Language segmentation for Optical Character Recognition using Self Organizing Maps

Kuzman Ganchev

Abstract

Modern optical character recognition (OCR) systems perform optimally on single-font monolingual texts, and have lower performance on bilingual and multilingual texts. For many OCR tasks it is necessary to accurately recognize characters from bilingual texts such as dictionaries or grammar books. We present a novel approach to segmenting bilingual text, easily extensible to more than two languages. Our approach uses self organizing maps to distinguish between characters of different languages allowing OCR to be performed on each part separately.

1 Introduction

Modern optical character recognition (OCR) systems perform optimally on single-font monolingual texts, especially when they have been trained on a font very similar to the one they need to recognize. Performance on bilingual texts, however is not nearly as good, especially when the two languages that occur in the text have similar characters. The reason for this is that dictionaries and language models are usually trained on a per-language basis. Sometimes the OCR program will assume that there are multiple fonts for the same language and make poor guesses about which characters are in which language. Despite these difficulties, there are real lines of motivation for performing bilingual OCR. For example is the rapid development of translation systems requires large amounts of training data. There are many languages around the world for which collections of texts as well as dictionaries and grammar books are readily available in printed form, but not in electronic form. For the task of rapidly developing a translation system, optical character recognition may be the only viable solution for obtaining training text.

This paper focuses on a sub-topic of bilingual OCR – namely deciding which characters are in which language. Once this has been done, monolingual OCR can be performed on each of the sections of text and the original document can be subsequently reconstructed. In order to perform this division, we use self organizing maps (SOMs) to distinguish between characters of one language and characters of the other. SOMs are a mechanism for sorting high dimensional vectors into a two dimensional grid, in such a way that similar vectors are sorted into grid locations near each other. A more detailed description of SOMs and how we use them is provided in Section 3.

Our results show that using the approach outlined in this paper, we can correctly determine the language of 98.9% of the characters in Uzbek-English dictionary text. Our experiments as well as these results are described in Section 4.

2 Related Work

2.1 Self Organizing Maps

Kohonen (1990) provides a concise introduction to self organizing maps in general, considering their applications since their conception. He describes how a SOM might work, and notes that it may be necessary to preprocess the information somehow: "it would often be absurd to use primary signal elements, such as . . . pixels of an image, for the components of [a SOM] directly"(Kohonen, 1990). Kohonen presents his SOMs in the context of "Vector Quantization" – the assigning of quantum values to vectors; similar to what we want to do, if the image is the vector, then the quantum representation would be the character that produced it or in the case of language segmentation, the language that that character is in. In this context he also talks about a "codebook" of vectors that are essentially the definition of the quantum to vector relationships. He presents three algorithms for "Learning Vector Quantization" called LVQ1, LVQ2 and LVQ3.

He and Ganchev (2003) use SOMs and neural networks for simple object recognition on a mobile robot, with limited success. The images taken from a camera

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 109–115
Computer Science Department, Swarthmore College

mounted on a Khepera robot moving in an enclosed space are sorted into a large SOM, after which a neural network is used to associate labels to images.

2.2 Optical Character Recognition

Berman and Fateman (Berman and Fateman, 1994) describe an OCR system for mathematical texts. The main application of their work is to recognize printed tables of integrals, or mathematical proofs. The algorithm they describe requires that images for every variation of every font be learned separately by the program. They use the Hausdorff asymmetric distance function which is defined as the largest Euclidean distance from any "on" pixel in the source image to the nearest "on" pixel in the destination image. One interesting feature of this metric is that in one direction it treats characters with missing "on" pixels as a perfect match, while in the other direction it treats characters with extra "on" pixels as a perfect match. This allows it to deal with broken glyphs and over-connected characters. Unfortunately the authors do not give any numerical results, and the system has to be retrained from scratch for even a slightly different font or font size. Since a maximum of distances is taken for the measure; this approach might also be very sensitive to dust. For example, a spec of dust in the middle of an "O" would make it as far from the model "O" as if it was completely filled in.

