely to have a high percentage of *T-h-e* and *t-h-e* sequences. Universal character code sets such as UNICODE, which can represent even non-phonetic scripts such as Chinese, have recently been introduced and can also be analyzed with n-grams.

But what of an electronic document that is stored as an image: that is, as a visual representation (bitmap) of the document? Here the main challenge is identifying the script the document uses. In the case of a script used by a single language (e.g., Korean), script identification is tantamount to language identification. In the case of a script used by several languages (e.g., Roman, Cyrillic), language identification is relatively simple once the script has been identified. For example, one can convert a Roman document to ASCII through optical character recognition (OCR) and perform n-gram analysis. Alternatively, one can scan the document for word shapes that correspond to familiar character sequences (Sibun & Spitz 1994). Thus *T-h-e* and *t-h-e* sequences in a typical English document can be identified, without OCR, as words consisting of two tall characters followed by a shorter character.

For hardcopy documents, script identification is likewise a crucial step in automatic language identification. Such documents are scanned into computers as images. The arguments presented just above then apply.

The Fuji Xerox group, in Palo Alto, California, has done extensive research on the topic of automatic script identification. Their approach combines automated and hands-on optical analysis of a training corpus to identify significant characteristics of the different scripts. In their current system, as summarized in Spitz 1994b, an initial division is made between Asian scripts (Chinese, Korean, or Japanese) and Roman. The basis of this distinction is that upward concavities are distributed fairly evenly along the vertical axis of Asian characters, while they usually appear at specific vertical locations in Roman characters. This difference stems from the greater complexity of Asian characters. Once this initial distinction is made, further distinctions among Asian scripts are made on the basis of character density.

Unfortunately, this approach requires a new hands-on analysis for each script. The salient distinctions between Chinese, Korean, Japanese, and Roman emerged from a careful inspection of the optical characteristics of these individual scripts. A similar analysis would be required for any script added to the system.

In contrast to the Fuji Xerox system, our script identifier is capable of *automatically* learning distinctions among an arbitrary number of scripts. In essence, it discovers frequent character shapes in each script, then looks for instances of these in new documents. Character shapes are discovered by means of cluster analysis. In a training phase, we find textual symbols in a representative set of documents, cluster them, and calculate each cluster's pixel-by-pixel average. This centroid serves as a representative symbol, or *template*, for the cluster. To identify the script used in a new document, we compare a subset of symbols from the document to the templates for each script, and choose the script whose templates provide the best match.

Our system currently distinguishes among the Armenian, Burmese, Chinese, Cyrillic, Ethiopic, Greek, Hebrew, Japanese, Korean, Roman, and Thai scripts with over 90% accuracy. Most of the errors that the system makes are directly traceable to weaknesses in our current training corpus, and should be eliminated in our next round of training and testing.

While the most distinctive feature of our system is the automatic learning process, the system also differs from previous work in its treatment of character fragments (e.g., a separated *g* descender) and conjoined characters (e.g., a blurred *th* combination). We view these phenomena as non-traditional textual symbols that will invariably occur in document images, especially those that have been degraded through successive steps such as copying and scanning. We therefore include character fragments and joined characters in the clustering process along with normal textual symbols such as letters, numbers, diacritics, and punctuation marks. This perspective is in contrast to the more traditional view of character fragments and conjoined characters as exceptional. In this view, character fragments should be connected (e.g., by inclusion in a single character cell), while conjoined characters can be separated by a variety of techniques such as splaying and bounding rectangle shrinking (see, e.g., Spitz 1994a).

We believe that our treatment of character fragments and conjoined characters makes for a more flexible system, as we should be able to process highly degraded documents by including equally degraded documents in the training set. In addition, it should reduce the preprocessing time required for each document.

2. Method

The essence of our approach is to discover frequent character shapes in each script, then look for instances of these in new documents. This process has six steps:

1. Assemble a training and test set of documents for each script.
2. Find and scale textual symbols in the training set.

3. Cluster similar symbols within each script, and prune minor clusters.
4. Make templates by calculating each cluster's centroid (average symbol).
5. Find a subset of symbols in a test document.
6. Match these symbols to the templates.

Most of this work was done within the framework of the Khoros™ image processing system (Rasure & Williams 1991). This system provides a set of standard image processing modules (e.g., image band extraction) and both keyboard and visual interfaces for linking modules. It also provides a framework for writing new image processing programs that can draw on Khoros library routines and link with established Khoros modules.

