Handwritten digits recognition with image processing

Dastan Iyembergen
Computer Science
Nazarbayev University
Nur-Sultan (Astana), Kazakhstan
dastan.iyembergen@nu.edu.kz

***Abstract*—In the modern world, where each field of human activity is related to computer, there is a need to extract information from images. Digit recognition is an example of such cases. There are many solutions for this problem, that include machine learning. Apart from such solutions, the solution introduced in this paper does not include machine learning and does not require training process. The recognizing logic was implicitly included in the source code of the system. The logic was based on the visual features of different digit types. According to my tests, the solution was quite successful with 95% of accuracy. However, my solution has some limitations related to input type and running time.**

***Index Terms*—digit recognition, computer vision, image processing**

I. Introduction

When human sees numbers, human brain understands the meaning of this numbers. That is numbers carries meaningful information. For computer, an image of numbers is just a matrix of pixels. In order to obtain information from image we should process this image and recognize numbers. With the progress of computers people try to digitalize and automate systems. This is also related to digits. So, the problem is to obtain numerical information from digital image of handwritten digits, in order to use this information further.

The example of such problem may be zip code recognition. In order to automate the delivery service, we need a system to recognize numbers from a digital image of a zip code. Formally, this task may be described as a function that takes an rgb image of digits. The output of the function should be the list of numbers written on the image.

There are already possible solutions for this problem. In terms of machine learning, it may be considered as a classification problem. Even so, every such system needs preprocessing steps and some application of image processing techniques. My approach is based on only the image processing techniques without the use of machine learning. Such approach better in terms of dataset requirements and training time, because my solution does not need dataset and training at all. Of course, such approach has own limitations on the input data, and it also requires long running time in the prediction step.

II. Previous works

In most of papers, the digit recognition is considered as a classification problem. Leon Bottou et al. have introduced the comparison of such classifiers [1]. According to the results, the convolutional networks performed very high accuracy, but the runtime of such models were long. One of the main challenges were the feature extraction. The image pixels itself may be inputted as features. However, such model is sensitive to shifts. So, the features are extracted from curves, lines, their positions etc. The models were trained and tested with big databases of handwritten digits.

Most of recognition techniques initially assume the input image to be preprocessed and one digit only per image. Denker et al. Introduced their approach for recognizing digits from zip code, where digits were extracted from an image [2]. In this work, segmentation, resizing, skeletonization processes were applied as a preprocessing. Then the preprocessed image was used in classifier model based on neural networks.

Generally, classifiers based on neural networks illustrate very high accuracy, but the common issue of network-based classifiers is training time and requirement for big database. On the other hand, classifiers such as kNN require no training time, but require long prediction time.

III. Recognition with image processing techniques

A. Problem specification

The task is to implement a function that will take a digital image and return the recognized digits.

- Input: RGB image.
- Output: list of digits.

The requirements for the input image:

- The input image should be RGB image.
- The digit must be written with darker color than the background color.
- There should not be any other objects on the image except digits.
- The background must cover the full image.
- The background must be of one single color (for example: white paper)
- Each digit written on the paper must be a connected single component.
- The digits must be disjoint with each other.
- The thickness of digits must not be wide (pen will be OK)
- The boundaries of digits must be sharp

It is desirable that digits are written with black or blue pen on a white paper, as the function was tested with such inputs.

The output of the function is list pf rows (also lists), where each row represents the appropriate row on the image. The number of digits on different rows may differ from each other.

My approach is based on pure image processing techniques, without machine learning.

B. Structure of the solution

The solution was implemented in Python 3. There are 5 '.py' files:

- 'main.py': main script where we read the input image and call the main recognizer function by giving the image as an argument.
- 'digit_recognizer.py': the script containing the main function.
- 'digit_recognition_tools.py': module containing helper functions to work with digit features.
- 'image_processing_tools.py': module containing the general image processing methods.
- 'coord_operations.py': module to work with coordinate system objects (lines, points). Pixels are considered as points on the x-y plane.

The 'digit_recognizer.py' is divided into two functions: $recognize()$ and $digit\_recognize()$. The first function is main function, that does preprocessing steps and call the second one for each digit separately. the function $digit\_recognize()$ is responsible for recognizing a single digit and returning the prediction.

C. Implemented tools

In order to make the recognition process easier, some useful tools were implemented as separate functions.

