Combined Script and Page Orientation Estimation using the Tesseract OCR engine

Ranjith Unnikrishnan and Ray Smith
Google Inc.,
1600 Amphitheatre Pkwy, Mountain View, CA 94043
ranjith@alumni.cmu.edu, theraysmith@gmail.com

ABSTRACT

This paper proposes a simple but effective algorithm to estimate the script and dominant page orientation of the text contained in an image. A candidate set of shape classes for each script is generated using synthetically rendered text and used to train a fast shape classifier. At run time, the classifier is applied independently to connected components in the image for each possible orientation of the component, and the accumulated confidence scores are used to determine the best estimate of page orientation and script. Results demonstrate the effectiveness of the approach on a dataset of 1846 documents containing a diverse set of images in 14 scripts and any of four possible page orientations.

A C++ implementation of this work will be made available in a future release of the open-source Tesseract OCR engine [1].

Categories and Subject Descriptors

I.7.5 [**Document and Text Processing**]: Document Capture—*Optical Character Recognition (OCR)*

General Terms

Algorithms, Languages

Keywords

Script detection, Page orientation detection, Tesseract

1. INTRODUCTION

This paper focuses on the problem of estimating the script and dominant page orientation of printed text in an image. To accurately recognize the text in an image, optical character recognition (OCR) algorithms often utilize a great deal of prior knowledge, such as of the shapes of characters, list of words and the frequencies and patterns with which they occur. Much if not all of this knowledge is language-specific and does not generalize across scripts. This makes the language of the text contained in an image a crucial input parameter to specify when using an OCR algorithm.

Figure 1: Image samples of the 15 scripts detected using the proposed algorithm. (Latin is not shown. Fraktur is treated as an independent script. See text for details.)

Scripts form a natural appearance-based grouping of languages, and many languages share the same script. For example, the Russian, Bulgarian and Ukranian languages share the "Cyrillic" script. Most Indic scripts, such as Telugu, Kannada and Tamil, have only one language associated with them, whereas the Latin script is shared by at least 26 common languages. Thus a solution to the script detection problem either solves the language identification problem or reduces it to a smaller problem.

Independent of knowledge of the script, OCR algorithms often make the natural assumption that the text in the image being processed is upright. This need not always be the case. For instance, large tables and figures in books are frequently printed in landscape mode while the content of the book may be oriented in portrait mode. Or the book itself could be fed as input in an incorrect orientation. Such examples of mismatch between the input and the expectations of typical OCR algorithms are common, and lead inevitably to the OCR algorithm producing garbage output.

One brute-force solution would be to use a traditional OCR algorithm to process each input page once for each possible combination of orientation and language mode. However, this would be impractically slow, and also require a

MOCR '09, July 25, 2009 Barcelona, Spain

separate procedure to determine validity of the text produced as output in each orientation-language configuration.

This paper proposes a simple but effective approach to estimate script and dominant orientation not only with high accuracy, but also in far less time than it would take to process the image even once using a traditional OCR algorithm. The solution scales well enough to distinguish between 14 scripts - Latin, Cyrillic, Greek, Hebrew, Arabic, Chinese (or Han), Japanese, Korean, Thai, Devanagari, Kannada, Tamil, Telugu and Bengali - spanning more than 40 languages, and also distinguish between Fraktur and non-Fraktur fonts.

1.1 Related work

Past approaches to solve the script detection problem may be grouped to three broad categories. The first may be referred to as "global" or texture-based approaches. Algorithms in this category compute discriminative features on blocks of text using image filters to determine patterns that are unique to the language or the script. Chaudhury et. al [2] proposed using a frequency domain representation of projection profiles of horizontal text lines. Busch et. al [3] present an extensive evaluation of a broad number of texture features, including projection profiles, Gabor and wavelet features and gray-level co-occurrence matrices for detecting the script. This category of approaches has the drawbacks of requiring large and aligned homogeneous regions of text in one script, and of the features in question often being neither very discriminative nor reliable to compute in the presence of noisy or skewed text.