3 Abstract Implementation

In order to investigate methods for using SOMs to divide an image into languages, we developed different versions of a system for classifying characters into two sets based on the language which generated them. The first subsection gives an overview of the architecture of this system. Subsequent subsections describe each of the system components in more detail. Subsection 3.2 describes the Gnu Optical Character Recognition program, Subsection 3.3 gives an overview of self organizing maps, and Subsection 3.4 describes the system components that we implemented.

3.1 Architectural Overview

Figure 1 shows a graphical representation of the system architecture. In the figure, the dotted lines represent flow of information, while the solid lines represent the flow of control. Given a source image, we use the Gnu Optical Character Recognition program (GOCR) to find the boxes that contain characters of text. We then wrote a program to extract data vectors corresponding to characters of text from the image using the box descriptions provided by GOCR. Once this training data has been extracted, it is used to train a self organizing map (SOM). We used the SOM_PAK (Kohonen et al., 1995) toolkit for all the SOM tasks. Finally, we use the trained SOM to distinguish between characters of different character sets.

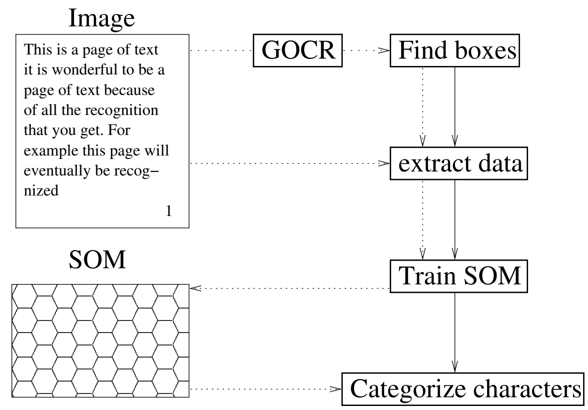


Figure 1: Architecture: The dotted lines represent flow of information, while the solid lines represent the flow of control. GOCR is used to find the boxes that contain characters, the data vectors corresponding to those boxes is extracted and used to train the SOM. Finally the trained SOM is used to categorize the characters into two languages.

3.2 GOCR

GOCR, developed by Joerg Schulenburg and Bruno Barberi Gnecco, is a free optical character recognition system under continuing development. GOCR reads in an image file, and scans it for boxes containing a single character of text. Each box ideally contains a single character and is the smallest rectangular area with sides parallel to the x and y axes. In practice boxes may contain more than one character (See figure 3) but in our experience they rarely contain less than a character. After a list of boxes has been generated, GOCR removes dust and pictures that do not need to be recognized, attempts to detect a rotation angle and lines of text within the scanned image, and attempts to correct glued boxes and broken characters. Finally GOCR runs an OCR engine on the sorted boxes. As an example of its performance GOCR program gives the following output for the image shown in Figure 2; GOCR also added two extraneous new lines between each pair of lines of text, probably because of the high resolution of the source image. The "_" characters represent characters that GOCR was not able to recognize. Figure 3 shows the boxes and lines of text that GOCR has found within the image.

```
itcan go. Butthe"3O's
pas_. The eleven mus
ch_sen as the great o
periodbe_een l94O
```


Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 109–115
Computer Science Department, Swarthmore College

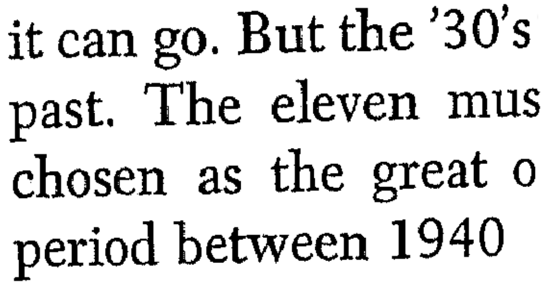


Figure 2: A sample of scanned text used for optical character recognition. Note that the image is very hight resolution, but imperfect and skewed.

