Digit recognition system using image processing techniques in Java

Troy Nechanicky (150405860), Sarah Johnston (150139570)

|  |  |  |
| --- | --- | --- |
| A R T I C L E I N F O  *Prepared for:*  Prepared for Dr. Maher Ahmed  CP467 Image Processing  Fall 2018, 2018-12-03  *Keywords:*  Symbol recognition  Digit recognition  Image Processing |  | A B S T R A C T  Over four months, a digit recognition system has been developed with the purpose to categorize handwritten digits. Numerous stages of the program are influenced and adapted from teaching material, online resources, and original ideas. Each stage of the program was implemented independently. These stages include image filtering, identifying connected regions, converting images to black and white, scaling, thinning, and identifying unknowns against fixed feature vectors for the digits between 0 and 9. The system has on average 82% classification accuracy and requires future work by developers before the program can be used in the practical world. |

1. Introduction

Computers are continuing to change human civilization by automating everyday tasks for businesses and consumers. However, the job of reading handwritten digits is still ag growing area of research. Over the course of four months, a digit recognition system was created in Java using image processing techniques learned in CP467 at Wilfrid Laurier University. The system begins by taking a grayscale image as input. Filters may be applied to the image to sharpen or blur it, and the image is converted to black and white. Then, the system identifies individual digits, scales each of them to the same size and thins them. Finally, the program compares each digit to known feature vectors and identifies them. The goal of this system is to accurately identify handwritten digits in efficient time.

1. Developed System

The digit recognition system is implemented in Java. The algorithms used in the program originate from various sources including online resources and materials taught in class by Dr. Ahmer Ahmed.

* 1. Image Filters

The first stage of the system is applying filters to the input image. This algorithm was adapted from the lectures. The filter is a 3x3 matrix given by the user that operates on the image. After the image is converted to grayscale, the filter is applied to it.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | 0 | 3 | 100 | | 0 | 10 | 5 | | 0 | 0 | 0 |   Current window on image | |  |  |  | | --- | --- | --- | | 0 | -1 | 0 | | -1 | 5 | -1 | | 0 | -1 | 0 |   Operator | |  |  |  | | --- | --- | --- | | 0 | 3 | 100 | | 0 | 42 | 5 | | 0 | 0 | 0 |   Result after operation |

A 3x3 window passes over the entire image until all pixels are operated on. For example, with the above 3x3 window and 3x3 operator, the current pixel of interest, 10, becomes 42:

In edge cases such as the one above where a 3x3 window is not possible, the value 0 is substituted. The results of different operators are demonstrated in Table 1.

**Table 1: Result of filters on grayscale image**

|  |  |  |
| --- | --- | --- |
| Operator | Original | Result |
| |  |  |  | | --- | --- | --- | | 0 | -1 | 0 | | -1 | 5 | -1 | | 0 | -1 | 0 | |  |  |
| |  |  |  | | --- | --- | --- | | 1/9 | 1/9 | 1/9 | | 1/9 | 1/9 | 1/9 | | 1/9 | 1/9 | 1/9 | |  |  |

The first filter sharpens the image and the second filter blurs the image. The floats in the filter can be changed to alter the degree of sharpening and blurring on the image.

* 1. Connected Regions

The next stage of the system is converting the grayscale image to black and white identifying connected regions. First, the algorithm accepts an input file name and a desired output file name. The input image could be in colour, so the first step is to make it grayscale. Then, the program traverses through all pixels and if the pixel value is above the programmer-set threshold (in this case 150), it becomes black. Otherwise, the pixel becomes white. Sample output with different thresholds is shown in Table 2.

**Table 2: Grayscale to black and white image**

|  |  |  |
| --- | --- | --- |
| Threshold | Original | Result |
| 150 |  |  |
| 100 |  |  |

The next step of this stage identifies all connected regions in the image and counts the number of pixels in each connected region. The algorithm used is called the Two Pass algorithm, and the code in the system was adapted from pseudo code from an article titled “Connected-component Labeling” on Wikipedia. On the first pass of the algorithm, the image is traversed pixel by pixel. When a black pixel is found, it receives the smallest component label of its neighbours in a 3x3 window. If no neighbours have been labeled yet, the current pixel receives a unique label. The labels are stored in their own data structure that maps to the image, so the original image is not altered. The second pass traverses through every pixel again and checks that each pixel has the smallest label possible. A sample output of the program is shown in Table 3.