The second category of approaches may be referred to as "local" or connected-component based. These utilize shape and stroke characteristics of individual connected components. Hochberg et. al [4] proposed using script-specific templates by clustering frequently occurring character or word shapes. Spitz et. al [5] construct shape codes that capture the concavities of characters, and use them to first classify them as Latin-based or Han-based, and then within those categories using other shape-features. Ma et al [6] use Gabor-filters with a nearest-neighbor classifier to determine script and font-type at the word-level. Several hybrid variants of local and global approaches have also been suggested [7].

The third category of approaches may be referred to as "text-based". Algorithms in this category work by processing the entire page with a traditional OCR engine using one or more "pilot" language modes and then use a separate procedure to analyze the statistics of the (potentially inaccurate) output to guess what language the original image text may have been in. This category of approaches cleverly utilizes the fact that although shape classifiers are predictably wrong when evaluated on classes they are not trained on, the errors they make tend to be repeatable. Hence processing an image of Arabic text in a Han language mode will give garbage text, but with a characteristic frequency of output characters. The statistics of the output text can then be analyzed to estimate the script of the input image. However, such techniques tend not scale very well to a large number of scripts/languages, and our experiments in Section 5 show them to have lower accuracy than the approach proposed in this document.

2. APPROACH

Our proposed approach falls into the category of "local" approaches and operates by classifying individual connected components independently. This strategy gives it the compelling advantages of not requiring word segmentation or text-line finding as a pre-processing step and of being able to work on small input images.

The basic idea behind the proposed approach is simple. A shape classifier is trained on characters (classes) from all the scripts of interest. At run-time, the classifier is run independently on each connected component (CC) in the image and the process is repeated after rotating each CC into three other candidate orientations (90°, 180° and 270° from the input orientation). The algorithm keeps track of the estimated number of characters in each script for a given orientation, and the accumulated classifier confidence score across all candidate orientations. The estimate of page orientation is chosen as the one with the highest cumulative confidence score, and the estimate of script is chosen as the one with the highest number of characters in that script for the best orientation estimate.

The main difficulty with this strategy is the large total number of classes associated with all 14 scripts. The Han script alone has several thousands of characters in frequent use. In scripts such as Devanagari and Arabic, the shapes of letters can assume different forms depending on context, and unlike Latin, the letters can join to form single or multiple connected components. These properties of scripts combinatorially increase the total number of possible shapes, and since our approach requires associating a class (and script) to each connected component, the effective number of classes to train a shape classifier with can be impractically large. Once the set of possible shape classes has been identified for all the scripts, the problem then becomes that of how to choose a subset of these classes with which to train the shape classifier.

One possible approach to the problem of selecting representative classes for the scripts is through *discriminative* selection. This involves selecting shape classes in a script that have a high "distance" to classes from other scripts. This distance between any two shapes may be computed as a function of the classifier confidence when trained on one shape and evaluated on the other, or by directly embedding the shapes in an appropriate feature space.

We see two main problems with a discriminative approach to class selection. The first is that discriminative classes are not necessarily frequent. For example, suppose we choose the lower-case 'q' as a class for the Latin script as it does not share a shape similarity with any character from a different script. Given an image of text, the frequency with which the letter 'q' occurs will likely be small. Thus, given a strategy of classifying a randomly sampled set of connected components, the expected time to identify the script as Latin will be high. This limits the accuracy of the algorithm when the input image has little text.

The second problem arises from the observation that the performance of the overall algorithm depends not only on the accuracy of the classifier on the classes that are in the training set, but also on its behavior on shapes that the classifier has *not* been trained on. For example, assume the shape classifier is not trained on the shape of the character 't'. When given the shape of 't' as input, the ideal outcome is for the classifier to declare that the shape may be one of

any candidate script with equal probability. However this ideal outcome never occurs in practice, and the normal behavior for the would be to give the scripts unequal posterior probability. This bias naturally affects the estimate of script made by the algorithm overall.