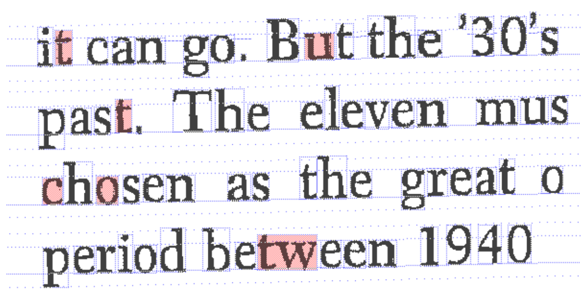


Figure 3: The same scanned piece of text, after processing by GOCR. The blue lines represent the lines of the text as well as the box boundaries. Characters for which GOCR does not have a high confidence are shaded in red. Note that the "tw" shaded in red on the bottom line is in fact a single box. GOCR is not able to recover from this error.

3.3 Self Organizing Maps

Conceptually, a self organizing map (SOM) is a structure for sorting vectors into a grid in such a way that vectors close together on the grid are similar, and vectors farther away are more dissimilar. After training, each grid location contains a vector that represents the ideal vector at that location called a "model vector". Typically, SOM grids are hexagonal, as shown in Figure 4. This allows each cell to be adjacent to six other cells, rather than four with a rectangular grid.

Before the SOM can be trained, it is initialized by setting all its model vectors to random values. Training is done as follows: for each data vector, find the model vector closest to it, and modify that vector and vectors nearby (in the grid) to make make them closer to the data vector. We used Euclidean distance to determine how close a data vector was to the model vector. We opted for this approach since it is the default for SOM_PAK; an other option would have been to use cosine similarity as the distance metric. One way to make one vector closer to another is to use a weighted average of the two. The relative weights of the vectors are determined by a learning rate parameter.

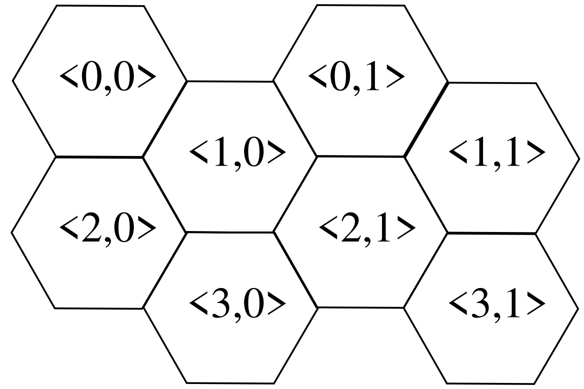


Figure 4: A hexagonal SOM topology. This is a 4×2 SOM – it has four rows and two columns.

There are two stages of training. During the first short stage the learning rate is set high and the neighborhood of the vectors is large, so that the SOM roughly but quickly organizes the vectors into groups. During the longer second stage of training a lower learning rate and smaller neighborhood are set so that local arrangements can be made and the model vectors approach ideal characters.

After the training phase is complete the same vectors used for training can be sorted into locations on the grid. The location and difference from the nearest model for each vector is recorded for use as described in Subsection 3.4. We used SOM_PAK (Kohonen et al., 1995) as the SOM implementation for our experiments.

3.4 Custom Components

Our system relies on a number of small components other than SOM_PAK and GOCR. Section 3.4.1 describes how we convert the images described by the character boxes generated by GOCR, into data vectors used to train the SOM. Section 3.4.2 describes how we use the trained SOM to segment bilingual text.