Some classical image processing techniques were implemented. They are: converting to gray, thresholding, taking negative, erosion, dilation, closing, spatial filtering, intersection, union, complement, sobel edge detection, bilinear interpolation, transpose.

get_holes. Function to extract the holes on an image as a list of matrices. The holes were detected and filled with BFS. The BFS was run for each single black pixel until there were not black pixels.

$major\_axis$. Function to find the edge points of the major axis of an object. The points were chosen as the most distant points from each other.

$minor\_axis$. Function to find minor axis points as a segment perpendicular to the major axis.

$defect\_filling$. Function to fill defects on the object. This was done by analyzing the neighboring pixels of each pixel. If there are white pixels on opposite edges on 3x3 matrix, and these pixels are not connected, then the center pixel was colored to white.

$dists\_from\_left$. The distances from left edge of the image to the first white pixel of the object were measured. This function where used to recognize the digit 5.

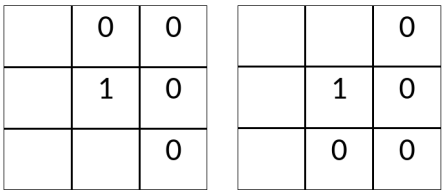


Fig. 1: (a)(b). (a) - structuring element for detecting upper point of right maximum, (b) - structuring element for detecting lower point of right maximum.

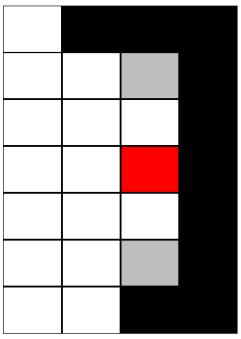


Fig. 2: The example of use of right maximum detection. The resulting pixels of applying hit or miss are colored with gray. The red pixel is center of maximum.

$digits\_from\_right$. Similar to the $dists\_from\_left$, the distances from right edge were calculated.

$draw\_line$. The function draws white line between the given points. The image matrices were considered as x-y plane.

$right\_maxs$. Finding a maximums from the right side. This was done by applying "hit or miss" method with the structuring elements given in figure 1. The results of application of these two structuring elements then were combined to find the middle of the maximum point. The example of right maximum detection was shown in figure 2.

$left\_maxs$, top_maxs. Similar to the $right\_maxs$.

$widths$. The function to find with of an object for each row of pixels. For each row of pixels, the distance between the leftmost white pixel and rightmost pixel was calculated.

D. Preprocessing

Converting to gray. The input RGB image was converted to the gray image, by applying the simple formula for each pixel: $p = \sqrt{r^2 + b^2 + c^2}$, where r, g, b are the red, green and blue components of a pixel.

Taking the negative. In our case, the objects are darker than the background, but for the processing purposes, it is better that the objects are white while the background is black. So, the negative of the gray image from the previous step was taken by simple formula $q = 255 - p$.

Converting to binary image. That was one of the challenging parts. Simple thresholding with static value was not

applicable for all inputs. Dynamic thresholding such as taking 1/4 of the biggest difference of pixels also was not efficient, because there may be shadowed regions, that will overcome this threshold value.

The possible solution for shadows was using edge detection techniques such as Sobel. A change in color around digit boundaries is big, while a change in shadow regions is almost zero, because shadow pixels change evenly. The issue with such method is that the edge detection detects only the boundaries of digits.

The solution was to combine thresholding and edge detection together. The idea was to apply dilation on edge detected image, but only for those pixels which has value 1 on the thresholded image. This guarantees that the pixels outside the edges will not be extended.

Closing. The output of the previous algorithm may still have some gaps on the object bodies. to fill such gaps closing procedure was applied: two dilation operations followed by two erosion operations.

Segmentation. Firstly, the image was divided into rows of digits. This was done by finding the pixel rows that have no white pixels. Such pixel rows are counted as a separation between rows of digits.

Each digit row then is analyzed from left to right. To extract the connected regions, Bread First Search (BFS) was applied for each white pixel. Then the resulting component was cropped out from the image. Before the BFS algorithm, region filling procedure based on dilation was tried. However, the running time of the region filling procedure was very long comparing to the BFS.

After all these prepossessing procedures, each component (digit) is given as an argument for the next function $digit\_recognize$.

On its turn, $digit\_recognize$ also perform some prepossessing operations.

Resizing. The cropped image of a single digit is resized to fit into 75x75 square. Analyzing the images of the same dimensions is much easier than dealing with different shapes of images.

Defect filling. Even after the prepossessing steps done before, digits may have some defects (gaps). To fill such defects $defect\_filling$ was applied. After all of these prepossessing steps, the digits images are ready for analyzing.

