Automatic Script Identification from Images Using Cluster-based Templates

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

Computer Research, MS B265, Los Alamos National Laboratory, Los Alamos NM 87545
judithh@c3.lanl.gov, gelt@c3.lanl.gov, kelly@c3.lanl.gov, trt@c3.lanl.gov

Abstract

We describe a system that automatically identifies the script used in documents stored electronically in image form. The system can learn to distinguish any number of scripts. It develops a set of representative symbols (templates) for each script by clustering textual symbols from a set of training documents and representing each cluster by its centroid. "Textual symbols" include discrete characters in scripts such as Cyrillic, as well as adjoined characters, character fragments, and whole words in connected scripts such as Arabic. To identify a new document's script, the system compares a subset of symbols from the document to each script's templates, screening out rare or unreliable templates, and choosing the script whose templates provide the best match. Our current system, trained on thirteen scripts, correctly identifies all test documents except those printed in fonts that differ markedly from fonts in the training set.

1. Introduction

Script identification is a key part of the automatic processing of document images in an international environment. A document's script determines the correct OCR algorithm to use. Further processing, such as indexing or translation, depends on the language used in a document. Script identification is either tantamount to language identification (e.g., for Korean), or is a necessary first step. For example, once a document's script has been identified as Roman, one can OCR the image and use an *n*-gram algorithm that looks for frequently occurring character sequences from English, French, etc. [1], or scan the image for word shapes that correspond to such sequences [2].

The Fuji Xerox group has done extensive research on the topic of automatic script identification [2, 3]. Their approach combines automated and hands-on analysis of a training corpus to characterize each script. For example, Asian scripts (Chinese, Korean, Japanese) are distinguished from Roman by a uniform vertical distribution of upward concavities, and are distinguished from each other on the basis of character density. This approach requires a new hands-on analysis for each script.

In contrast, our script identifier automatically learns distinctions among an arbitrary number of scripts. It discovers frequent character or word shapes in each script by means of cluster analysis, then looks for instances of these in new documents. The cluster analysis identifies textual symbols in a representative set of training documents, clusters them, and calculates each cluster's centroid, or pixel-by-pixel average. This serves as a representative symbol, or *template*, for the cluster. To identify the script used in a new document, we compare a subset of its symbols to the templates for each script, and choose the script whose templates provide the best match.

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). Traditionally, one attempts to correct such phenomena in preprocessing (e.g., [4]). In contrast, we include them in the clustering process along with complete letters and numerals, large diacritics and punctuation marks, and whole words in connected scripts such as Arabic. We believe that this approach makes for a more flexible system, as one 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.

Our system currently distinguishes among Arabic, Armenian, Burmese, Chinese, Cyrillic, Devanagari, Ethiopic, Greek, Hebrew, Japanese, Korean, Roman, and Thai. The only documents it misclassifies are those printed in fonts that differ markedly from those in the training set. In any application based on our research, one should be able to prevent such errors by augmenting the training set.

2. Method

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

1. Assemble training and test sets of document images.
2. Find and rescale textual symbols in the training set.
3. Cluster similar symbols within each script. Make templates by calculating each cluster's centroid (average symbol). Downcast to bit. Eliminate minor clusters, and identify unreliable clusters.
4. Match a subset of symbols from a new image to the templates.

Most of this work was done within the framework of the Khoros™ image processing system [5].

2.1 Datasets

Our data consisted of 276 document images from 33 languages written in 13 scripts, including two scripts with connected characters (Arabic, Devanagari) and two non-alphabetic scripts (Chinese, Japanese). The images were scanned in from books, newspapers, magazines, and computer printouts. We divided them into three sets. The *training* set had ten images from each script, with at least two images from each source used. We included a variety of type styles for each script, such as original Chinese characters and simplified ones (PRC), and oblique and straight Armenian fonts. The *test* set had five images from each script, drawn from the same sources as the training set (e.g., an additional page from the same book). The *challenge* set had up to 16 images from each script, drawn from sources not used in the training and test sets, including novel fonts and languages. Table 1 summarizes the makeup of the data sets.

We describe below our treatment of several data issues:

Skew. We included images with line skew up to 10˚. Images scanned in from books were often skewed toward the center binding. We often cut-and-pasted several document portions to make a larger image, resulting in several different skews on a single page.

Color. All images were black-on-white, though we retained white-on-black portions in some challenge images.

Illustrations. We whited out illustrations in training and test images, but not challenge images.

Special characters. Most documents included numerals; none were handwritten. We whited out foreign characters in training and test images, but not challenge images.

2.2 Textual symbols

We used an 8-connected region growing algorithm to locate all textual symbols in the training set. We removed many non-textual symbols by filtering out regions containing fewer than 10 pixels, or whose bounding boxes were more than 80 pixels high. We retained large and long regions in order to avoid filtering out long Arabic and Devanagari words.