The following sections describe these steps in more detail.

2.1 Dataset

Our dataset consisted of 176 documents from seventeen languages written in eleven scripts. Table 1 summarizes the number of documents from each language and script. The set was restricted to scripts with discrete characters. While we believe that connected scripts such as Arabic and Devanagari can be incorporated into the general framework of our approach, we want to focus on crucial issues such as system accuracy and speed before attempting this next step.

Table 1. Dataset

Script	Language	Number of documents
Armenian	Armenian	9
Burmese	Burmese	4
Chinese	Chinese	24
Cyrillic	Bulgarian	1
	Russian	4
Ethiopic	Amharic	19
	Tigrinya	3

Greek	Greek	2
Hebrew	Hebrew	9
Japanese	Japanese	5
Korean	Korean	18
Latin	Czech	3
	English	45
	Muskogee	3
	Slovak	4
	Turkish	2
Thai	Thai	21

Most of the document images in our dataset were from newspapers, magazines, or books; some were from computer printouts. All were scanned in from hardcopy. Image sizes ranged from 346×258 pixels to 2200×1701. Smaller documents represented a single paragraph of text, while larger documents contained several densely packed paragraphs. The Roman documents included samples of several different serif and sans serif fonts, both straight and oblique (*italic*). We were also able to find a variety of fonts for Chinese, Hebrew, Korean, and Thai.

All documents in the dataset were correctly oriented, or skewed by at most five or ten degrees. Our method could be adapted to more sharply skewed documents by applying standard line-straightening algorithms before training.

We divided the dataset into a training set of 115 documents and a test set of 61 documents, with proportionate representation from each script.

2.2 Textual symbols

We used a region growing algorithm to automatically locate all symbols in the training set. This algorithm scanned the image of a document. When it encountered a black pixel it then looked for another black pixel that was adjacent to the first pixel in any of eight directions: top, bottom, left, right, and the four diagonals. If it found an adjacent pixel, it then tried to grow outward from that

one as well. The set of contiguous black pixels is termed a *region*. The growing process for each region continued until the algorithm failed to locate any additional adjacent pixels. A *bounding box* -- a set of four (x, y) pixel coordinates defining the smallest possible rectangle around the region -- was identified for each region in the image.

While most regions corresponded to letters, diacritic marks, numbers, and punctuation marks, some corresponded to non-textual elements such as flecks, blobs, or borders. We removed these undesirable elements by filtering out regions containing fewer than 10 or more than 550 pixels, or whose bounding boxes were more than 80 pixels long or wide.

It is important to note that the resulting set of textual symbols included symbols that deviated from standard characters. As long as they were not too small or too large, character fragments and conjoined characters survived the filtering process. As described in the Introduction, we accept these as phenomena that occur in document images, and therefore required the system to learn them along with standard characters.

We scaled each textual symbol to a constant size (currently 30×30 pixels) using a standard resizing algorithm. Some information was lost in this process. For example, a vertical line (e.g., a sans serif l) and a horizontal line (e.g., a hyphen, or the Chinese character for the word *one*) were both scaled to a solid black square. However, this process accomplished the crucial goal of equalizing font (point) sizes between documents. It also altered the proportions of character components in a way that increased the similarity between fonts. For example, two equal-height capital T's with different-sized top strokes would have identical-sized top strokes after scaling.

2.3 Clustering symbols

Once all the textual symbols in the training documents for a script were identified and scaled, we clustered together similar symbols within the script. The purpose of this step was to group together different versions of the same symbols: for example, serif T and sans serif T.

The clustering algorithm proceeded as follows. All scaled symbols from the training documents for a single script were examined in turn. The first symbol was assigned to an initial cluster. Each subsequent symbol was assigned to the cluster to which it was most similar; if it differed markedly from all existing clusters, it was assigned to a new cluster.

Each cluster was represented by the first symbol assigned to it. The similarity between a new symbol and the symbol representing an established cluster was measured by inverse Hamming distance: the number of pixels that the two symbols had in common. The number of clusters generated depended on the setting of a parameter that determined whether a symbol was added to its most similar cluster, or was assigned to a new cluster. Under our current parameter setting, if a symbol had more than 650 pixels in common with the symbol that represented its most similar cluster, it was added to the cluster. If it had 650 or fewer pixels in common with any existing cluster, it was used to initiate a new cluster. We chose this parameter setting empirically by determining the number of clusters made at different settings and selecting what seemed to be a stable point. We also satisfied ourselves by inspection that this parameter setting caused visually similar symbols to be clustered together, while distinct symbols were assigned to different clusters.