So instead, we adopt an approach of *generative* class selection. This is done by ranking the classes in order of frequency and other statistics of interest, and then pruning this list by coverage. This approach is motivated by the following observations

1. The coverage graph of classes is often steep. This property is particularly true for scripts like Latin and Cyrillic, that share similarly shaped characters. Thus in practice, only a small number of shape classes need be chosen to represent such scripts.

2. Both of the previously mentioned problems with discriminative selection occur only in proportion to the frequency of occurrence of the classes that, respectively, are or are not in the training set. Choice of a shape class associated with frequently occurring characters in text reduces the expected time to identify its script. Similarly, the inability to recognize the shape of the letter 't', following the previous example, and the consequent assignment of poor posterior script probability estimates has an effect only in proportion to the natural frequency of occurrence of the letter 't' in text.

In Section 3, we detail the processes of identifying the shape classes associated with a given script and then generatively selecting a subset of these classes for training the Tesseract shape classifier. Section 4 explains the algorithm at run-time when processing an image of text. Section 5 then explains the dataset we use for testing the algorithm as well as an alternate approach that is also evaluated for comparison. We then conclude in Section 6 with some general observations and directions for future work.

3. TRAINING

The intent of the Training stage is to prepare a specially trained classifier, which in our case is the static shape classifier used by the Tesseract OCR engine [8]. It takes as input a set of text corpora in the scripts that we are interested in, and gives as output for each script a set of classes and shapes to train the classifier on. This stage may be divided into three steps - candidate class creation, generative selection and classifier training.

3.1 Candidate class creation

The goal of this first step in training is to exhaustively identify a set of image shape primitives and associated text that collectively represent any body of text in that script. The resulting collection of shape-text pairs will form a candidate set of classes from which a subset will be later chosen to train the shape classifier with in subsequent steps.

Our strategy for this step is to use natural text in each script, as obtained from a text corpus crawled from the web, by rendering the text and isolating the connected components associated with the text in the rendered page. The text for each component is known from the rendering procedure, and the shapes of the text are easily obtained using a connected component extraction algorithm [9]. This gives

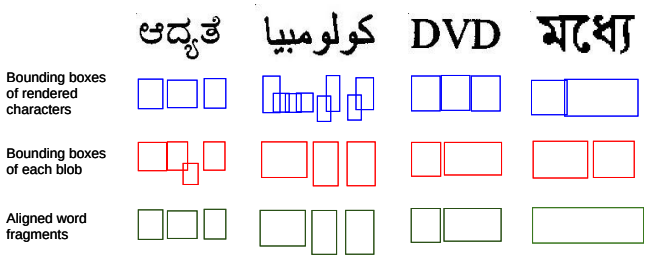

Figure 2: Examples of word fragment extraction for (left-to-right) Kannada, Arabic, Latin and Bengali, illustrating different cases of character overlap and resulting fragment generation. See text for details.

a set of shape-text pairs and allows us to obtain relevant statistics from them for use in the subsequent selection step.

In more detail, our preferred implementation generates the candidate set by the following procedure that is performed for each script of interest:

1. Render words from a text corpus in the script to an image with variable degradation in one or more supported fonts using a standard rendering engine. Our implementation uses the International Components for Unicode (ICU) layout engine which provides support for a variety of non-Latin scripts, including Devanagari and Arabic, that have complex rendering rules. The degradation consists of varying levels of morphological erosion/dilation and noise addition, and is done to train the classifiers to be robust.
   This step yields a set of bounding boxes in reading-order around each character in the image, along with the character associated with each box. Figure 2 shows bounding boxes in blue around each rendered character for example words in Kannada, English, Arabic and Bengali scripts.

2. Process the image to find its connected components and order their bounding boxes (colored red in Figure 2) in reading order. We use the existing Tesseract implementation to process the page in this step. The Tesseract pre-processor also groups together connected components based on a horizontal overlap criterion into nested lists of components termed *blobs*. This ensures, for instance, that the dot in the shape of lower-case 'i' is grouped together with the vertical stem of the 'i' and not potentially treated as a separate character. Figure 2 shows bounding boxes in red around each obtained blob for the example words.
   This use of Tesseract during training, as opposed to only during classification, has the benefit of yielding groups of connected components and shapes in a manner that matches what would be obtained at run-time.

