An Elementary Optical Character Recognition System

Jason Aubert, Will Detlor, Jesper Leung

December 2015

Abstract

This paper goes over an elementary implementation of a complete optical character recognition system, capable of recognizing handwritten digits from 0 to 9, and uppercase letters from A to Z. Prior to recognition, every image that is inputted into the system is thinned and scaled, with the option of processing multiple characters also available. Once scaled, each individual character is then put through the recognition system, which consists of the use of a statistical method where the ratio of black pixels to white pixels in certain zones of the image is calculated, and used to generate a feature vector to describe the input character. This feature vector is later compared to a large number of other feature vectors from test images in a database and the feature vector with the lowest Euclidean distance with the input image will then be used to recognize the input character.

1 Introduction

An optical character recognition system (OCR), is characterized by a conversion from images of handwritten or printed text into data that can be later used for processing by a computer system. OCR systems are typically used to process printed texts for applications such as digital storage, and electronic modifications. OCR systems can also be found in other applications, such as the translation of handwritten languages, as well as in text-to-speech implementations.

Through the use of various functions, this optical character recognition system is able to recognize images of handwritten digits from 0 to 9, as well as uppercase letters from A to Z. After processing, the system will then output the digit or letter that it believes to most closely represent the image that was inputted. This process is explored more in depth in the following section.

This paper will then explore the results of the implemented method, through the use of 75 test images per digits to test the system's accuracy, which will then be used to explore the limitations of the statistical method for feature extraction that is used in this OCR system.

Future considerations for improving the OCR system will then be explored in the penultimate section of this paper, where suggestions for future developments are considered and discussed, and where areas of improvement within the current iteration of the system can be found.

2 OCR System

The OCR system that has been developed involves many steps to go from the initial character input to the character recognition output. The system was written in the Python programming language and implements the pillow library to assist with image handling.

2.1 Binary Image Conversion

To begin, a character is imported into the system by the user entering the file path of the image into the program. The image is imported as a gray-scale image, which the system converts to a black and white image. The system converts the image to black and white since our implementations of a thinning algorithm requires each pixel to be either white or black. Additionally, this is also a requirement for the feature vector since the feature vector is calculated by the amount of black pixels to the total amount of pixels in each zone. As such, not converting an image to black and white will cause an error. To convert an image to black and white, the system implements a function that iterates through each pixel in the image and if the pixel value is 255 (white) then the pixel is unchanged, otherwise the pixel is changed to 0 (black).

2.2 Image Segmentation

After the character has been imported by the system correctly and converted to a black and white image, the system checks to see how many charac-

ters are contained within the image. The algorithm implemented works as follows:

1. The algorithm begins by iterating through each pixel in the image until a non-white pixel is found.

2. Once a non-white pixel is found, the algorithm recursively searches each pixel around the found non-white pixel, adding all non-white pixels to a new image and keeping track of non-white pixels in a data structure.

3. The algorithm will continue to recursively search the neighbours of found non-white pixels until there are no more non-white pixels to be found. Thus, once this process is complete a new image with a single character is returned and the pixels found are kept track of.

4. The algorithm continues iterating through each pixel in the image from where it left off, ignoring non-white pixels that have already been found. This is done in order to ensure only pixels that are not part of characters already found are searched.

5. The algorithm finishes when all pixels have been searched, and a Python list of all the images of characters found are returned.

2.3 Scaling

After identifying all of the characters in an image is complete, the system crops and scales each image to a uniform size. Although scaling to a uniform size may cause a character to potentially be stretched or compressed in some cases, the benefits outweigh this risk by improving the consistency of the statistical recognition methods (outlined later on) used by this system. Specifically, the scaling algorithm performs as follows:

1. The algorithm begins by iterating through all of the pixels to identify the non-white pixel with the min/max x/y values.

2. These min/max x/y values are used to crop the excess white space in the image, ensuring that the character fills the entirety of the image.

3. Afterwards, the image is re-sized to a given width/height and returned.

2.4 Thinning

Once the image has been scaled the system pads the image with one layer of zeros around the border so that the thinning algorithm will perform correctly. The proposed system implements the ZS algorithm which was developed by T.Y. Zhang and C.Y. Suen to thin the scaled image. The ZS thinning algorithm consists of two sub-iterations each containing four rules that are checked. On the first sub-iteration the algorithm deletes the pixels on the south boundary and the east boundary, along with the north-west corner points. On the second sub-iteration the algorithm deletes the pixels on the north boundary and the west boundary along with the south-east corner points. The algorithm continues to run until the inputted character is thinned into a one-pixel skeleton of the original image. For more information on the ZS algorithm refer to reference 4.