After clustering the training symbols we *pruned* (eliminated) clusters with only one or two members. The reasoning behind this step was that, if the training set was representative of the range of documents that the system would encounter, then these minor clusters are not likely to be of much use in analyzing new documents. Pruning minor clusters also dramatically sped up the identification process by reducing the number of templates to which each new symbol was compared (see section 2.6, *Matching new symbols*).

Table 2 summarizes the number of training symbols in each script, the number of clusters made, and the number and percentage of clusters retained after pruning.

Table 2. Symbols and clusters				
		Number of Clusters		
Script	Number of Training Symbols	before pruning	after pruning	% Retained
Armenian	6272	116	99	85
Burmese	5826	354	202	57
Chinese	24927	2261	1183	52
Cyrillic	10975	488	258	53
Ethiopic	31722	371	297	80
Greek	1152	87	63	72
Hebrew	3206	78	47	60
Japanese	3478	394	212	54
Korean	6486	320	227	71
Latin	58359	797	493	62
Thai	13778	379	273	72

2.4 Making templates

Once symbols had been clustered, we calculated each cluster's centroid. This then became the representative symbol, or template, for the cluster.

Centroids were calculated as follows. As our algorithm assigned symbols to clusters, it generated a pixel-by-pixel sum of pixel values for each cluster. In the images, 1 represented a black pixel and 0 a white pixel. Thus, for example, in a cluster consisting of ten scaled symbols, each with a black pixel in the upper left-hand corner, the sum for that pixel would be 10. After all symbols in the training set were assigned to clusters, the summed values for each cluster were averaged by dividing by the number of symbols in the cluster. The result was the cluster's centroid.

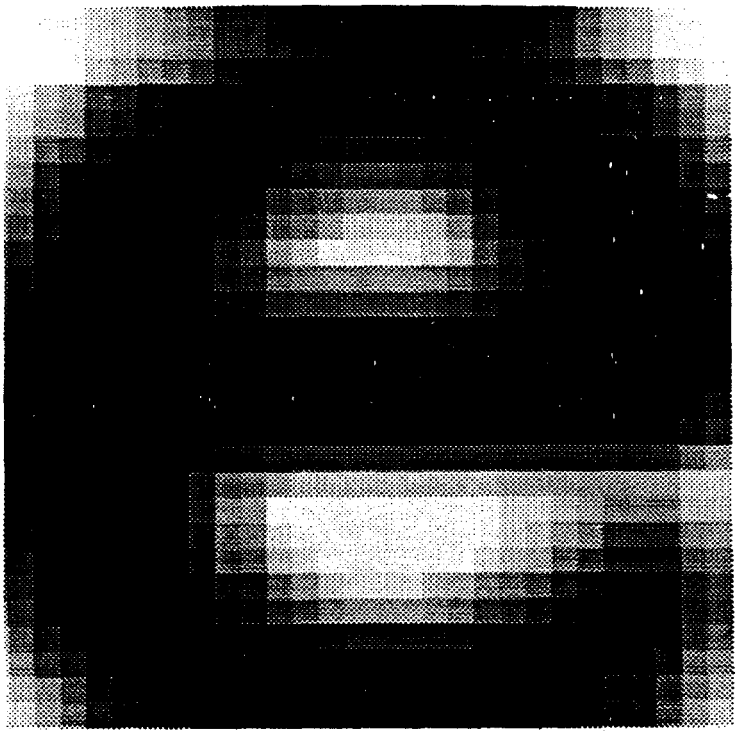

Figure 1. A template for lower-case *e*

A magnified example of a template is shown in Figure 1. The template has a blurred appearance because it combines aspects of the different symbols and/or fonts included in the cluster (it may include some instances of lower-case *c*). Pixels in the template are white if all corresponding pixels in its component symbols are white, black if all are black, and some shade of gray if its component symbols differed in that pixel. The template is stored as an eight-bit grayscale image.

2.5 Finding symbols in test documents

When processing a test document, we first identified and scaled its textual symbols, as described in section 2.2. We got good results, with a huge gain in time, by only using a subset of *N* symbols from the document. The results shown in section 3 are based on *N*s of 50 or 200 symbols.