We rescaled each textual symbol to 30 X 30 pixels in order to equalize font (point) sizes between documents. For connected scripts, rescaling blurred individual word characteristics while preserving the overall Arabic or Devanagari look of the word; see Figure 1.

Figure 1. Rescaling Arabic and Devanagari words

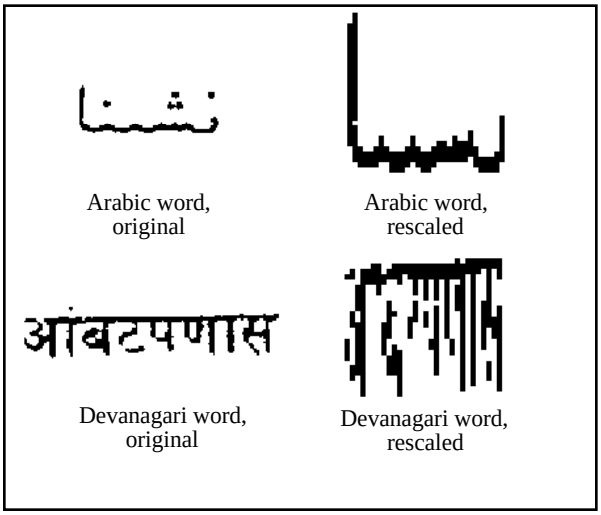

Table 1. Datasets

	Language (number of images)		
Script	**Training**	**Testing**	**Challenge**
Arabic	Arabic (4), Dari (2), Farsi (4)	Arabic (2), Dari (1), Farsi (2)	Arabic (4)
Armenian	Armenian (10)	Armenian (5)	Armenian (3)
Burmese	Burmese (10)	Burmese (5)	
Chinese	Chinese (10)	Chinese (5)	Chinese (7)
Cyrillic	Russian (8), Serbian (2)	Russian (4), Serbian (1)	Russian (3), Macedonian (1), Ukrainian (1)
Devanagari	Hindi (6), Marathi (2), Sanskrit (2)	Hindi (3), Marathi (1), Sanskrit (1)	Hindi (2)
Ethiopic	Amharic (8), Tigrinian (2)	Amharic (4), Tigrinian (1)	Amharic (5), Tigrinian (2)
Greek	Greek (10)	Greek (5)	Greek (6)
Hebrew	Hebrew (7), Yiddish (3)	Hebrew (3), Yiddish (2)	Hebrew (5)
Japanese	Japanese (10)	Japanese (5)	Japanese (3)
Korean	Korean (10)	Korean (5)	Korean (6)
Roman	English (2), French (2), German (2), Italian (2), Slovak (2)	English (1), French (1), German (1), Italian (1), Slovak (1)	Eng. (5), Gmn. (2), Gael. (1), Welsh (1), Hung. (1), It. (1), Pol. (1), Port. (1), Span. (2), Swed. (1)
Thai	Thai (10)	Thai (5)	Thai (4)

2.3 Templates

We used a hierarchical clustering algorithm to find similar symbols within each script. Each training symbol was examined in turn. If it differed markedly from all existing clusters (each defined by their first member) it was assigned to a new cluster; otherwise, it was added to the most similar cluster. The similarity metric used was Hamming distance: the number of pixels (black or white) that differed between two symbols. "Markedly different" was defined as having 250 or more differing pixels (out of 900).

Once symbols had been clustered, we calculated each cluster's centroid: the pixel-by-pixel average (a byte value) of all the symbols in the cluster. We then downcast the centroid to bit (black-and-white) by thresholding at a grayscale value of 128. This was the cluster's template.

We eliminated clusters with only one or two members in order to focus on the templates that were most likely to be useful, and to speed up the identification process. We then made a second pass through the training set in order to quantify the *reliability* of the remaining templates. For each training symbol, we found the overall best match from among all templates based on Hamming distance. As we did this we kept track, for each template, of the number of symbols matched to the template and the proportion of these that were from the correct script. For example, of the 479 training symbols best-matched to our third Cyrillic template, 468 were Cyrillic, giving a reliability figure of 98%.

Table 2 summarizes the number of templates made for each script. Figure 2 shows the two most frequent templates from each script with a reliability figure of at least 90%.

Table 2. Templates

Script	Original templates	% Eliminated	Final templates
Arabic	148	14	127
Armenian	195	21	155
Burmese	432	34	283
Chinese	1623	50	804
Cyrillic	329	32	223
Devanagari	674	30	470
Ethiopic	390	33	261
Greek	284	37	178
Hebrew	140	23	108
Japanese	719	49	367
Korean	339	18	279
Roman	337	33	226
Thai	463	28	334

2.4 Matching new symbols to the templates

Our classification algorithm accepted two parameters: *N*, the number of symbols to examine, and *R*, a reliability threshold. When processing a test document, we first identified and rescaled *N* textual symbols. For each of these symbols, we found the best match within each script (based on Hamming distance), saving the matching scores. Ignoring symbols whose *overall* best match (across all scripts) was to a template with reliability less than *R*, we calculated the mean matching score for each script. We picked the script with the best mean matching score.