3. Align the two sequences of bounding boxes around the characters and bounding boxes around the blobs to form a set of *word fragments*. This alignment can be done greedily by testing overlap between the bounding boxes of each blob and character and forming clusters. By rendering one word at a time, the fragment boundaries can be ensured to not cross word boundaries, thus further simplifying the problem of alignment.

Figure 3: Portion of synthetically rendered training image for the Kannada script. Bounding boxes around word fragments selected for this script are shown colored in green.

Figure 2 illustrates some possible scenarios for word fragment (shown as green boxes) generation in different scripts. The Kannada word has its second character consisting of two connected components that do not significantly overlap, whereas the Arabic characters combine to form connected components. While Latin characters do not normally combine, the example above happens to have two characters that overlap due to the chosen font and degradation level in the rendering. The second character in the Bengali word consists of two disjoint connected components, but its left component overlaps with the previous character resulting in a single word fragment consisting of two characters. Thus a fragment may consist of one or more characters and one or more blobs (or connected components).

4. Reject a fragment if it has an aspect ratio that exceeds a certain threshold, or has other indicators that it may be easily confused with other characters or non-text shapes. If a fragment is not rejected, keep track of the number of times it is seen in the input text corpus, the number of characters and connected components it represents, the text associated with it and the script of the text, the properties of the font used to render the fragment, and any other relevant statistics.

The resulting set of word fragments forms the candidate set of shape classes for the particular script. This step is then repeated for all scripts and for each font that supports the script to yield a set of fragments covering all scripts of interest. This marks the end of the candidate class identification stage and makes way for the next step of generative class selection.

3.2 Generative selection

In this step, a representative subset of classes is selected from the previously formed candidate set. In our implementation, this is done by first ranking the fragments of each script in decreasing order of frequency of occurrence. If two fragments are within a margin of frequency, they are re-ranked in increasing order of the number of characters they represent, then by the connected components they contain and so on for any other statistics of interest. This re-ranking step gives a mild preference to fragments that have simple non-intricate shapes, as they tend to be more amenable to encoding and representation by the shape classifier.

The ordered set for *each* script s of interest is then pruned to the top x_s% by total number of occurrences (coverage). The value of x_s controls the number of classes to learn for script s and may be varied independently per script to trade off processing time with classification error. Our implementation starts by setting the value of x_s to an equal low value, say 10%, for all scripts and then increasing the value for each script independently in steps until the classification error across all scripts for the trained fonts no longer decreases. Section 4 details the steps performed at classification time. The end result of this step is a set of chosen word fragments for each script.

3.3 Shape classifier training

The final step is to use the selected word fragments for training the shape classifier. Our implementation uses Tesseract's static character classifier which we outline in Section 4 and is also detailed in [8].

The input required to the train the classifier consists of an image of text, along with a set of image bounding box coordinates and associated character text for each character to train on. We generate this data using the list of word fragments pruned in the generative selection step. In the candidate class creation step (Section 3.1) we keep track of the fragments contained in each word that is encountered in the text corpus. Words are then concatenated in decreasing order of frequency for each script-font pair until they collectively include all the fragments selected for the script. The selected words are then rendered in the font and the fragment bounding boxes are extracted using knowledge of the statistics (number of blobs and characters) for each fragment contained in the word. The bounding box coordinates of fragments that are included in the selected set of classes for the script are written to a file and later used along with the rendered image for training the character classifier.

The use of frequent previously seen words from the corpus in this manner has two benefits. First, it allows the character classifier to pick up side-information like character height in the context of frequently seen words. Second, the resulting box-image file pairs form very compact training data. Figure 3 shows an example of the training image and fragment boxes corresponding to the Kannada script for the Kedage font.