3.4.1 Converting Images to Vectors

In using a SOM to sort images, we need to first convert the images into some vector representation, since SOMs are designed to work with vectors. We do this by using "1" to designate a white pixel and "0" to designate a black pixel, and then creating a vector that has one dimension per pixel. For example a 3×3 image with black top and bottom rows and a white middle row would be rep-

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 109–115
Computer Science Department, Swarthmore College

resented as “(0,0,0,1,1,1,0,0,0)”. This provides a mechanism to convert an image to a vector and vice-versa, but the sizes of the vectors depend on the size of the image, so a “.” character, might be a 3×3 image (corresponding to a nine dimensional vector) while a “M”character might be much larger. This complicates matters because a SOM only works with vectors of the same size. We describe two techniques we used to deal with this in Section 4.

3.4.2 After Training

Once we have a SOM trained in one language, we map characters from an unknown language into that SOM. Based on the distance from the character vector to the nearest model vector in the SOM, we decide whether the character belongs to the same language on which the SOM has been trained or to some other language.

4 Experimental Results

This section describes the empirical evaluation we performed on our system. Subsection 4.1 describes our initial approach for converting character boxes to vectors – placing the box at the top left-hand corner of the image – and describes the problems that produces. Subsection 4.2 describes a refinement to this approach – scaling the characters – and provides comparative results to those obtained in Subsection 4.1. Subsection 4.3 describes how the algorithm performs on hand-selected training data for the task of segmenting dictionary text. Finally, Subsection 4.4 describes its performance when trained using unedited “dirty” training data.

4.1 Top Left Corner

The first approach we use to convert character boxes into vectors is to place the character box in the top left-hand corner of a white image that is the same size for all the characters. We create a white image, then copy the pixels of the character into the top left hand corner of this image. Figure 5 illustrates this. We can then convert the image into a vector as described in Section 3.4.1. Since the image into which we are copying the characters is the same for each character, all the vectors will be of the same size and we can use them to train the SOM and to categorize the characters.

This initial approach has a few limitations. Firstly, for even a slightly different font or different font-size, the SOM might have to be retrained from scratch. This is because a character in a different size or font would show a significant mismatch of the model vectors pixels, even when the fonts are identical. Secondly, when we started to test the system to find unlikely characters we found that this approach gave a considerable advantage to smaller characters over larger ones. Figure 6 shows some results that demonstrate this. The grey areas are parts of the image that were not part of a box. The color of each character is determined by how close the vector representation of the character is to the nearest model vector in the trained SOM, and hence how confident we are that they are in the language the SOM was trained on. Blue characters are close to the SOM model vector, while red characters are distant.

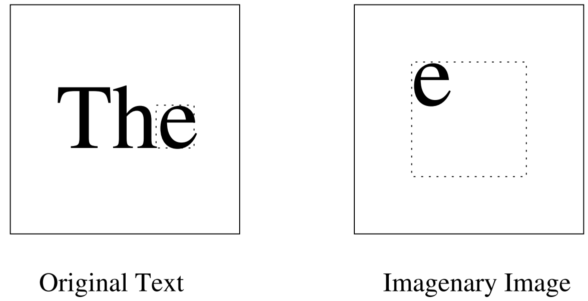


Figure 5: Initial “top left corner” method of converting character boxes to vectors. On the left is a segment of text with a box (dotted line) drawn around a character boundary, on the right is the image that will be used to generate the vector for the character.

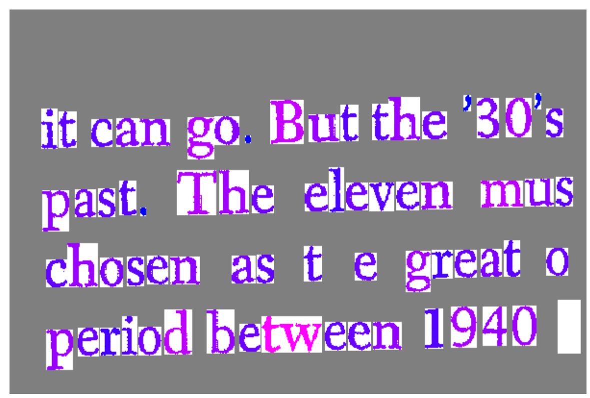


Figure 6: Distance from the nearest matching model vector for the characters in the text presented in Figure 2. Blue characters are close to the language the SOM was trained on, while red characters are classified outside that language. The ideal result would be to color all characters characters, except for the box that contains “tw”, since this is not an English character (it is two).