2.6 Matching new symbols to the templates

We used two related methods for identifying the script of a test document. The methods gave roughly equivalent results, with the first method slightly stronger than the second. They can be summarized as follows:

i. Pick the script whose templates are, on average, closest to the symbols in the document.

ii. Pick the script that provides a best match for a plurality of the symbols in the document.

Both methods relied on finding the best match within each script for each test symbol examined. Best matches were found by comparing the test symbol to all templates from a script (after pruning minor templates as described in section 2.3) and picking the template that produced the smallest Euclidean distance. Euclidean distance was calculated as follows:

$$D_E(S,T) = \sqrt{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \left(S_{(x,y)} - T_{(x,y)} \right)^2}$$

where S and T are the symbol and template being compared, w and h are the width and height of S and T (in pixels), and $S_{(x,y)}$ and $T_{(x,y)}$ are single pixel values.

For the first method, as we processed the document we kept a running total of the best-match distances for each script. For each script, this total was:

$$\sum_{i=0}^{N-1} D_E\left(S_i , BM_{S_i} \right)$$

where N is the number of symbols examined, and BM_{S_i} is the template from the script that yields the best match for the symbol in question. After examining the N symbols we chose the script with the lowest total distance.

For the second method, as we processed each symbol we determined which script gave the best match overall (call this a *hit*), and kept a running total of the number of hits per script. At the end we picked the script with the most hits.

An enhancement that we applied to both of these methods took into account what we term the *reliability* of each template. This is a measure of the likelihood that a symbol that was a hit to a

template was from the script that the template belonged to. For example, if most hits to Cyrillic template #5 were symbols from Roman documents, then this template was unreliable. Most templates were highly reliable, while others, chiefly *blobs* (templates that are mostly black) were less reliable and frequently contributed to misidentifications of new documents. We identified the less reliable templates by making a second pass through the training set, matching training symbols to templates and keeping track of the percentage of correct hits for each template. We then defined a *reliability threshold* for each script: a percentage of correct hits below which a template was considered unreliable. For scripts with many templates, such as Chinese, we were able to set this threshold very high, at 90%. For scripts with few templates, such as Greek and Hebrew, we had to set the threshold lower, at 40%, to avoid eliminating many templates.[1] When processing new documents, we filtered out symbols that hit templates with reliability percentages below the relevant reliability threshold. For example, if we examined 50 symbols from a document, and 5 of these were hits to templates whose reliability percentages were below the thresholds set for their respective scripts, we only took the 45 remaining symbols into consideration in script identification.

3. Results

Table 3 summarizes our results for each script under four conditions: using an N of 50 versus 200 test symbols in each document, and with or without the reliability enhancement. All results in this table were obtained using the first method described in section 2.6 ("Pick the script whose templates are, on average, closest to the symbols in the document"). Our baseline result, obtained using 50 symbols and no reliability enhancement, was eight documents misclassified out of 61.

---

[1]This threshold setting is the only part of our procedure that we have not yet automated. We set thresholds by inspection after making a histogram of reliability percentages for each script's templates. We intend to automate this process in the next round of training and testing.

Six of these misclassifications were somewhat sensible, as they were Japanese and Korean documents misclassified as Chinese. In addition, one Armenian document was misclassified as Roman, and one Hebrew document as Thai.

Table 3. Results

		Number of documents incorrectly classified			
		No reliability enhancement		Reliability enhancement	
Script	Number of test documents	$N = 50$	$N = 200$	$N = 50$	$N = 200$
Armenian	3	**1**	**1**	**1**	**1**
Burmese	1	0	0	0	0
Chinese	7	0	0	0	0
Cyrillic	1	0	0	0	0
Ethiopic	8	0	0	0	0
Greek	1	0	0	0	0
Hebrew	3	**1**	**1**	**1**	**1**
Japanese	3	**2**	0	**1**	0
Korean	7	**4**	**4**	**4**	**3**
Latin	20	0	0	0	0
Thai	7	0	0	0	0
TOTAL	**61**	**8**	**6**	**7**	**5**

Increasing the number of symbols or adding the reliability enhancement both improved this result. Our best results, five misclassified documents, were achieved with 200 symbols and the reliability enhancement. When we use the second method described in section 2.6 ("Pick the script that provides a best match for a plurality of the symbols in the document"), the only difference was that four Korean documents were misclassified under all conditions.

We consider our best results, which amount to over 91% correct classification, to be promising, given the imbalances in our training set. All of the misclassified documents were from among the six scripts for which we had the fewest symbols in the training set (see Table 2).