Table 1 lists the composition of the training classes used by script. The shape classifier is trained on a total number of 1808 classes and on one font per script. To not be confused by the occasional page containing large tables of numbers, the commonly used Indo-Arabic numerals are separately added to the set of training classes, and they are listed under the script name of "Common" in the table.

4. CLASSIFICATION

Before detailing the script and page orientation estimation algorithm at run-time, we give a brief overview of the internal feature representation and the form of the feature classifier used in Tesseract.

Tesseract's static character classifier uses two types of internal feature representation for each word fragment and two stages of classification [8]. The features used in the trained models are 4-dimensional ((x, y) position, direction and length) segments of a polygonized outline, and the features of the unknown are 3-dimensional, obtained by breaking each segment into multiple unit-length fragments.

The first stage of classification, the class pruner, produces a short-list of classes using a technique similar to the idea of forgiving hashing [11]. The second stage of character clas-

Script name	#classes	Coverage (x_s)
Arabic	200	60%
Bengali	101	30%
Common	10	100%
Cyrillic	28	90%
Devanagari	223	10%
Greek	7	40%
Han	578	25%
Hangul	543	30%
Hebrew	10	70%
Hiragana	28	50%
Kannada	69	60%
Katakana	68	50%
Latin	30	60%
Tamil	15	20%
Telugu	84	60%
Thai	14	60%

Table 1: Composition of the 1808 classes in the training set across scripts: Coverage values x_s reflect the cumulative frequency of occurrence of the selected classes over the set of fragments after filtering (Step 2a in Section 4).

sification consists of matching the polygonal segments from training to those obtained from the outline of the unknown shape. Two segments are declared to match if they are proximal with respect to (x, y) position and angle. The distance between the two shapes is computed as a weighted average of the distance segments in the prototype shape are from the unknown shape, and vice versa. Although this second step can be computationally expensive, it is only performed on the classes in the short-list returned by the class pruner. The class with minimum shape distance from the unknown shape is noted, and the script of the text associated with the class is taken as the best estimate of the script of that shape. Fraktur is treated as a separate script, although it is detected using knowledge of the font properties associated with the trained class instead of the script of the associated text.

Given an image of a page as input, the procedure at runtime is as follows:

1. Binarize the image and segment it into connected components. Group connected components having significant horizontal overlap into blobs. Note that no line-finding algorithm or connected-component ordering procedure is required at this stage. However, depending on the type of input expected, an algorithm to find non-text regions [9, 10] may be required as a pre-processing step prior to this in order to remove blobs incorrectly detected in those regions from consideration. This however is a common requirement for OCR in general and is not a limitation of the proposed algorithm.

2. For each blob from a randomly selected N-sized subset of all the blobs in the image:

   (a) If the blob has an aspect ratio that exceeds a certain threshold, has height or width outside an acceptable range of values to consist of valid text characters, or has other indicators that it may be easily confused with other characters or non-text shapes, reject it and continue.

   (b) Classify the blob and find the most likely script it belongs to. If the confidence score for the best estimate of this script is low, or is within some margin of the confidence of the next best estimate of the script, reject this blob and continue.

   (c) Accumulate the confidence score associated with the best estimate of the script to a total.

   (d) If the font properties of the estimated class indicate that the unknown shape was rendered with a Fraktur font, increment the count of the "Fraktur" script by one. Otherwise look up the script of the estimated class, and increment the count for the script by 1.

   (e) Repeat 2a-2d after rotating the connected component in the three other possible orientations (90°, 180° and 270° from the input orientation).
   The end result is four sets, one for each examined orientation, of counts for each script of interest along with an accumulated confidence score for that orientation.

3. Choose the orientation with the high total confidence score as the best estimate of the page orientation.

4. Choose the script that has the highest count for the estimated orientation.

Some scripts, such as Korean and Japanese are strictly pseudo-scripts whose text consists of a combination of characters belonging to other "true" scripts. For example, Japanese exists as a combination of text in Katakana, Hiragana and Han scripts, and Korean exists as a combination of Hangul and Han.