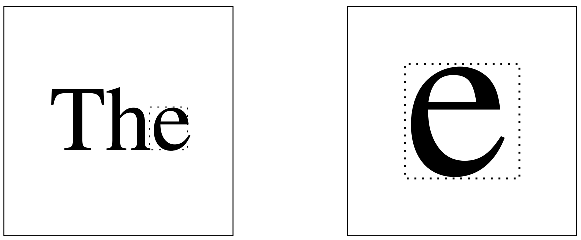
We note that the small characters are closer to the models than the larger ones. This is because of the way we convert the images to vectors; any two small characters share a lot of “white space” and can differ only in the top left hand corner of the image we use to generate the vectors, since we pad the rest of the space with white pixels for both small characters. So for example, comparing a “.” to a “;” the difference ”,” might occupy five pixels,

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 109–115
Computer Science Department, Swarthmore College

while two capital “M”s might have a larger difference. While both “.” and “;” occur in English, small characters in another character set are more likely to be closer to small Latin characters than two scans of identical characters.

4.2 Scaling Characters

To overcome the problems the “top left corner” technique incurs with different sizes of the same font and of giving an advantage to small characters, we tried scaling the characters when converting them to a vector representation. Again, we create intermediate images of the same size for all the characters, but this time we scale the character to this image. The intermediate image is then converted to a vector as above. Figure 7 illustrates the conversion.



Original Text Imagenary Image

Figure 7: The “scaling” method of converting character boxes to vectors. On the left is a segment of text with a box (dotted line) drawn around a character boundary, on the right is the intermediate image with the character scaled and placed in the top left hand corner.

This method produced much better results. Figure 8 shows a sample of text color-coded in the same way as Figure 6. With the exception of the “m”s the boxes containing a single character in the main font are colored blue, as we would like. The italicized characters and the box containing the “th” bigram are correctly classified as different than the rest of the text. Section 5 describes possible improvements to this approach to deal with misclassified character like the “m”. The following experiment applies this approach to bilingual text.

4.3 Hand-Selected Regions

To test our system on the task of bilingual text segmentation, we attempted to segment text from the Hippocrene Uzbek-English English-Uzbek dictionary (Khakimov, 1994) into Cyrillic and Latin characters. We train the SOM using regions of a dictionary selected by hand to contain English text, and then use the trained SOM to estimate the language of unknown characters. Figure 9 shows a sample of Latin and Cyrillic as they occur in the dictionary, as well as the part selected as Latin text. The

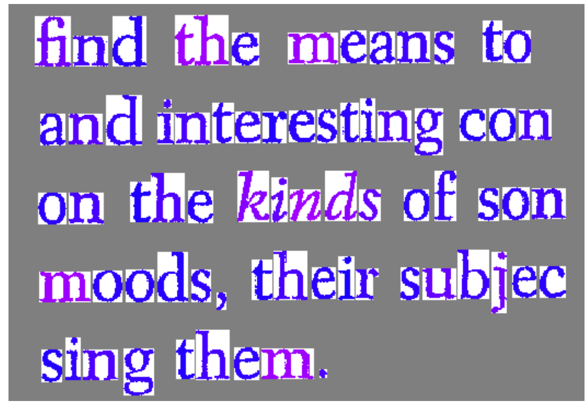

find the means to
and interesting con
on the *kinds* of son
moods, their subjec
sing them.

Figure 8: Distance from the nearest matching model vector using the scaling approach on English text. Blue characters are close to the nearest SOM model vector, red characters are far from all SOM model vectors. Note that the italicized characters are more pink than the rest of the text, as expected.

box in the figure is a part selected as English. Note that the word “goalkeeper” was not selected in this process to save time. The word will be classified as English by the trained system. As a rough estimate of how much human time this initial selection entailed, selecting English potions from 40 pages of dictionary text took roughly fifteen minutes.