E. Analyzing digits

The analysis was done in a decision tree manner. Each new extracted feature narrows down the possible outcomes, until we have only one possible outcome.

Holes. The first target was holes and their positions. The holes were extracted by running get_holes. The position of a hole were defined as a mid point of major axis of a hole. Based on these holes all digits that have hole are recognized (figure 3):

- If the number of holes is two, then it is 8.
- If there is one big hole, it is 0.
- If there is one hole at the upper part, it is 9.

Fig. 3: Digits with holes

Fig. 4: Major axis point on a single lined 1

- If there is one hole at the lower part, it is 6 or 2. the digits 6 and 2 are distinguished based on the size of the hole. The hole of the digit 6 is larger than the hole of the digit 2.

Single lined 1. Next, the major and minor axis were measured. If the minor axis is equal to the thickness, this digit is 1 (figure 4).

Recognizing 4. The next step was to recognize 4. the digit 4 has two line ends on the upper part of the object. These two points were detected by top_maxs. If we connect these two points with $draw\_line$, then the number of hole (found with get_holes) will be 1 (figure 5).

Recognizing 7 with horizontal segment (-) at the center. Similarly to the previous step, line ends from left and right were found with $right\_maxs$ and $left\_maxs$. Then the points were checked for the connection, if there is connection then it is line segment. If this line segment is lies approximately at the center, then the digit is 7 (figure 6).

Recognizing 3 with 3 key points. 3 has 3 maxima points from left. the maximums where detected with $left\_maxs$.

Fig. 5: drawing line on the top of 4. The red points are the detected top maxima

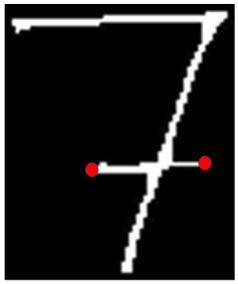


Fig. 6: Recognizing 7 with the line segment at the middle part. The line ends of the segment were pointed with red points

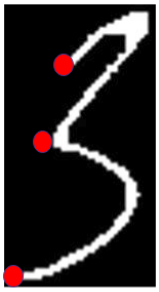


Fig. 7: Three key points to detect the digit 3.

Then the three points were analyzed with distance between them, and the connection. These three points should not be connected with a line, and the mean should be approximately at the middle part of the object.

Recognizing 1 (without base horizontal line) and 7 (without line segment at the center). These digits are very similar. The only difference is the angle between base line and the top line. If we calculate the width of the object for each row of pixels (with $widths$), 7 will have more thin numbers from bottom than 1 (figure 8).

Recognizing 5. The digit 5 was recognized with distances from left and right. If we divide a digit 5 into upper and lower parts (equally), the intersection of the longest distance from the right on the upper part and the longest distance from the left on the lower part will cover the most part of the maximum horizontal distance of the object (figure 9).

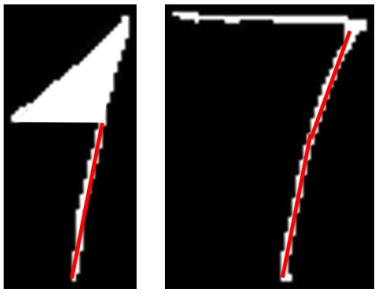


Fig. 8: recognizing 1 and 7. Red lines - thin parts of from $widths$

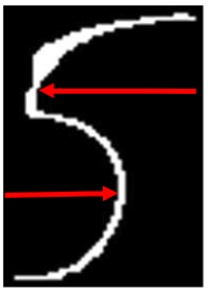


Fig. 9: Recognizing 5 using distances from right and left.

Fig. 10: Distinguishing between 1 and 2 with base lines.

1 (with horizontal base) and 2. These digits are also similar to each other. the significant difference between them is the intersection of the main line with the base line. This intersection is on the middle of the base line, while for 2 this intersection is located at the left part of the base line. If we divide the object to upper and lower parts (equally), and connect the topmost point with the leftmost point of the base line with a straight line, one hole may appear. For 1, one hole below the drawn line will appear. For 2, there may be no hole, or hole may be above the drawn line. This feature was illustrated on figure 10.

