TJHSST Senior Research Project
Devlopment of a Generic Font OCR
First Quarter Research Paper
2007-2008

Nathan Harmata

November 2, 2007

Abstract

OCR (Optical Character Recognition) is a very practical field of Computer Science. Since the late 1980's, researchers have been developing systems to identify text from non electronic text sources, like pictures or papers. The use of OCR systems has spanned from making books in Braille to sorting mail by zip code by United States Post Office.

Keywords: OCR, Image Processing

1 Introduction

The goal of this project is to create an application that can read text from electronic picture files. One of the main focuses will be developing a generic way to recognize different fonts, rather than hardcoding in definitions for specific fonts. Although OCR is by no means a "new" field, it has still yet to be fully explored. There are very few OCR applications readily available to the public, and even the ones that are free of charge are lacking in performance and consistancy.

For now, an application that can read and recognize "simple" pictures, i.e. ones with minimal headers and only text, will be developed. If that

is successful, more advanced techniques to handle headers and "background noise" will be used. This project might also explore the field of handwriting recognition.

2 Background

OCR systems have been around since the late 1980's. Still, they are not widely available or used by the public. The results from a review of the free ones on the Linux operating system [1]are not very promising. Although most of them had measured accuracies above 94 percent, that is not good enough. The one commercial product tested, Aspire OCR, was accurate only 91.5 percent of the time. The most likely industry standard, Tesseract, is also one of the oldest OCR systems. The review measured it to have an accuracy rate of 99 percent. Development on it started in 1985 and it is still used as the OCR engine for Ocropus, Google's textual analysis application. It is unlikely that this project will be able to achieve similar success, but the goal is to be on par with the current OCR options.

3 Procedures

The current version works as follows:

The application accepts a picture file as input. The input file must be a png image of text that is Courier font and size 24 bold. The picture is first boxed to remove excess whitespace along the border. Then, the program finds the locations of horizontal lines on whitespace inside the new image and pairs them so that paragraphs and the spaces between paragraphs become separated. A similar method is used to parse paragrahs into lines. Spaces between words are handled by having the minimum space size for Courier font size 24 bold hardcoded in. Lines are broken into words, which in turn are broken into letters. The idea of this is to then have individual pictures of each letter, which can be interpreted as a graph of pixels.

Each character is then compared to a database so that the best match can be determined. The database was generated by creating an image containing all the possible characters (uppercase letters, lowercase letters, and punctuation) and then "telling" the program what the correct match is. The

program went through each character in the input and broke the final graph of pixels into four quadrants.

The number of non-text (i.e. whitespace) pixels in each quadrant was counted. Using these counts, a database of characters and their corresponding quadrant counts was created.

To find the best match for a given image, the same counts are generated and the relative pixel coverage for each quadrant was determined. The relative coverage for quadrant one is:

$$R_1 = \frac{C_1}{C_1 + C_2 + C_3 + C_4} \tag{1}$$

This is used to handle text of different sizes, since the relative coverages should be more or less the same.

3.1 Testing

1. A user-friendly interface for viewing the pictures of parsed words has already been developed. This allows for manual visual detection of errors in letter parsing.

2. An experiment was conducted to determine the effectiveness of relative pixel coverages. An error rate of 0 is expected for size 24 fonts, but there may be major errors in the analysis of text of different sizes. See the "Results" section.

3. Once more work is done on developing the database of general letter definitions, as described above, a more automated and effective method of testing will have to be employ. One idea is to write a script that creates picture inputs from randomly generated combinations of letters and then compares the results from the OCR application with the actual text. Another idea is to develop a way to test the accuracy of the OCR system in different areas, like how well it performs on individual letters versus complete sentences.

4. The current method of making direct pixel comparisons has encountered several problems. First off, in the Courier font, certain capital letters "elide" together. By this, it is meant that there is no vertical space between adjacent letters. One example is the combination of "M" and "X" to form "MX". Thus, the program interprets "MX" as a character that is the combination of "M" and "X," which of course is not

going to yield a direct match with the cache. An algorithm to discern between elisions and actual characters will have to be developed. An example of an elision:

The following computer languages, algorithms and programs are being used.

3.2 Software

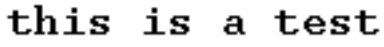
1. Java is used for picture input and output.

3.3 Algorithms/Programs

1. KolourPaint is being used to make picture files for input and to precisely view pictures.

4 Results and Conclusions

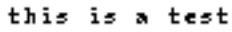
An experiment was conducted to determine how effective using relative pixel coverages is for analyzing data of different text sizes. The results may also give insight into how text is resized by scaling. Fourteen images were generated of the same text in Courier bold font but with different size, varying from size 6 to size 32, like this one:



Including spaces, each input has fourteen characters. Using the size 24 cache, each image was analyzed. For each image, the number of characters correctly read was recorded. These are the results:

Input Size	Number Right
6	1
8	2
10	3
12	3
14	3
16	4
18	9
20	9
22	9
24	14
26	14
28	14
30	14
32	14

As expected, the analysis of the size 24 data was 100% correct. The other results, however, were fairly suprising. Before performing the experiment, I did realize that my program would not be accurate on inputs of text not the same size as the cache, but the results were much more extreme than I expected. For the input of small text size, like size 6, the error rate was very high. One explanation for this is that really small text tends to be "blurry," like below:



Still, this does not explain why the size 16 input was only read 29% correctly. However, the results for the inputs larger than the cache may provide an answer. The fact that all the larger inputs were read correctly suggests that it is better to have a smaller cache. Perhaps resizing text works by scaling the image up, but does not work in reverse. This would certainly explain why all the inputs smaller than the cache yielded poor results.

These results suggest that a cache should be generated using images of small text.

References

[1] A Review of Free Optical Character Recognition Software *ground-state.ca/ocr*