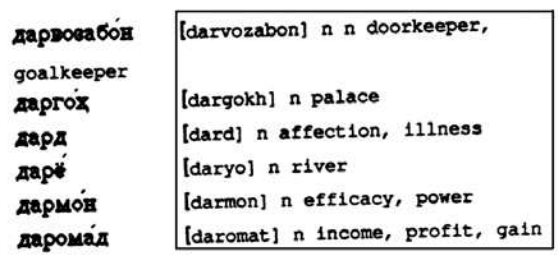

дарвозабо́н [darvozabon] n n doorkeeper,
goalkeeper
даргох [dargokh] n palace
дард [dard] n affection, illness
дарё [daryo] n river
дармо́н [darmon] n efficacy, power
даромад [daromat] n income, profit, gain

Figure 9: Part of a page from the Uzbek-English dictionary used for the experiment described in Section 4.3. The part of the image enclosed by the box has been selected as Latin text, and so can be used to train the SOM.

It is important to note that the Cyrillic characters are in a bold font while the Latin characters are not, which may help performance. It is probably not unjustified to assume that this will often be the case, since changing the font also makes it easier for a human to distinguish between languages at a glance and a typesetter might do this to aid the human reader.

Figure 10 shows part of a dictionary page color-coded by the system. Letters closer to blue are more likely to be Latin, while those closer to red are more likely to be Cyrillic. We see that most of the English text is correctly

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 109–115
Computer Science Department, Swarthmore College

classified – the word "goalkeeper" is most interesting, since it was not used during training, and was correctly classified except for the "pe" bigram for which GOCR did not find the bounding box correctly. We also see that on a word by word basis, the system works pretty well – the Cyrillic words are more red than the Latin words, but there are a number of places where we make mistakes; several Latin characters are very pink, and several Cyrillic characters are very blue.

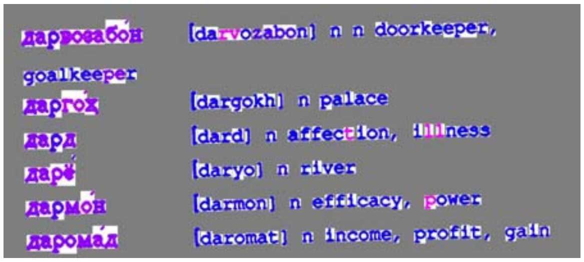


Figure 10: Distance from the nearest matching model vector for dictionary text. Blue characters are classified as Latin, red characters are classified as Cyrillic.

To obtain quantitative results for this experiment, we hand-selected all the English text in a dictionary page, and saved only this in an image. Similarly we placed all the Uzbek text from three dictionary pages into one image. This gave us 576 characters of English and 312 characters of Uzbek. We then used our system to classify all the characters in each image. The results are summarized in Table 1. The system made a correct guess of 98.9% of the time.

Language	Correct	Percentage
English	566 / 576	98.3%
Uzbek	312 / 312	100%
both	878 / 888	98.9%

Table 1: The performance of our system with hand-selected training data on the task of segmenting dictionary text.

4.4 Pages of a Dictionary

The results from the previous experiment are encouraging, but we wanted to investigate whether the human effort of segmenting some pages of text was really necessary. To investigate this we repeated the above experiment, but instead of training the system on hand-selected parts dictionary pages, we trained on entire pages of the dictionary. The idea behind this is to assume that since most of the characters on the page are of one language (English for the portion of the dictionary we used), it may be acceptable to pretend the rest of the characters are noise. Unfortunately, the results this provides are not nearly as good as those obtained in the previous experiments. Figure 11 illustrates the results for the same part of the page shown in Figure 10.

