Application for Performing Matrix operations Using Images as Input

Vidhixa Joshi and Divya Lalwani
CSCI-B657 Submission

Abstract

Digit recognition has many applications in day to day life. They can be used for automatic navigation of posts based reading their zipcode, converting analog records into digital format, etc. We attempt to use the concept of digit OCR to recognize digits in images containing matrices. Our application will provide functionality for matrix multiplication, addition, etc. We use two approaches here. First is to build our own library of images and use Knearest neighbor. Second is to use Tesseract's inbuilt library for the same. With the Tesseract library, the application works significantly well as compared to Knearest which has higher error rate.

Introduction

Matrix operations form major part of mathematics in the fields of Computer vision, astronomy, graphics, etc. Calculators are convenient for basic arithmetic and trigonometric type of operations but they are tedious when it comes to matrix operations. For large matrices, wouldn't it be great to get output by passing images of matrices. That was the idea and motivation behind our project; to build an application which could perform matrix operations by processing images.

Background

For our application, the main part is to implement digit recognition. We referred to the papers written by Nakashima et al on handwritten digit recognition. This paper focused on the importance of normalizing theimage in size and intensity before building up feature map[1].

Knearest neighbor technique is simple to implement and provide good results[2]. It is dependent on training sets and when training set is extrememly large, it takes longer time, which is comparable to more sophisticated techniques that give better results. Thus we have followed the approach of keeping a small dataset for learning.

Related Work

Digit recognition can be implemented in many ways. Some of the literature that we reviewed used the back-propagation technique for building feature maps and thereby developing classifier for digits[3]. This technique is very slow to implement.

Another popular approach is to create a Histogram of Gradients[4]. This technique is used widely for object recognition. In this, a feature vector is built and classifier is trained.

Apart from these, there are various OCR applications available in market which are platform independent are written in Python, Java, C++, etc.

Our approach:

The user has to choose the operation first. Then depending upon his selection, he/she would be asked to enter 1 or 2 images of matrix. The user will input the name of matrix and the program will locate the same in the folder. The user will then be asked to enter the number of rows and columns. With this as input, the pre-processing step starts. Preprocessing is performed by both

While preprocessing the image, we recognize firstly the blob of matrix. As we proceed, we convert it into greyscale. Then we blur the image to normalize the noise across the image which is followed by adaptive thresholding. The values are set based on the results obtained by various trials. This is followed by finding contours in the image. Contour helps us in recognizing blobs of data which in our case are our numbers.

Second step is the learning phase. We use the k nearest neighbor algorithm for this purpose. We use a subset of MNIST dataset for our convenience. There are 60 images which are a mix of printed as well as handwritten digits. Each image has a label associated which is its digit. In the learning phase, all images have to be of specified size of 20*30. Contours will recognize the shape of digit and we built a cvMat vector and associate labels to each each corresponding image.

Next step is to actually recognize the matrix image inputted by the user. This goes through the same steps as learning step. Then we extract cells from by spatial segmentation and each part is cropped to be used as a separate image for recognition. We try and divide the larger problem into smaller sub-problems. Image could be of larger/smaller size then training data. It works well with both as pre-processing handles the sizing problems.

As we recognize digits and display on the screen, they are being saved in an array which will be used to perform matrix operation.

The digit recognition techniques used can be explained as follows:

1. KNearestNeighbors using OpenCV:
   Knearest neighbor needs a training set which in our case is a self-built library

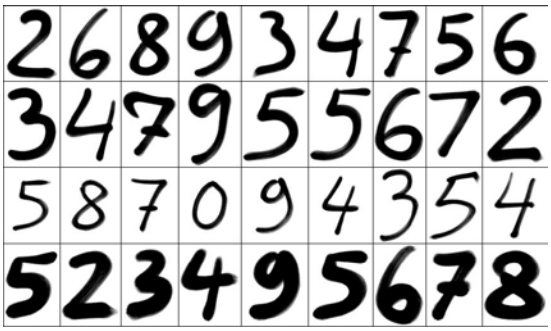

Fig 1: Handwritten digits

of over 60 images. We were facing issues to load the entire MNIST database. So, we have used a subset of MNIST dataset. K nearest neighbor will associate the label with images of training set. While recognizing the images, closest match is found based on the training set.