2.5 Feature Extraction

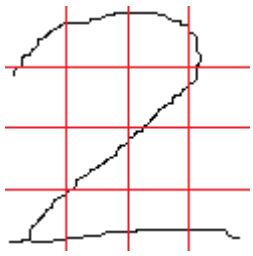

Figure 2.5.1: An example showing how the zoning method is implemented on an image after all previous processes

The implemented OCR system utilizes the zoning method for feature extraction. After the image has been scaled and thinned, the system divides the image into sixteen zones as shown in Figure 2.5.1. Each

zone represents a component of the feature vector. For every zone, the ratio of black pixels in the zone to total number of pixels in the zone is found which represents one component in the feature vector. The implemented OCR system utilizes a 4x4 grid to create a feature vector with 16 components.

$$Z_i = \frac{P_{black}}{P_{total}}$$

Figure 2.5.2: Pixel percentage in each zone: number of black pixels divide by the total number of pixels in the the zone.

2.6 Character Recognition

The feature vector determined from the inputted image is then compared to those stored in the database. The database has been preloaded with five feature vectors for each number and capital letter. For each character, 0 to 9 and A-Z in uppercase, the system is trained with four handwritten characters and one typed character. For the uppercase letters the typed characters uses the font Calibri and for the numbers the typed characters uses font Arial. The system compares the inputted feature vector to each vector in the database to determine which vector in the database has the smallest Euclidean distance compared to inputted feature vector. The vector with the smallest Euclidean distance is determined to be the match and the value associated with that vector is outputted as the recognized character by the system.

[0.0375, 0.0333, 0.0333, 0.0204,
0.0043, 0.0000, 0.0366, 0.0001,
0.0000, 0.0398, 0.0135, 0.0000,
0.0645, 0.0333, 0.0333, 0.0289]

Figure 2.6.1: Sample feature vector

2.7 System Learning

In order to improve the recognition rate over time, the system features a user feedback system to update the database based on the results of its findings. After the system has displayed the value that corresponds to the vector that was chosen with the smallest Euclidean distance, the user is then asked if the character outputted was the correct character. If the inputted character and the recognized character do not match, the user is then able to tell the system what character the inputted image should have been recognized as. The system then adds the inputted feature vector as a new entry in the database as the character that the user specified was the correct value. Thus, in the future similar characters that were not recognized will be able to be identified by the system.

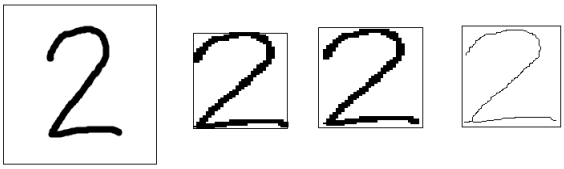

Figure 2.7.1: Example of the OCR system; Image 1. The original input image; Image 2. The scaled image; Image 3. The scaled and padded image; Image 4. The thinned image ready to be divided into the feature vector

3 Results

The proposed recognition system was tested with 75 sample images for each number from 0 to 9, for a total of 750 test images (sample test in figure 3.1). These test images were compared against two separate sets of data within the database, one of which only contained numbers while the other data set contained both numbers from 0 to 9 and capital letters from A to Z in order to compare the effect on the recognition rate. Afterwards, five test images of each letter are tested against the numbers and letters data set. The results for each data set are examined below.

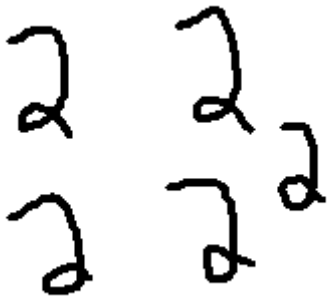

Figure 3.1: Sample image tested with the proposed OCR system

3.1 Numbers Only

In this situation 75 test images of each number were run through the implemented OCR system, where the database consisted only of sample numbers (no letters). Overall, this data had a recognition rate of 69.33%. Figure 3.1.1 displays the recognition rates for each number derived from this test.