Moreover, when we took a closer look at the five documents misclassified in our best condition, we found that most of these seemed to have been misclassified because of specific lacunae in the training set. The three misclassified Korean documents used font types that were not represented in the training set: the *godic* font, which acts as a kind of boldface, and a font that is analogous to a Gothic font for the Roman script. These fonts differ strikingly in appearance from the standard Korean font, *myung cho*, which was used in all of our Korean training documents. Samples of these three fonts are presented in Figures 2a-2c. Likewise, the single misclassified Hebrew document, shown in Figure 3a, was written in a modern, blockish font, while the training documents used more traditional fonts such as that illustrated in Figure 3b. The three Korean test documents in the *myung cho* font, and the two Hebrew test documents in more traditional fonts, were all correctly classified.

The lack of errors in our five best-represented scripts, and our analysis of the misclassified files, led us to conclude that our performance should be greatly improved by balancing the representation of scripts within the training set, and of fonts within each script. Certainly it is impractical to include samples of every possible font variation for each script set. Moreover, the system should be capable of reasonable generalizations: from Times to New York, for example. Rather, our results underscore the importance of including samples of the different *types* of fonts that will be encountered in new documents.

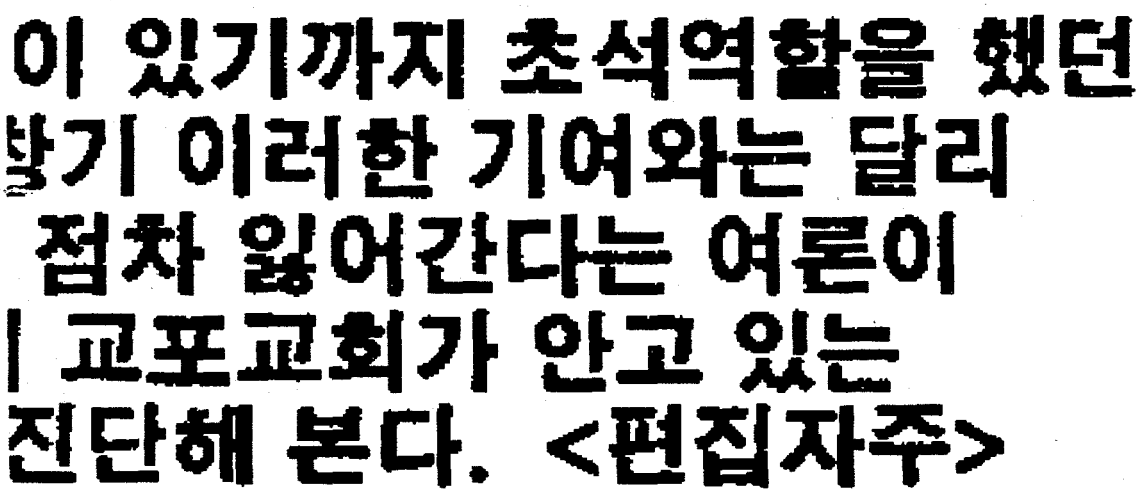

Figure 2a. Part of a misclassified Korean document in *godic* font

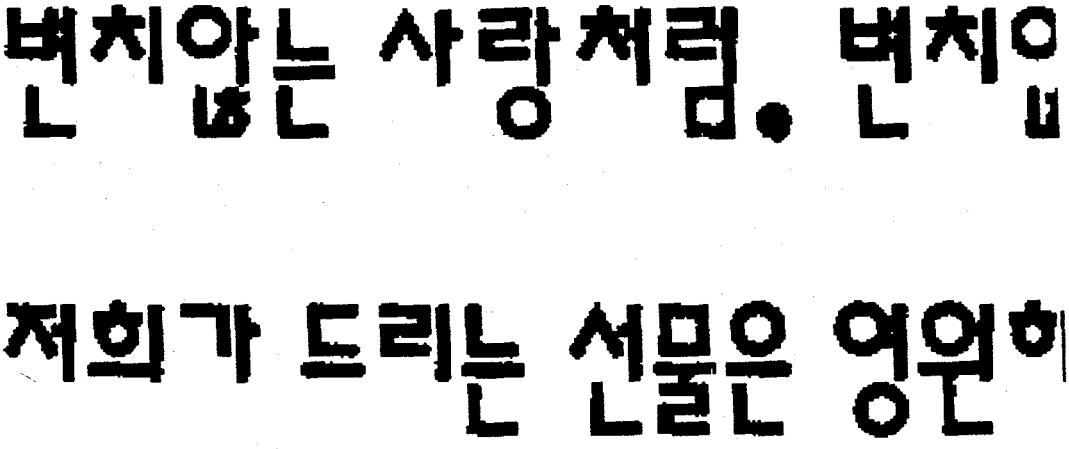

Figure 2b. Part of a misclassified Korean document in Gothic-like font

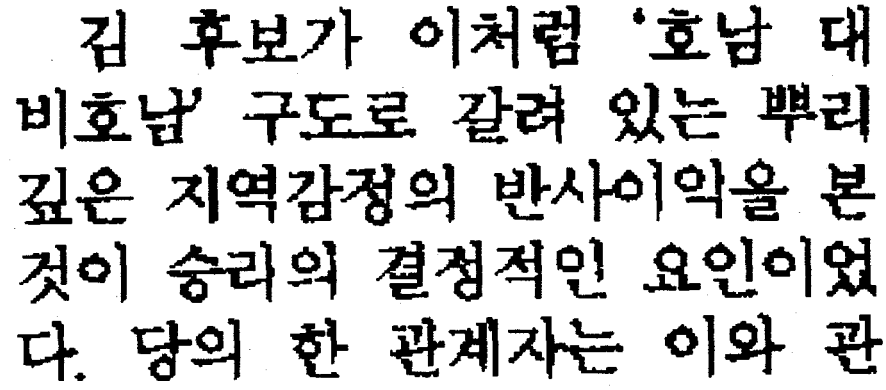

Figure 2c. Part of a correctly classified Korean document in standard *myung cho* font

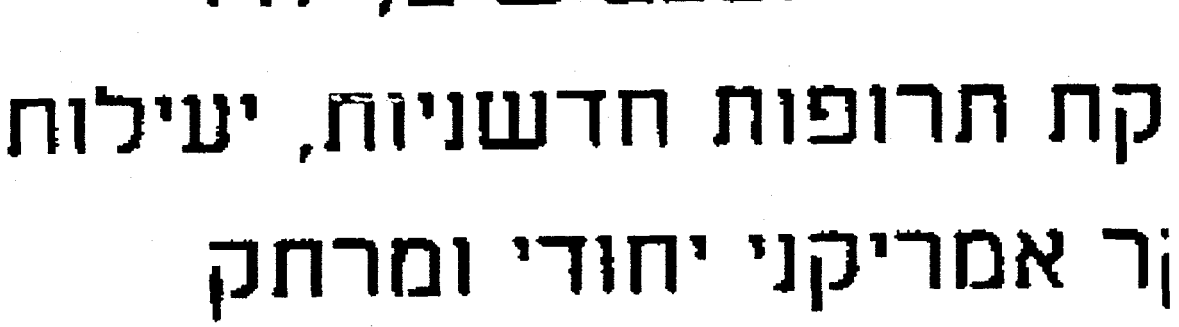

Figure 31. Part of a misclassified Hebrew document, in modern font

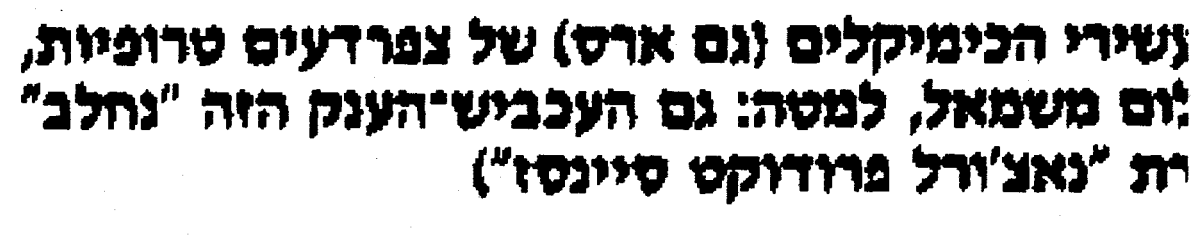

Figure 3b. Part of a correctly classified Hebrew document, in traditional font

4. Conclusion and discussion

The process we have described in this paper identifies the script of a document in bitimage form, choosing among elevent scripts with over 90% accuracy. The process trains and identifies automatically. Now that the various parameters in the system have been set (e.g., scaled image size, Hamming distance threshold for starting a new cluster), additional scripts could be added to the system without any hands-on effort.