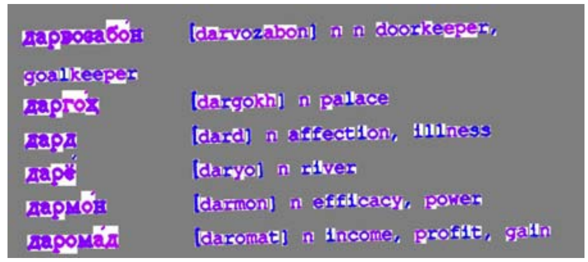


Figure 11: Distance from the nearest matching model vector for dictionary text, when using unedited pages of the dictionary. Blue characters are classified as Latin, red characters are classified as Cyrillic.

Quantitative results were obtained as above, after the system was trained on unsegmented pages of the dictionary. Overall the system makes the correct estimation 72.2% of the time. As a comparison, always choosing English would have given a performance of 64.7%, so this is not an impressive result at all. The details for each language are shown in Table 2.

Language	Correct	Percentage
English	339 / 576	58.9%
Uzbek	302 / 312	96.8%
both	641 / 888	72.2%

Table 2: The performance of our system trained on unedited pages of the dictionary.

5 Conclusions and Future directions

We presented a novel approach for using self organizing maps to segment bilingual text into its component languages, with applications to OCR. We tried this approach on pages from an Uzbek-English dictionary, and found that we were able to identify the correct language for 100% of the Uzbek characters and 98.3% of the English characters. With this success in mind, there are a number of limitations to the approach presented here, and we discuss these in Section 5.1. Section 5.2 concludes the paper with a discussion of possible future work.

5.1 Problems with this Approach

There are a number of limitations to the method presented in this paper. In order to get the results we obtained, we needed to manually select English text on which the SOM could be trained. While this did not take a lot of time, it is a cost that needs to be incurred for each new book on which OCR is to be performed. The method we describe

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 109–115
Computer Science Department, Swarthmore College

is also very computationally intensive and requires a lot of space; training the system took several hours on the available hardware and the training data occupied about 115 MB of disk space (compared to less than 2.5 MB for the images). We also do not use character position information in any way, so we may guess that a character that is part of a long word is of a different language than the rest of the word. Making the assumption that there is one language per word would probably considerably improve our results.

5.2 Future Work

Investigating the limitations mentioned above would be the first line of future work. One part of this would be to investigate how much training data is really necessary. Perhaps it is enough for a human to segment one page of training data instead of forty. If a few pages of training data are sufficient this would also alleviate a lot of the space requirements we currently have. We also do not investigate the possibility of training the SOM for a smaller number of iterations. It would be interesting to find out how this affects performance.

Extending the method to use position information would also be a line of future work. The one language per word assumption would not be difficult to implement and may improve result – especially when we only have scarce training data. Further position information could also be learned on the fly. For example, in our dictionary all the characters in the left two-thirds of the page were English. Another easy extension of this work would be to adapt it to deal with more than two languages.

A more difficult future work (suggested by one of the anonymous reviewers) would be to combine our approach with a language model approach. For example, if we perform OCR on each word in the image assuming it is in one language, the resulting text would match the language model best if we guessed the correct language. We could repeat this for each possible language before making a final decision. Combining this approach with ours would be helpful, since it could be used to provide training data with no human involvement, while our approach would deal with words that do not match the language model, such as the Latinization of the Cyrillic words in our dictionary.

References

Benjamin P. Berman and Richard J. Fateman. 1994. Optical character recognition for typeset mathematics. In *Proceedings of the international symposium on Symbolic and algebraic computation*, pages 348–353. ACM Press.

Kamran Khakimov. 1994. Hippocrene Books.

Kohonen, Hynninen, Kangras, and Laaksonen. 1995. The self organizing map program package.

Teuvo Kohonen. 1990. The self-organizing map. *Proceedings of the IEEE*, 78(9).

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 109–115
Computer Science Department, Swarthmore College