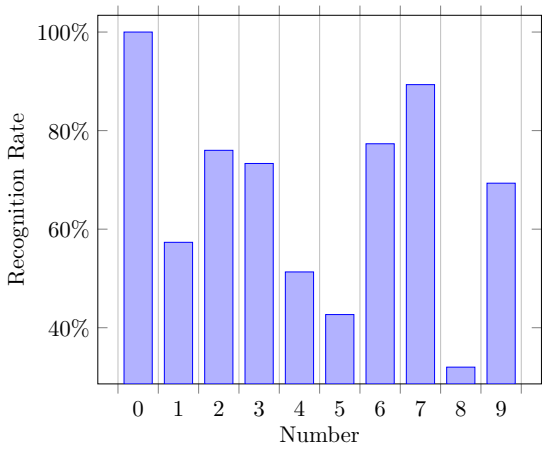

Figure 3.1.1: Graph with recognition rates of numbers against a database containing only numbers

There is a large variation in the resulting recognition rates for each number. Some numbers achieve a very high recognition rate, such as zero and seven, which have a 100% and 89% recognition rate respectively. Other numbers have a very low recognition rate such as five and eight, which are at 43% and 32%.

The reason for this variation in recognition rate could be due to several distinct factors. First of all, numbers may fail to be recognized due to their similarity to other numbers. For example, eight has a low recognition rate as it is consistently mistaken for other numbers such as zero, three, six, and nine. This could be in part due to the fact that eight is very similar to these numbers, as for each of them the difference with eight is only one extra stroke.

Additionally, another factor that could have influenced the results was the fact that the characters in the database and the test characters were drawn by different individuals. As such, differences in writing styles for specific numbers may have had an effect on their recognition rates. Figure 3.1.2 displays once such difference in writing style for the number five.

Figure 3.1.2: Two sample images, one taken from the database and one used as a test image displaying the difference in writing style

3.2 Numbers & Letters

This sample consisted of testing 75 images of each number against a database containing both numbers and letters. The recognition rates are displayed below in figure 3.2.1.

In comparison to the database with numbers only recognition rates for all numbers decreased as anticipated, although some more than others. Overall, the

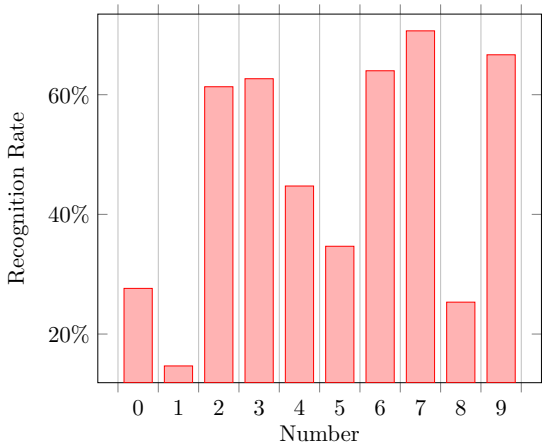

Figure 3.2.1: Graph with recognition rates of numbers against a database containing numbers and letters.

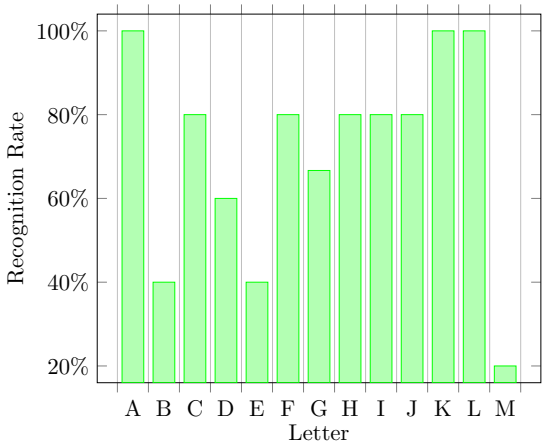

Figure 3.3.1: Graph with recognition rates of letters from A to M against a database containing numbers and letters.

recognition rate decreased from 69.33% to 47.24%. Some numbers such as five and nine had their respective recognition rates decrease very little, while others such as zero and one saw drastic decreases. Several rationals are presented below as to the cause of this variation.

One rational as to why the zero's and one's recognition rates decreased significantly with the introduction of the letters into the database was that both zero and one are very similar to one of the letters. The results back this proposition up, as many zeros were misinterpreted as 'O's and many one's were misrepresented as 'T', where both feature vectors strongly correlate. In fact, over 90% of zero's were identified as either a zero or 'O'. Thus, in future developments finding ways to distinguish between highly similar symbols such as zero and 'O' would be valuable in improving the recognition rate.

