**REal-Time Handwiritng recognition using tensorflow and Raspberry pi**

PREPARED BY  
Hai Ho

In partial fulfillment of the requirements for  
 CPE 4903

PREPARED BY: ASHWEEN DEDANI   
  
APPROVED BY:

MAY 05, 2021  
  
KENNESAW STATE UNIVERSITY

ABSTRACT

Real-Time Hand Writing project lets the user detect the handwritten digit. This small project is implemented on the Raspberry Pi which gives the user complete flexibility to travel with the device and show its use. We make the digit display on the SenseHat of the Raspberry pi. Since we are trying to implement the project in real-time, we loop through the camera to capture the photos and run it through the trained Neural Network and check for the confidence level to be more than 50%. If the captured photo gives 50% of the confidence we then print the digit on the SenseHat.

In order to train the Neural Network to give the best result we use Convolution Neural Network to train till the point it gives accuracy above 95%. With such accuracy we can be certain for the Network to give the accurate result for most of the case. Since we are using the Raspberry pi camera we need to check with the resolution to get the maximum output. Raspberry pi is a development platform where Neural Network can be trained and implemented with not such a great difficulty, in fact Raspberry pi allows users to capture images on a regular interval that is such a great use in this project. .

Table of contents

Section Title Page

1. INTRODUCTION 1

1.1 Libraries Used 1

1.2 Hardware and Setups 1

1.2.1 Implementing pi Setup 2

1.2.2 Implementing MNIst Code 3

1.2.3 Saving Weights and Training Outputs.

2. CAMERA SETUP 4

2.1 Code Structure 4

2.1.1 Image Construction 4

2.1.2 Confidence Construction 5

2.2 Digits placement 5

3. GRAPHS 6

3.1 Training Graphs 6

4. CONCLUSION 7

BIBLIOGRAPHY 8

1. INTRODUCTION

1.1 LIBRARIES USED

Raspberry pi is a developmental board running on LINUX OS. We use tensorflow, opencv, numpy, and different utilities in our project to enable Raspberry pi to train Convolutional Neural Network and detect the digit.

