Digit Recognition through Finger Movement

Jeremy Harisch
jharisch@postech.ac.kr

Kimberly Lange
kimberly@postech.ac.kr

Department of Computer Science and Engineering,
Pohang University of Science and Technology,
77 Cheongam-Ro, Nam-Gu, Pohang 37673, South Korea

Abstract

This term project in Computer Vision aims to reliably recognize digits through finger movement. Therefore, a finger with easily noticeable tape on it moves in front of a camera while the program records the path and analyzes it in order to return the digit that was drawn in the air. To achieve this, different computer vision techniques were used based on machine learning and non-machine learning approaches.

1. Introduction

This project aims to provide another way to input data - in this case digits - into a digital device that is equipped with or can be connected to a camera. Therefore, the user positions himself in front of the camera and draws a number between 0 and 9 with his finger in the air, which then automatically is recognized by the program.

In order to achieve this goal, a couple of requirements had to be met. First of all the camera has to be accessed and a way to detect the finger has to be found. To make it easier, the finger was marked with coloured tape. This allows to enter the colour into the program and make it search for the colour in the camera frame. Additionally, the application also has to store the path of the finger in an appropriate way and then classify which digit it shows.

To meet these requirements different approaches can be chosen. Basically, for both main tasks - tracking and classifying - either machine learning or non-machine learning approaches can be chosen. However, some might work better than others. Consequently, the main task was to find the best methods to fulfill the requirements and create an application with a sufficiently good performance. How this was implemented and what results were achieved will be explained in detail in the following sections. After that, possible use cases, limitations, and improvements will be discussed.

2. Implementation

In the following subsections the implementation will be outlined in detail. The work can be split into two major parts. First, the finger in front of the camera has to be recognized and its movement has to be tracked. In addition to the tracking the path has to be recorded and stored for later analysis. Second, the stored data has to be classified. More specifically, the program has to recognize which digit was drawn in front of the camera.

As the development environment Python 2 was chosen because it works well with the OpenCV library that was used for image processing and initializing, training, and testing of the neural network. This library provides many useful functions that were used in both parts of the implementation. Also, the NumPy library was used for efficient and easy computations on vectors and matrices.

2.1. Object Detection and Path Tracking

In order to record the finger movement in front of the camera the `VideoCapture(0)` method from the OpenCV library was used. With the parameter given the function was told to use the default camera. As long as the video is captured and the user does not close the window, the frames are analyzed and processed.

To make the finger tracking less complicated, a coloured tape has to be attached to the finger tip. This allows to easily convert the BGR image into a HSV image and threshold the result for a range containing the colour of the tape. Of course the colour of the tape can also be different and as long as the user takes care that there are no objects with the same colour in front of the camera, the OpenCV method `inRange` reliably extracts only the parts between the two given colour values.

Before extracting the position of the fingertip in the frame and tracking the movement gaussian blur is applied in order to remove noise and fuzzy contours. Thereafter, OpenCV's function `findContours` finds the contours in

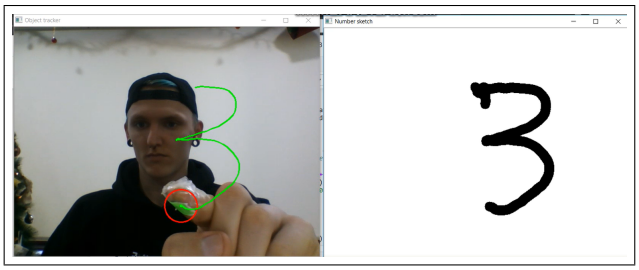

Figure 1. Drawing a digit

the HSV image. From all these contours the longest one is selected, as this most probably contours the tape on the finger if the person is close enough to the camera, and saved in a variable. In this way other smaller objects in the background with the same colour are disregarded. Moreover, the main area is required to be larger than a prior defined threshold to avoid choosing the wrong object if the tape finger is too far away from the camera or not recognized correctly.