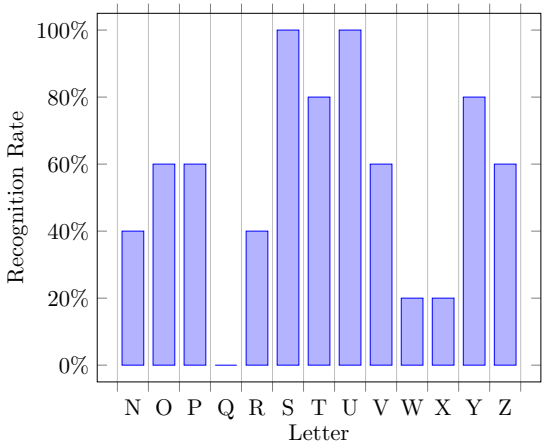

Figure 3.3.2: Graph with recognition rates of letters from N to Z against a database containing numbers and letters.

3.3 Letters

This sample consisted of testing five images of each letter against a database containing both numbers and letters. Overall, a recognition rate of 63.33% was achieved. The recognition rates for each letter are displayed below in figure 3.3.1 and 3.3.2.

As was the case with numbers, some letters have higher recognition rates than others. In this situation, letters such as 'Q' and 'M' had low recognition rates while others such as 'K' and 'L' had high recognition rates. Again, this could be attributed to the similarity of 'Q' and 'M' to other characters in the database. Many 'Q's that were run through the OCR

system were misidentified as 'O' or zero, which overall appear very similar to a 'Q'. This was also the case with 'M', which was often misinterpreted as 'H', 'A', or 'W'. The inverse was also found with characters such as 'K' and 'L', which have fewer characters that are highly correlated. This finding also identifies the need to develop a strategy that can distinguish between characters with small differences (i.e. the line that distinguishes a 'O' and a 'Q'.

4 Extra features

4.1 Thinning

In addition to the ZS algorithm implemented for thinning in this system, an elementary implementation of the 20 rule algorithm developed by Maher Ahmed and Rabab Ward was also included in an attempt to improve the speed of the system. The 20 rule algorithm is an iterative algorithm, which at each iteration checks each pixel to see if the pixel is on the edge of the image, and if the pixel is on the edge then the pixel is deleted. Once every pixel in the image has been checked, another iteration is ran until only a 1-pixel width skeleton of the image is left. All 20 rules are implemented in the program correctly, but when the image has a width of two pixels then both pixels are deleted since both pixels are on an edge in the same iteration. This occurs due to an issue with the systems implementation of the 20-rule algorithm. As such, there are discontinuities in some inputted images which is why the system still implements the ZS algorithm instead of the 20-rule algorithm.

Figure 4.1.1: Original image and the corresponding thinned image using the 20-rule thinning algorithm

4.2 Filtering

Although not a part of the recognition system, filtering is demonstrated in this OCR system as a proof of concept. To achieve this, the image is first padded to allow for the convolution operation to work fully. The convolution operation involves the use of a 3x3 matrix known as the "kernel", which is used to modify the image pixel by pixel. The convolution operation takes each pixel in the target image and multiplies the surrounding pixels of the target image with those in the kernel, in order to form a sum that is then used to replace the target pixel in the image, as demonstrated in Figure 4.2.3.

i_0	i_1	i_2
i_3	i_4	i_5
i_6	i_7	i_8

Figure 4.2.1: A 9x9 matrix representing the image

k_0	k_1	k_2
k_3	k_4	k_5
k_6	k_7	k_8

Figure 4.2.2: A 9x9 matrix representing the kernel

$$i_4 = (k_0 * i_0) + (k_1 * i_1) + ... + (k_8 * i_8)$$

Figure 4.2.3: Convolution equation on pixel i_4

As can be seen above, a convolution operation on a single pixel involves its 8 surrounding pixels. As such, padding is necessary in order to properly apply convolution to edge pixels. While padding the image with pixel values of 0 will still allow for convolution of the entire image to work, padding the image with its edge values, while more complicated of an operation, nets a better result. This padding operation is seen below in Figure 4.2.4

Depending on the size of the image, the difference in padding techniques cannot be very well seen, as it only affects the pixels at the edge of the input image.