2. Tesseract library
   We make use of Tesseract's in-built dataset for digit recognition. We installed Tesseract baseapi and Leptonica library installed on our machines. We then include the library and pass it an image for recognition. With Tesseract we did not feel the need for preprocessing images as in OpenCV.

After recognizing digits from above approaches, we construct 2 different matrix from these numbers. Each matrix represents one of the above approaches.

Results

We tested our application using both handwritten and printed digits. We place a restriction on the quality of image required. It has to be taken in flash light otherwise we don't get the correct output. The result that we obtained from printed images were about 90% accurate. We observed false recognition in some common cases like recognizing 5 as 6, recognizing 8 as 0 etc. Here are some screenshots of results that we obtained.

(The screenshots of running code on handwritten as well as printed digits)

Challenges

Android configurations were the biggest hurdles which resulted in shift of scope. We majorly faced these issues on Windows machines this we moved to Linux. Even though we had started working on C++, we were simultaneously working on the configuring Android in Eclipse for Linux. We also began coding for project with JavaCV but since we were new to it, we spent a lot of time understanding minute things about it. OpenCV with C++ was simpler and allowed us to explore its built-in functions. Thus, we used OpenCV with our own library and implemented the same using Tesseract with Leptonica.

Future Scope

We wish to get it running on android. Also we want to find a less memory and time intensive dataset like MNIST. Get minimal inputs from user. Include as many test cases as possible to correctly evaluate the working.

Conclusions

The application works accurately with Tesseract library. In OpenCV the major hurdles were to get the digits extracted correctly. We assume that the image is deskwed using an application like photoshop and then passed on. The application has a lot of scope as it would prove to be handy once it gets running on local devices.

References

[1] Handwritten Digit Recognition: Investigation of Normalization and Feature extraction techniques

http://www.sciencedirect.com/science/article/pii/S0031320303002243

[2]Knearest neighbor working by Jeremy Kun http://jeremykun.com/2012/08/26/k-nearest-neighbors-and-handwritten-digit-classification/

[3] Handwritten Digit Recognition Using Back Propagation Technique

Paper presented in class

[4] Handwritten Digit Recognition by Jitendra Malik

https://mlss.soe.ucsc.edu/sites/default/files/malik-lec2.pdf

For OpenCV, we have followed the approach mentioned in this blog http://stackoverflow.com/questions/9413216/simple-digit-recognition-ocr-in-opencv-python

http://www.aishack.in/2010/08/sudoku-grabber-with-opencv/

http://my-tech-talk.blogspot.com/2012/06/digit-recognition-with-opencv.html

Figure 2: Recognition on printed digits and taking transpose. Give 100% accurate result

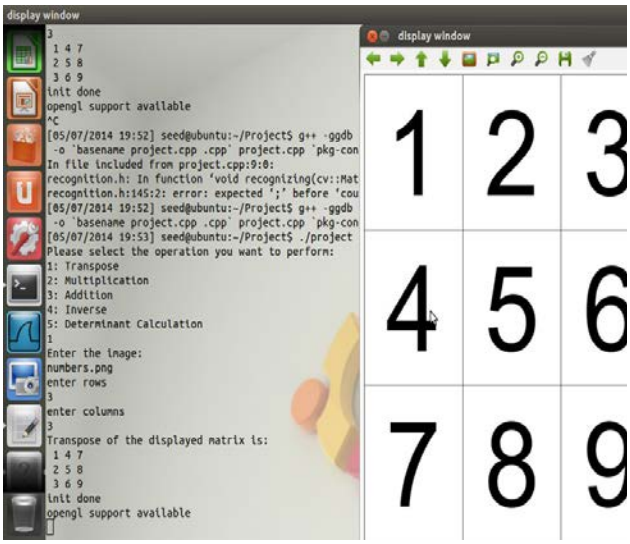

Figure 3: Recognition on handwritten digits and taking transpose. Gives average result