After this step, a minimum circle around the main area is constructed with the help of the `minEnclosingCircle` function and the circles centre coordinated and its radius are stored. With these parameters two circles are drawn into the frame. One is the minimal enclosing circle and the other one is only a small dot indicating the centre of the fingertip. This position is used for tracking the path. Showing this allows the user to check if the tape was recognized correctly and if the program will be able to track and record the right path.

Furthermore, the circle's centre coordinates are stored in an array so that the finger's path can easily be drawn by looping through all the points. However, just adding new points and drawing all of them would cause the window to look too chaotic and cluttered. Consequently, the first point in the array is removed every time the list exceeds a predetermined length so that only the most recent movements of the taped finger are displayed.

In addition, a second window is shown simultaneously where only the path is drawn with a thick black line on a white background. Here, a point does not disappear anymore once it was drawn because this picture will be used later for the digit recognition. Moreover, it is important to flip the image. Otherwise it would be mirrored and more difficult to classify later if the neural network was trained with non-mirrored images of handwritten digits.

One step of drawing a digit and tracing the path can be seen in figure 1.

2.2. Digit Recognition

In order to classify the digit drawn in front of the camera correctly, a neural network was trained with pictures of handwritten digits.

The numbers were extracted from an image containing 50000 handwritten digits, 500 pictures for each number

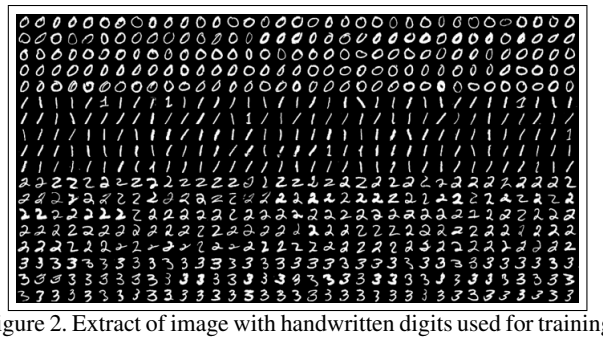

Figure 2. Extract of image with handwritten digits used for training the neural network

(see figure 2).Therefore, the picture was first transformed to gray-scale and down-scaled. As all digits are contained in this one picture, the image has to be split up in small 20 times 20 pictures.

The actual training of the neural network is realized with the k-Nearest Neighbors(KNN) algorithm, it is one of the simplest algorithms for image classification and regression-methods. Furthermore, it is an unsupervised learning algorithm, which is a good solution for this project. Thus it has a very fast training time after recording the number. If we had used a supervised training algorithm, we would have used a support vector machine(SVM). But due to its long training time, we have chosen the KNN. But also on the other side, we would have much better results with the SVM.

70% of our training data will actually be used for training the model, the other 30% will be used for testing the neural network. With this training split-up, we had the best results for classifying hand-written digits. In addition, we have chosen to classify each picture with nine nearest neighbors. The average accuracy of this training is 92.13%, which is not the best but a good basement to start of.

2.3. Connecting both parts

Finally, the image generated in 2.1 has to be classified by the neural network initialized and trained in 2.2.

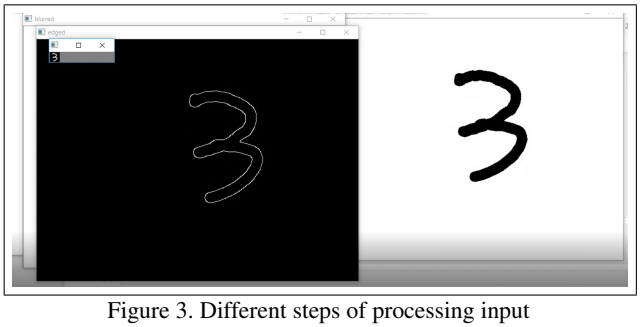

Figure 3. Different steps of processing input

Hence, the image containing the finger's path is converted to gray-scale and blurred in order to reduce noise and soften the edges. After that, the OpenCV function for

canny edge detection is used in order to find all edges in the picture. This makes it possible to extract the contour of the digit in the picture (see 4). The contour is needed to easily square and resize the picture. Converting the image into a square is done by a separate method which first checks if the picture already is a square, which is the case when the height equals the width of the image, and makes it squared if it is not. This can be done by simply adding black pixels at two opposite sides of the picture, either top and bottom or left and right, depending on the ratio between width and height of the image.