Improving accuracy. Our top priority is to improve the accuracy of our system for the current set of eleven scripts. As discussed just above, we believe that the best way to do this is to construct a training set that is more balanced (i) among scripts, and (ii) among font types within scripts. This is the main focus of our current effort.

Improving speed. Our current system requires each test symbol to be compared to every template in every script, with each comparison involving a 900-pixel Euclidean distance calculation. This is a slow process even when examining only 50 or 200 symbols per document. We are exploring the possibility of replacing our current symbol representation with a more compact moment-based representation in order to speed up this process. Moment representations have proved useful for a simple OCR task (Cash & Hatamian 1987).

Clustering algorithm. Our current clustering algorithm is *hierarchical*, meaning that clusters are initiated with a single symbol, and augmented when similar symbols are found in the dataset. An *iterative* clustering algorithm, in which all symbols are given an initial cluster assignment, and cluster membership is iteratively adjusted until some error criterion is satisfied, would afford more of a guarantee that the final solution is optimal. We are planning to implement an extremely fast iterative algorithm that our research group has developed, the continuous *k*-means algorithm (Faber 1994).[2]

Connected scripts. As noted in section 2, our current system has only been fielded on scripts with discrete characters. It may prove difficult to apply our current normalization scheme to connected scripts such as Arabic and Devanagari: cramming a long, connected word into a square normalization space would probably distort it beyond the point of useful recognition. One possibility for dealing with connected scripts would be to distinguish them from discrete-character scripts on the basis of bounding box proportions, then develop a separate method for

---

[2]The continuous *k*-means algorithm is part of a patented application for improving both the processing speed and the appearance of color video displays. The application is commercially available for Macintosh computers under the names *Fast Eddie*, © 1992 and *Planet Color*, © 1993, by Paradigm Concepts, Inc., Santa Fe, NM. This software was developed by Vance Faber, Mark O. Mundt, Jeffrey S. Saltzman, and James M. White.

distinguishing among them. However, this would run counter to our general philosophy of automated learning.

A more attractive possibility is to replace our current normalization with one appropriate for both discrete and connected characters. For example, we could resize regions to a constant height, while adjusting width proportionately, thus preserving the original proportions of the region. In this case we would need to rework our current grouping and symbol matching algorithms: given regions of different sizes, Hamming and Euclidean distance would no longer apply. A moment-based representation (or another summary representation) could provide a solution.

Linking with language identification. As noted in the Introduction, script identification is tantamount to language identification for some scripts, but not for others. Once we have addressed the issues raised above, we plan to link our script identifier with appropriate language identification algorithms for multi-language scripts such as Roman. The most obvious approach would be to perform script-specific OCR followed by n-gram analysis. However, we would prefer to develop a new language recognition scheme based on the templates we use for script identification. This would have two advantages. First, it would be faster, as the symbols would have already been matched to templates. Second, it would avoid the paradox pointed out by Sibun and Spitz 1994: that OCR is most accurate when the language of a document is already known.

ACKNOWLEDGEMENTS

We thank Doug Muir for much useful feedback on our method, Calvin Hamilton for help with Khoros programming, and Wooyoung Choi and Tal Grossman for consulting on Korean and Hebrew fonts.

REFERENCES

Cash, G.L. & M. Hatamian (1987) Optical character recognition by the method of moments. *Computer Vision, Graphics, and Image Processing* 39: 291-310.

Church, K. (1986) Stress assignment in letter to sound rules for speech synthesis. *Proceedings of ICASSP 1986 (Tokyo)*, pp. 2423-6.

Faber, V. (1994) Clustering and the continuous *k*-means algorithm. *Los Alamos Science* 22: 138-49.

Rasure, J. & C. Williams (1991) An integrated visual language and software development environment. Journal of Visual Languages and Computing 2: 217-46.

Sibun, P. & A.L. Spitz (1994) Language determination: Natural language processing from scanned document images. *Proceedings of ANLP 1994.*

Spitz, A.L. (1994a) Text characterization by connected component transformations. In L.M. Vincent & T. Pavlidis (Eds.), *Document Recognition (SPIE Vol. 2181)*, pp. 97-105.

Spitz, A.L. (1994b) Script and language determination from document images. *Proceedings of the 3rd Annual Symposium on Document Analysis and Information Retrieval, April 1994 (Las Vegas, Nevada)*, pp. 229-35.