IV. Results

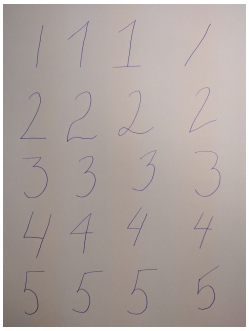
The system was tested on the dataset of 40 digits with 4 instances for each digit (figure 11). Note, that the digits were written enough accurately, as the system is not capable of recognizing difficult and clumsy digits. The pictures were taken with cellphone camera with lamp light. The digits were written with blue pen.

The outputs for these two images were:

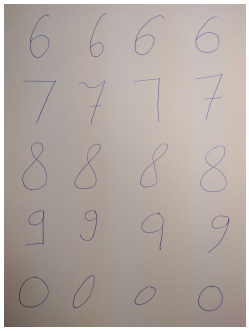
$$output_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 9 & 4 & 4 \\ 5 & 5 & 5 & 5 \end{bmatrix} \quad output_2 = \begin{bmatrix} 6 & 6 & 6 & 6 \\ 7 & 4 & 7 & 7 \\ 8 & 8 & 8 & 8 \\ 9 & 9 & 9 & 9 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Preprocessing. The preprocessed image were shown in figure 12.

Digits Recognition. The accuracy of the system was 95%. The only digits that were incorrectly recognized are 4 with a hole and 7 with wave like line at the top.

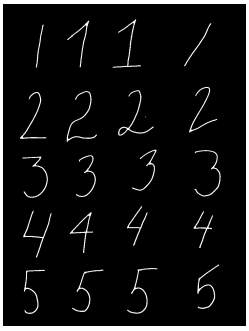


(a) For digits 1-5

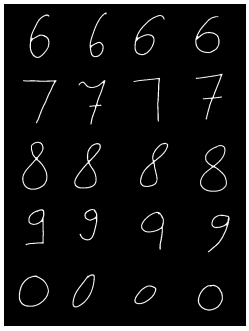


(b) For digits 6-9, 0

Fig. 11: Testing images



(a) for test 1



(b) for test 2

Fig. 12: Preprocessed images

Runtime. The running time for these two testing images was quite long. The most part of time were spent on preprocessing step. The preprocessing step took approximately 4 minutes for each input image.

V. Discussion

The preprocessing step was good enough to detect all the digits and ignore the small noises. The positions of the digits also were detected correctly as a matrix 5x4. However, it should be noted that the preprocessing is sensitive for

noises and extra objects on the image. In our case there was not any problem, as the image was taken accurately.

Mistakes in recognition are related to missed cases. The problem with 4 with hole was that the system automatically thinks that it is 9 as the hole is located at the top part. The problem with the 7 with wave like line at the top was that when we connect two top maxes there appears new hole as in case of 4. These cases of the digits were missed in development process. The recognition process also is highly sensitive to noises, as we have a deal with pixel-wise operations.

The running time of the system depends on the size of the input image. In our case, the both images were 1280x960. One solution for this problem may be to use the computational power of GPU. Also, the system may be rewritten in C++, that is several times faster than Python 3.

VI. Conclusion

There is need for recognizing handwritten digits from an digital image in different spheres of human's life. There are many possible solutions for this problem. Most of such solutions use machine learning in some forms. Some of the solutions based on neural networks and convolutional networks are quite accurate. However, such accurate models require big dataset for training and time to train and predict. In addition, such models work only with preprocessed inputs.

So, my approach combines both preprocessing and recognition steps. The solution does not use any kind of machine learning, so that there is no need to train the model. The idea behind is that the model was trained implicitly by writing the whole logic within the code.

The system was able to correctly recognize 95% of the testing digits, and correctly do the ordering of the digits. However, the solution has some limitations. The system is not capable to recognize extraordinary written digits, and sensitive to noises. The mistakes with the different cases of digits may be adjusted in the internal logic of the system. The issue with the running time of the preprocessing step may be partially solved by using more computational power and faster programming language.

References

[1] L. Bottou, C. Cortes, J. Denker, H. Druckera, I. Guyon, L. Jackel, Y. LeCun, U. Muller, E. Sackinger, P. Simard, and V. Vapnik, "Comparison of classifier methods: a case study in handwritten digit recognition," *Proceedings of the 12th IAPR International Conference on Pattern Recognition*, vol. 3, 1994.

[2] J. Denker, W. R. Gardner, H. Graf, D. Henderson, R. E. Howard, W. Hubbard, L. Jackel, H. Baird, and I. Guyon, "Neural network recognizer for hand-written zip code digits," *Neural Information Processing Systems (NIPS)*, vol. 1, pp. 323–331, 01 1989.