Thereafter, another function resizes the image 20 times 20 pixels in order to make it comparable to the images learned by the neural network. This comparision is then done in the next step. Calling the function `findNearest` and giving it the small, processed image as a parameter finds the handwritten digit the given picture is closest to and finally returns the result of this search.

3. Results

While testing the actual application, we were facing a lot of problems. First of all, in this prototype we are using a tape at a finger tip to recognize it. Thus, makes it way easier to track and record the movement of it. But unfortunately, it is not stable to some noise of the light. For example, if it is too shiny, the colored finger tip will not be recognized anymore. Another point is, that maybe something in the background also has the same color as the tape and this may interrupt with recognizing the finger tip.

Furthermore, this prototype does not have any error handling if the number cannot be classified. If this happens, the program shuts down with an error log.

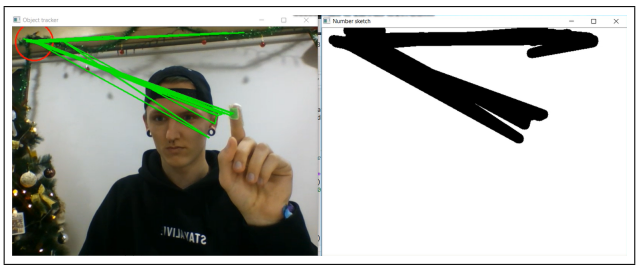

Figure 4. Performance lost due to background noise

But on the other hand, if there is no background noise, the program is able to recognize and record the movement of the colored fingertip. Furthermore, it is able to classify the digit which was input, in a small time with a high accuracy.

To improve this program, we want to implement an actual hand and finger recognition algorithm. Thus there will be way less background noises and the performance can be enhanced. A second improvement would be to change the actual neural network into a pretrained one. In this way, it is possible to improve the accuracy up to 99% or even higher.

4. Discussion

As already outlined in the last section, the program code has its problems and limitations, but in most sessions the application still worked well and provided us with satisfactory results for the short period of time that we worked on it. Moreover, it is good to see that the application actually could be useful after some further development.

Possible use cases for this program could be all those situations where the input of digits is required and it is not possible or convenient to draw the digits on a touchscreen or enter them via a keyboard.

A good example for this are smart-watches. Many of them already have an integrated camera and the touch-pad is too small for user input by hand. Consequently, it would be very practical to write the input in the air in front of the camera. The program would then recognize and classify the input.

Furthermore, this application could also substitute remote controls. Some TVs already have a built-in camera and the others could just be connected to one. Then the spectator would be able to change the channel by entering the channel's number only with gestures and would not have to search for the remote control again.

At the same time, these examples also reveal possible improvements and extensions for the program. In order to fully take advantage of the writing through finger movement, extending the classifier to recognize handwritten letters as well would be really helpful. This could for example allow the user of a smart-watch do send short replies to text messages very fast without opening the connected phone or struggling with the small touch-pad.

Recognizing handwritten letters, though, is much more difficult than recognizing handwritten digits. First, there are more than twice as many letters than digits. This makes it more difficult for a neural network to learn to distinguish the inputs and classify them correctly. Also, more data and computing power are needed for training. With today's state-of-the-art networks and computers this, however, is not a very big problem anymore. Another problem to face would be the fact that some letters cannot be drawn - or at least not perfectly drawn - without setting down the pen. This would cause problems with the finger tracking.

Moreover, neural networks can always be improved, trying to achieve even more accuracy.

Finally, it can be concluded that the presented application already works well in practice and fulfills the main requirements. Future work could focus on improving the digit recognition and expanding the program by letter recognition, maybe even for different alphabets.