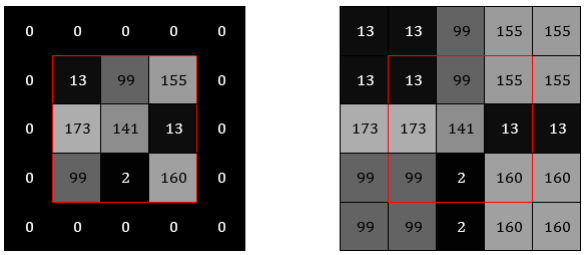

Figure 4.2.4: Illustration of padding with 0s and with edge values

Because of this, the difference in padding will only affect the convoluted pixels of the outside edge of the image, meaning this effect decreases at larger image sizes. However, at smaller image sizes, the difference is much more discernible when the edges of the image are much more prominent. While the difference in padding techniques makes little difference on relatively sizable images, its consistency at smaller image sizes means it is much more consistent overall.

5 Future Considerations

5.1 Increasing scope

Future improvements to this OCR system should include all letters, in both upper and lowercase along with numbers. This improvement would be hard to implement as different characters such as 'I' and '1' look very similar along with 'o','O' and '0'. To be able to handle characters that are very similar, the program would have to extract more features from the original image.

5.2 Limitations of Python

The OCR system could also be written in a different language as Python is a simple language. By switching to a different language the system would be able to run faster and the database could be loaded with more test vectors without slowing down the response time of the system. As well, Python has recursion depth limits which affect the segmentation algorithm on larger sized images. Thus, in this case another language would be beneficial as it would be able to handle large images sizes effectively.

5.3 Improving recognition

As demonstrated in the "Results" section of this paper, the zoning method that is used for feature extraction is not enough on its own to provide consistent results across all fronts. In order to improve the overall accuracy of the OCR system, additional features would need to be implemented. One such feature is the detection of strokes. By decomposing characters into strokes, it would be possible to discern between characters based on how the entire character is pieced together in handwriting, which in turn would provide more areas of comparison when recognizing characters. However, due to the nature of this OCR system implementation and the time allotted for its creation, this feature is left unimplemented, and only one feature is extracted and used for conducting comparisons and recognitions.

5.4 Implementing grammar

With the addition of characters in the database as explored in section 3.2, overall recognition of both numbers and letters drops significantly. This is due to the fact that certain characters are very similar in the way they are written and perceived by the program. For example, two and 'Z' can be mistaken very easily with the current amalgamation of zoning and statistical methods of feature extraction. As such, implementing a way for the system to make an educated decision on which symbol to recognize an image as would be useful.

One such way would be to implement grammar rules into the recognition system, which would take into account the surrounding characters that have already been recognized to make an educated guess as to whether the character in question is a digit or a letter. For example, without an implementation of grammar, it would be easy to see how the OCR system outlined in this paper would have trouble discerning between "balloon" and "ba1100n". In this case, the implementation of grammar would allow the

system to make the decision that the "1100" it is recognizing should be recognized as "lloo" instead since it would be able to determine that a word is most likely being inputted, rather than a combination of letters and numbers. Through the implementation of grammar, it would be possible for an OCR system to recognize numbers and characters depending on context, and would therefore improve recognition across both sets of characters.

6 Conclusion

An elementary optical character recognition system is presented in this paper. The system involves many functions that takes an image of a character, either 0-9 or capital letters A-Z, and returns a feature vector of sixteen elements using the zoning method. The implemented system has a recognition rate of 69.33% for comparing numbers to a database with only number vectors, 47.24% for comparing numbers to a database with number and capital letter vectors, and 63.33% for comparing capital letters to a database with capital letter and number vectors.

7 References

1. Powell, Victor. "Image Kernels Explained Visually." Explained Visually. N.p., n.d. Web. 05 Dec. 2015.

2. Ahmed, M., and R. Ward. "A Rotation Invariant Rule-based Thinning Algorithm for Character Recognition." IEEE Transactions on Pattern Analysis and Machine Intelligence IEEE Trans. Pattern Anal. Machine Intell. 24.12 (2002): 1672-678. Web.

3. Lundh, Fredrik, and Alex Clark. "Pillow." Pillow — Pillow (PIL Fork) 2.6.1 Documentation. Secret Labs AB, n.d. Web. 06 Dec. 2015.

4. Zhang, T. Y., and C. Y. Suen. "A Fast Parallel Algorithm for Thinning Digital Patterns." Communications of the ACM Commun. ACM 27.3 (1984): 236-39. Print.