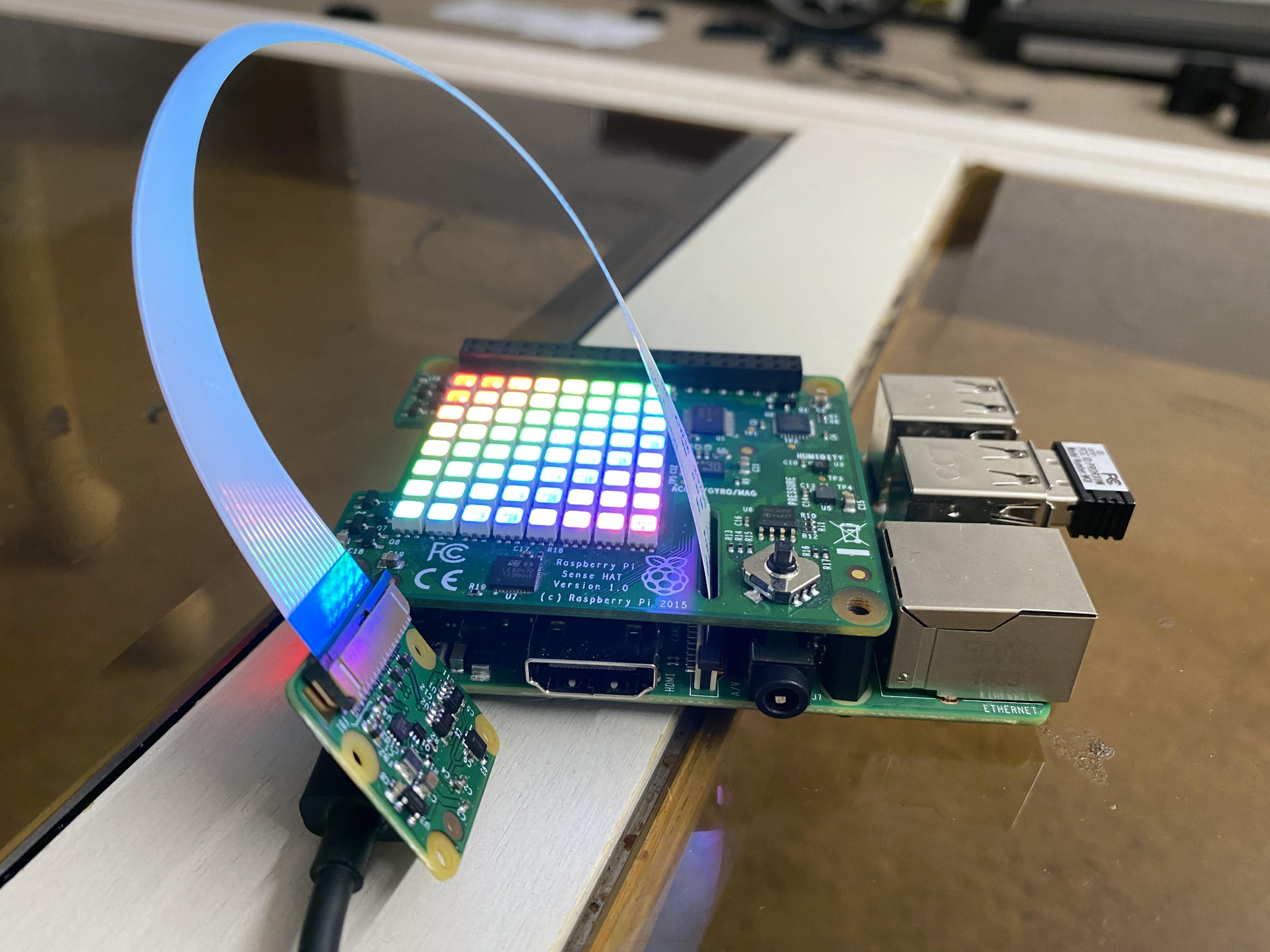
OpenCV is basically a library that serves as the wrapper for Python in C/C++. It uses numpy to perform the mathematical operation on images and it is used in real-time computer vision problems. In this project we use OpenCV to make the images captured to be of designated resolution that converts it into the array of numbers.

TensorFlow is a library based on statistics and mathematical data flow solutions that is used in this project to train the Neural Network. Tensorflow module called Keras that is basically the mathematical solution to find the Gradient Descent, Forward and Backward Propagation, logistic regression, linear regression and many more. We use it to make the network convolutionally trained.

1.2 HARDWARE AND SETUP

We try to keep the complication level to its least and thus the Hardware used are basic and can easily be found in the market. Raspberry pi, Raspberry pi camera, SenseHat, Micro SD card, USB, and a laptop.

1.2.1 IMPLEMENTING PI SETUP. Picture of my setup is given below. Setup shows the way in which the pi camera, senseHat and the board are integrated in a small space.

Figure 1

In the Figure2 we see the sense displaying a number. It’s the number that is predicted by the Pi and the Figure3 shows the Handwritten digit itself.