We address the problem of identifying such pseudo-scripts by using fractional counts. In the event that the estimated script for an unknown shape is Han, the fact that the true script text may also be Korean or Japanese is taken into account by not only incrementing the count of the Han script by 1, but also incrementing the count of the Japanese and Korean scripts but by a smaller fraction. The optimal value of these fractions is estimated from analyzing the natural frequency of occurrence of Han symbols in Japanese and Korean from the text corpus of the two latter scripts. In our implementation, the weight of Han characters for the estimation of Japanese and Korean scripts are chosen as 0.2 and 0.6 respectively.

5. EVALUATION AND EXPERIMENTAL RESULTS

We evaluate the proposed algorithm on a dataset of 1846 multi-page documents obtained from scanning books in any of the 15 scripts (including Fraktur) in roughly equal proportions. Ground-truth for each document is available as a dominant orientation and a list of one or more scripts that the document contains. The algorithm is run on 10 randomly sampled pages of the document, and for a maximum of 250 blobs in each page. The confidence scores for orientation and script estimates are averaged across pages. This ensures that the estimate of the script is not biased by

Ground-truth script \ Estimated script	ara	ben	cyr	dev	frk	gre	han	heb	jpn	kan	kor	lat	tam	tel	tha
ara	**100.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ben	0.00	**100.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
cyr	0.00	0.00	**100.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
dev	0.00	0.00	0.00	**100.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
frk	0.00	0.00	0.00	0.00	**100.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
gre	0.00	0.00	0.00	0.00	0.00	**100.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
han	3.14	0.00	0.00	0.00	0.00	0.00	**92.15**	0.00	4.71	0.00	0.00	0.00	0.00	0.00	0.00
heb	2.01	0.00	0.00	0.00	0.00	0.00	0.00	**97.99**	0.00	0.00	0.00	0.00	0.00	0.00	0.00
jpn	0.00	0.00	1.03	0.00	0.00	0.00	4.12	0.00	**94.85**	0.00	0.00	0.00	0.00	0.00	0.00
kan	2.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**97.50**	0.00	0.00	0.00	0.00	0.00
kor	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.12	0.00	**96.88**	0.00	0.00	0.00	0.00
lat	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**100.00**	0.00	0.00	0.00
tam	0.95	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**99.05**	0.00	0.00
tel	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**100.00**	0.00
tha	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.01	0.00	0.00	0.00	0.00	0.00	**98.99**

Table 2: Confusion matrix of script detection results on the 1846 document dataset: Values are of accuracy and are in units of percentage. Rows correspond to ground-truth script names and columns correspond to the estimated script names given by the codes: ara=Arabic, ben=Bengali, cyr=Cyrillic, dev=Devanagari, frk=Fraktur, gre=Greek, han=Han, heb=Hebrew, jpn=Japanese, kan=Kannada, kor=Korean, lat=Latin, tam=Tamil, tel=Telugu, tha=Thai.

pages containing prefaces or author forewords with dense text in, say, English (Latin) even though the document may predominantly be in a different script.

Script and page orientation detection are problems of categorical classification, and so the error metric of choice is the modified indicator function, which we define as 1 if the best estimate of script is not in the list of the ground-truth scripts, and 0 otherwise. Since our approach gives as output a list of scripts ordered by confidence score, one may consider additional metrics, both discrete and continuous in nature. For instance, a "top two error" metric may be defined as 1 if neither of the two scripts with the highest confidence scores are in the list of ground-truth scripts. For brevity, this paper only reports the errors from modified indicator function.

Over the 1846 document dataset, our experiments recorded an error rate of **0.2%** in orientation and an error rate of **1.84%** in script identification. We make use of no class priors, and all candidate scripts and orientations are treated identically and considered equally likely. Table 2 shows the confusion matrix of ground-truth scripts and estimated scripts.

The time to process a page varies on the content of the page, and is often comparable to the time taken by the thresholding algorithm used in Tesseract. We have observed the time taken by the thresholder to be 0.3-1.2 seconds while the classifier, being time bounded by the maximum number of blobs it is allowed to process in a page, takes between 0.5-0.6 seconds.