**Table 3: Sample output of connected segments algorithm**

|  |  |
| --- | --- |
| Input Image | Result |
|  | Segment 1 has 213 black pixels.  Segment 2 has 185 black pixels.  Segment 3 has 68 black pixels.  Segment 4 has 67 black pixels.  Segment 5 has 65 black pixels.  Segment 6 has 406 black pixels. |

* 1. Appling Different Filters

At this stage, the program output of applying a 1x3 and 3x1 filter was compared to the output of applying a 3x3 filter with squared terms. It was found that the outputs were identical.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | 1/9 | 1/9 | 1/9 | | 1/9 | 1/9 | 1/9 | | 1/9 | 1/9 | 1/9 |   Filter |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | 1/3 | 1/3 | 1/3 |   First filter | |  | | --- | | 1/3 | | 1/3 | | 1/3 |   Second filter |  |

* 1. Scaling Connected Components

The next stage of the program scales the components to a desired size. User input includes source file name, desired output file name, and desired scaling factor (as a float).

|  |  |  |
| --- | --- | --- |
| Scaling Factor | Original | Result |
| 0.5 |  |  |
| 0.5 |  |  |
| 0.2 |  |  |
| 0.75 |  |  |

The algorithm begins by extracting each connected component and saving separate images for each one. For each segment, two new objects are created: one object to sum the colour values that map to each coordinate (Array1), and one object to keep track of how many points influence the colour of that maps to each coordinate (Array2). The algorithm traverses through each pixel of the original segment’s image. The mapped point is calculated as current x multiplied by scale factor (new\_x) and current y multiplied by scale factor (new\_y). Then, the value of the current pixel is added to Array1(new\_x, new\_y), and the count is increased by one at Array2(new\_x, new\_y). After the entire original image is traversed, a new Image object is created. The size of the new image is the segment’s height multiplied by the scale factor by the segment’s width multiplied by the scale factor. Now, the algorithm does a pass over

Array1 and Array2. The pixel value at Image(x, y) is calculated as Array1(x, y) / Array2(x, y). This means the new pixel value is equal to the sum of colour values from the original that influence the current mapped pixel divided by the number of pixel points from the original image that map to the current pixel. Once this pass is complete, the system has N images, where N is the number of connected components. It now creates a new Image object that has the same dimensions as the input image, stitches all the scaled components together, and returns this as the output image.

* 1. Ratio of Black Pixels

This part of the system calculates the zoning feature vector for each segment in the image. First the image is converted to black and white. Then step 2.2 is used to get the segments of the image. Each segment is then treated as if it was split into a 3x3 grid.

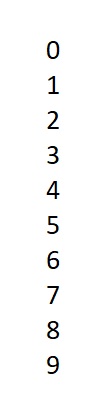
The percentage of black to white pixels is calculated for each zone of size 1/3. If the image can’t be evenly divided between the zones, then the border pixels are shared between the zones (ex. width of 7 pixels, 1st zone gets first 2 pixels plus 1/3 of value of 3rd pixel). The vector comprised of the percentages for each zone of a segment is called a zoning feature vector and can be used to describe that segment.

0.428571 0.314286 0.600000,

0.428571 0.000000 0.600000,

0.542857 0.428571 0.657143

Creating Feature Vectors

Step 2.5 was used to get the zoning feature vectors for digits 0-9. These vectors were then stored in a csv format, for easy reading and writing. The csv format is: digit,feature0,...featureN\n

0,0.42857143,0.31428573,0.6,0.42857143,0.0,0.6,0.54285717,0.42857143,0.6571429

1,0.25,0.5714286,0.2,0.0,0.5,0.2,0.42857143,0.71428573,0.54285717

2,0.2857143,0.32142857,0.61904764,0.035714287,0.32142857,0.42857143,0.75,0.64285713,0.52380955

3,0.2857143,0.32142857,0.6857143,0.17857143,0.32142857,0.6,0.39285713,0.42857143,0.6857143

4,0.114285715,0.5714286,0.6,0.45714286,0.34285715,0.6571429,0.42857143,0.54285717,0.7714286

5,0.4642857,0.2857143,0.37142858,0.25,0.25,0.54285717,0.42857143,0.42857143,0.6857143

6,0.39285713,0.2857143,0.35714287,0.60714287,0.25,0.54761904,0.60714287,0.42857143,0.64285713

7,0.2857143,0.2857143,0.64285713,0.0,0.39285713,0.4047619,0.35714287,0.53571427,0.2857143

8,0.42857143,0.31428573,0.6,0.4,0.42857143,0.6,0.5714286,0.42857143,0.6571429

9,0.45714286,0.31428573,0.6,0.31428573,0.25714287,0.6857143,0.34285715,0.45714286,0.5714286