Figure 2. The two most frequent 90% reliable templates in each script

3. Results

As shown in Table 3, with *N* over 75 and *R* over 50 all test images, and all but two or three challenge images, were correctly classified. All misclassified images in this region, and most outside of it, were printed in fonts markedly different from those in the training set. For example, a German image in the Fraktur font was misclassified as Ethiopic or Thai, a Spanish image in italics was misclassified as Armenian, and a Cyrillic image in a modern sans-serif font was misclassified as Roman or Greek. The correct classification of all other challenge images, including many with novel fonts and languages, showed that the system was able to generalize except in extreme cases.

This conclusion is buttressed by our experience with the system. In our previous round of training and testing, one Hebrew challenge image with a modern font was consistently misclassified as Thai. Adding a few images with modern fonts (distinct from the problematic font) to the training set corrected the misclassification.

4. Conclusion

Our system learns to accurately distinguish among a set of scripts that includes alphabetic and non-alphabetic scripts, discrete and connected scripts, and pairs or triplets of fairly similar scripts. Little preprocessing is required. The system can overcome 'noise' in the form of moderate skew, numerals, foreign characters, illustrations, and blurred or fragmented characters.

The next step in our research is to link this algorithm with appropriate language identification algorithms for multi-language scripts such as Roman. One approach would be to perform script-specific OCR followed by *n*-gram analysis. However, we would prefer to base language identification on the script identification templates. This would be faster, as the symbols would have already been matched to templates. It would also avoid the paradox pointed out in [2], that OCR is most accurate when the language of a document is already known.

REFERENCES

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

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

[3] Spitz, A.L. (1994) 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.

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

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

Acknowledgments

This work was performed under the auspices of the United States Department of Energy, contract W-7405-ENG-36; a patent has been filed. We thank the many individuals who helped us build our dataset, and Chris Brislawn, Jeff Kubina, and Doug Muir, for much useful input.

Table 3. Number of misclassified document images (test/challenge) out of 65 test, 68 challenge

	Value of R parameter (reliability threshold)									
No. of symbols	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
10	1 / 17	2 / 14	3 / 16	4 / 16	4 / 16	5 / 17	5 / 16	4 / 14	6 / 19	9 / 21
20	0 / 13	0 / 10	0 / 11	0 / 11	0 / 10	0 / 11	0 / 9	0 / 8	1 / 11	2 / 10
30	0 / 10	0 / 7	0 / 8	0 / 7	0 / 9	0 / 7	0 / 7	0 / 8	1 / 8	1 / 8
40	0 / 7	0 / 4	0 / 4	0 / 4	0 / 4	0 / 4	0 / 2	0 / 3	0 / 4	1 / 3
50	0 / 6	0 / 4	0 / 2	0 / 2	0 / 4	0 / 3	0 / 3	0 / 3	0 / 3	1 / 2
75	0 / 6	0 / 6	0 / 5	0 / 3	0 / 5	0 / 5	0 / 4	0 / 3	0 / 4	0 / 1
100	0 / 6	0 / 4	0 / 3	0 / 2	0 / 3	0 / 3	**0 / 3**	**0 / 2**	**0 / 2**	**0 / 2**
150	0 / 6	0 / 4	0 / 4	0 / 2	0 / 2	0 / 3	**0 / 2**	**0 / 2**	**0 / 2**	**0 / 2**
200	0 / 6	0 / 4	0 / 3	0 / 3	0 / 3	0 / 2	**0 / 2**	**0 / 2**	**0 / 2**	**0 / 2**
250	0 / 5	0 / 4	0 / 3	0 / 3	0 / 2	0 / 2	**0 / 2**	**0 / 2**	**0 / 2**	**0 / 2**
300	0 / 6	0 / 4	0 / 3	0 / 3	0 / 3	0 / 3	**0 / 2**	**0 / 2**	**0 / 2**	**0 / 2**
350	0 / 6	0 / 4	0 / 3	0 / 3	0 / 3	0 / 3	**0 / 2**	**0 / 3**	**0 / 2**	**0 / 2**
400	0 / 6	0 / 4	0 / 3	0 / 3	0 / 4	0 / 4	**0 / 3**	**0 / 2**	**0 / 2**	**0 / 2**
450	0 / 5	0 / 4	0 / 4	0 / 3	0 / 4	0 / 3	**0 / 3**	**0 / 3**	**0 / 2**	**0 / 2**
500	0 / 5	0 / 4	0 / 3	0 / 3	0 / 3	0 / 3	**0 / 2**	**0 / 3**	**0 / 2**	**0 / 2**