Comparison to a text-based language ID algorithm:

We also compared the proposed algorithm to an alternate text-based approach. As outlined earlier in Section 1.1, the competing approach works by processing the whole page using an OCR engine trained in a pilot language and analyzing the statistics of the garbled output text. Our implementation of this approach processed each image with an OCR engine twice - first in a mode to recognize Latin characters and the second time in a mode to recognize Han. The unigram statistics of the output text were analyzed in each case to determine the likelihood of the text being Latin-like, or some other family of scripts.

Since this category of approaches was not originally intended to handle non-upright orientation, we evaluated it against the proposed algorithm on a smaller set of multi-page documents that all had upright orientation. We found the text-based approach to have an error rate of about 7.52% while the proposed approach had a lower error rate of 2.11%. The proposed approach was also about 6-10 times faster, owing largely to it not requiring any input image to be processed in entirety.

6. CONCLUSIONS

This paper proposed a simple yet effective algorithm to combine script and page orientation detection using the Tesseract shape classifier. There were many observations and design choices made that enabled the proposed approach to work well. One observation was that coverage distribution of shape classes in many scripts is steep, which allows a generative class selection scheme to work reasonably well using a small number of training classes. The design choices of using a "local" approach operating at the level of individual connected components and of interleaving the blob rotation and shape classification operations makes the overall algorithm efficient.

However, there are still several failure cases that need to be addressed. As may also be observed from the confusion matrix of Table 2, a significant number of classification errors are due to the Japanese script being mistaken as the Han script, or vice versa. This is a common source of errors for script detection algorithms in general and is largely due to the two scripts having many symbols in common.

Other failure cases are when documents contain degraded or handwritten text, or have unusual images or line drawings that are not removed from consideration in the pre-

processing step, or have scripts in fonts that have not been trained on. Many of these sources of error are shared with the more general OCR problem and are the subject of ongoing work.

7. ACKNOWLEDGMENTS

The authors thank Dar-Shyang Lee for devising and implementing the text-based language identification algorithm that this work is evaluated against, as well as for his useful suggestions and feedback.

8. REFERENCES

[1] Tesseract open source OCR engine. `http://code.google.com/p/tesseract-ocr`.

[2] S. Chaudhury, R. Sheth: Trainable Script Identification Strategies for Indian Languages, *Proc. 5th IEEE Intl. Conf. on Document Analysis and Recognition (ICDAR)*, pp. 657-680, 1999.

[3] A. Busch, W. Boles, S. Sridharan: Texture for Script Identification, *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*, 27 (11), pp. 1720-1732, 2005.

[4] J. Hochberg, P. Kelly, T. Thomas, L. Kerns: Automatic Script Identification From Document Images Using Cluster-Based Templates, *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, pp. 176-181, 1997.

[5] A. L. Spitz: Determination of the Script and Language Content of Document Images, *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*, pp. 235-245, 1997.

[6] H. Ma, D. Doermann: Gabor filter based multi-class classifier for scanned document images, *Proc. 7th IEEE Intl. Conf. on Document Analysis and Recognition (ICDAR)*, pp. 968-972, 2003.

[7] L. J. Zhou, Y. Lu, C. L. Tan: Bangla/English Script Identification Based on Analysis of Connected Component Profiles, *7th IAPR Workshop on Document Analysis Systems (DAS)*, pp. 243-254, 2006.

[8] R. Smith: An overview of the Tesseract OCR Engine, *Proc. 9th IEEE Intl. Conf. on Document Analysis and Recognition (ICDAR)*, pp. 629-633, 2007.

[9] Leptonica image processing and analysis library. `http://www.leptonica.com`.

[10] R. Smith: Hybrid Page Layout via Tab-stop Detection, *Proc. 10th IEEE Intl. Conf. on Document Analysis and Recognition (ICDAR)*, 2009.

[11] S. Baluja and M. Covell: Learning Forgiving Hash Functions: Algorithms and Large Scale Tests, *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.