* 1. Classifying Unknown Symbols

The classification of unknown digits in the system is done using the k-nearest neighbour algorithm. Code implementation was adapted from lecture notes. The feature vectors generated from stage 2.5 were used as knowns to classify the unknown symbols. Input for this stage includes input file of unknown digits to classify, and the file name containing the feature vectors.

|  |  |  |
| --- | --- | --- |
| Input Image | Output | Classification Accuracy |
|  | 1 2 5 1 1 | 80% |
|  | 5 1 2 | 100% |
|  | 1 9 3 | 66% |

Thus, the system has approximately 82% classification accuracy based on these trials. Due to the limit of the feature vectors, the system does not have many knowns to use for the k-nearest neighbour algorithm. Future work could include training the system with more knowns and adding additional feature vectors.

* 1. Thinning

In this stage the image is thinned according to the Z-S algorithm, as described at https://rosettacode.org/wiki/Zhang-Suen\_thinning\_algorithm:

|  |  |  |
| --- | --- | --- |
| P9 | P2 | P3 |
| P8 | P1 | P4 |
| P7 | P6 | P5 |

P1 is the pixel being tested. In the case where the surrounding pixels do not exist, they are treated as being white.

**Step 1**

Check all pixels for compliance with the following:

1. The pixel is black and has eight neighbours
2. {\displaystyle 2<=B(P1)<=6}Between 2 and 6 of the neighbours are black
3. The number of transitions from white to black, (0 -> 1) in the sequence P2,P3,P4,P5,P6,P7,P8,P9,P2 = 1
4. At least one of P2 and P4 and P6 is white
5. At least one of P4 and P6 and P8 is white

All compliant pixels are set to white once the entire image has been checked.

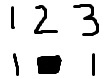
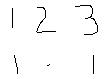
**Step 2**

Check all pixels for compliance with the following:

1. The pixel is black and has eight neighbours
2. {\displaystyle 2<=B(P1)<=6}Between 2 and 6 of the neighbours are black
3. The number of transitions from white to black, (0 -> 1) in the sequence P2,P3,P4,P5,P6,P7,P8,P9,P2 = 1
4. At least one of P2 and P4 and **P8** is white
5. At least one of **P2** and P6 and P8 is white

All compliant pixels are set to white once the entire image has been checked.

If any pixels were set in this round of either step 1 or step 2, then the steps are repeated until no pixels are changed.



1. Results

After various trials, the system correctly classified on average 82% of digits from 0 to 9. The system successfully scales, thins, filters, and coverts input images to black and white to improve image processing of handwritten digits. Incorrect classifications could be due to lack of training data, lack of features, or poor algorithm choice (k-nearest neighhbour). For example, the program often classified a handwritten ‘4’ as ‘9’. This is because a typed ‘4’ may not resemble how some people handwrite it, and it appears more like a ‘9’ to the system. To fix this issue, each digit should have multiple feature vectors to reflect the variability of handwritten digits in the practical world. In conclusion, although the system accurately classifies 82% of handwritten digits, it still has many limitations and future work is required before it can be used in the real world.

1. System limitations and future work

Although the system has the capability of identifying some digits, it still has many limitations and requires future work before it can be used for practical cases. Since the developers of this program have minimal experience and knowledge in the image processing field, future work will need to be done to improve each stage of the system. To start, the filtering stage is limited to a 3x3 window. Accuracy could be improved with a larger window. The scaling stage only works to shrink connected regions and does not increase size. Also, the program is limited to how many features it can categorize. In the practical world, image processing systems for businesses such as banks require recognition for numbers, digits, and symbols such as dollar signs and punctuation. In order to create a system with high accuracy and the ability to identify many classes of symbols, deep learning or artificial intelligence would need to be introduced to the program to be able to incorporate context. This system is limited to identifying 10 digits. The classification method for this program is also finite: the only knowns are ten fixed feature vectors that represent the digits from 0 to 9. In the future, more feature vectors might be added to allow higher accuracy. For example, the digit ‘7’ can be handwritten in various ways, so the program should incorporate multiple feature vectors for each digit. Finally, the overall accuracy of the program is not suitable for the real world. Overall improvements by more experienced developers would need to be made first before any practical use of the program could be considered.

1. Conclusions

Over four months, the system has been developed with the purpose to categorize handwritten digits. Numerous stages of the program are influenced and adapted from teaching material and online resources. With limited experience, two developers completed the system within the allocated time frame. Various programming, image processing, and analytics skills were developed while completing this project. The program is written in Java with a focus on good organization, code readability, and high accuracy. The program identifies some digits between 0 and 9, but still has some limitations and requires future work.