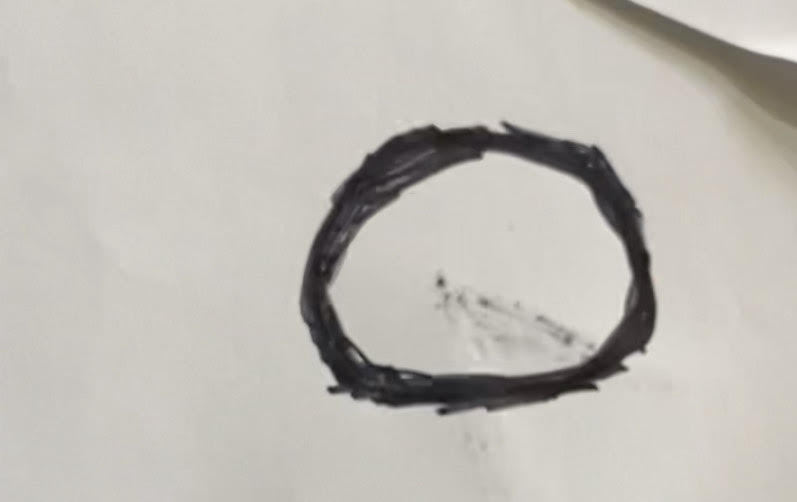
 

Figure 2 Figure 3

.1.2.2 IMPLEMENTING MNIST TRAINING. We train the Convolutional training Network till it gives us 95% of the accuray. In my training I try to make sure that the loss is linear and that there is a smooth convergence in the lost. The graph is plotted in the next section but the part that show the code to implement the CNN and its summary is posted below:



Figure 4 Figure 5

Keras setup shown in Figure 4 and the number of weights used can be seen in Figure 5.

1.2.3 SAVING WEIGHTS AND OUTPUT. We need to save the weights from the training part. It is a crucial part of implementing the Pi setup. The weights are saved in the ‘.h5’ file. Figure 6 shows the code that saves the weight in the .h5 in the same working directory. Figure 7 shows the training output of accuray 98%.

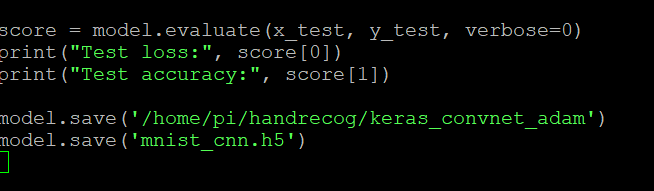


Figure 6 (The last part of the code save the .h5 file)



Figure 7 (training output of 98%)

2. CAMERA SETUP

2.1 CODE STRUCTURE

It's the most crucial and fun part of the project. In this part we implement the code that loads in the weight, captures the image, constructs and displays the output on the SenseHat. The difficult part that one faces is constructing the image and making it pass through the weights. In this section we look at the structure of code and how to overcome the problems.

2.1.1 IMAGE CONSTRUCTION. When we train the Network it takes in 1x28x28 construction of the image and with the black background and white colored digits. But when we take the image it is of different resolution and has the background of white and black ink. Thus we need to make the image specification of the level that takes in the desired input. We resize the image 1x28x28 and make the image inverted with the background swapped. The following code does that:

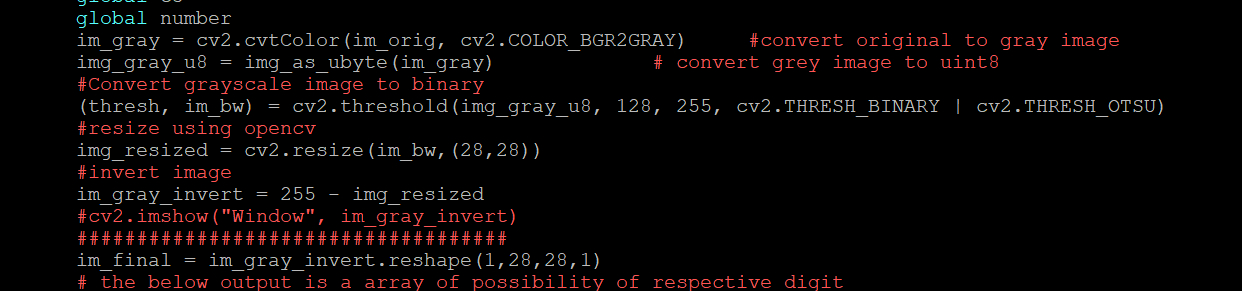


Figure 8 Image Construction

2.1.2 CONFIDENCE CONSTRUCTION We need to make sure that the confidence level is above 50% and the following can be achieved by making the camera capture in every .5 seconds and evaluating the result. If the following digit appears the highest for all the 5 runs, we can be sure with the confidence of 50% that the image is of the following digit. Picture below shows the implementation of the confidence level:

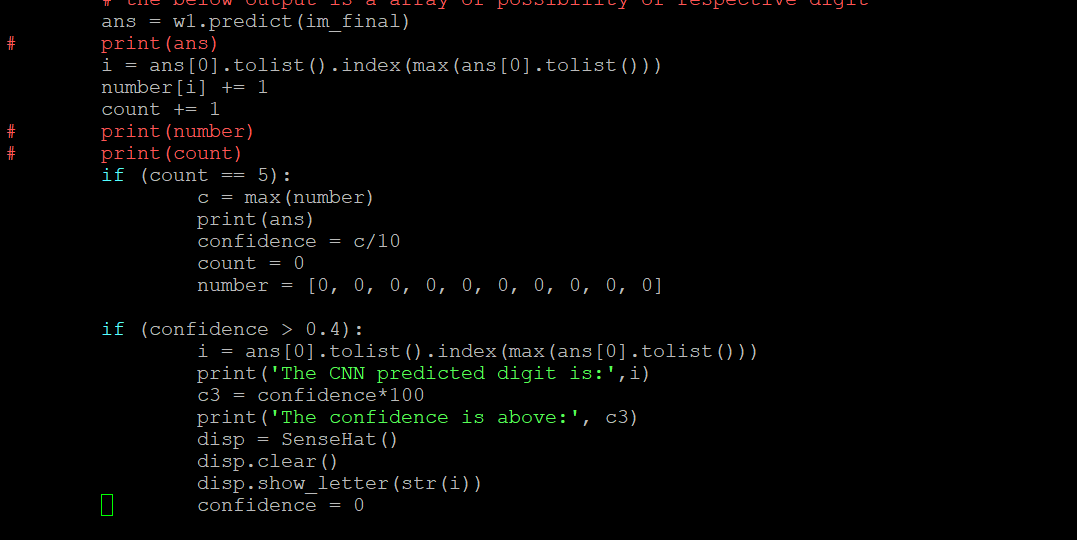


Figure 9 Confidence Construction

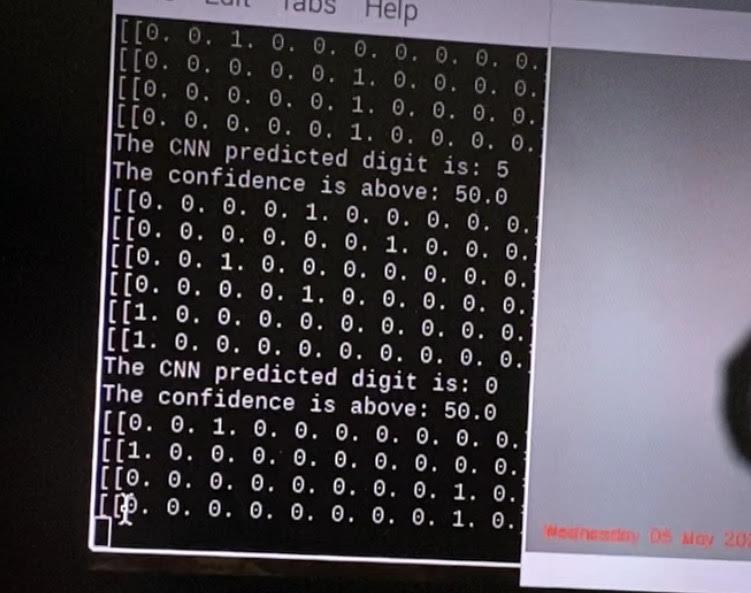


Figure 10 Confidence display

2.2 DIGITS PLACEMENT. Now the challenging part is to make the digit appear on the frame. When we code for the camera to capture the digit we need to be sure that the digit is seen in the frame of the camera. We can accomplish this by displaying the videostream of the camera and adjusting the lense. The following picture of code shows the videostream.



Figure 11 Code for Videostream

3. GRAPHS

3.1 TRAINING GRAPH

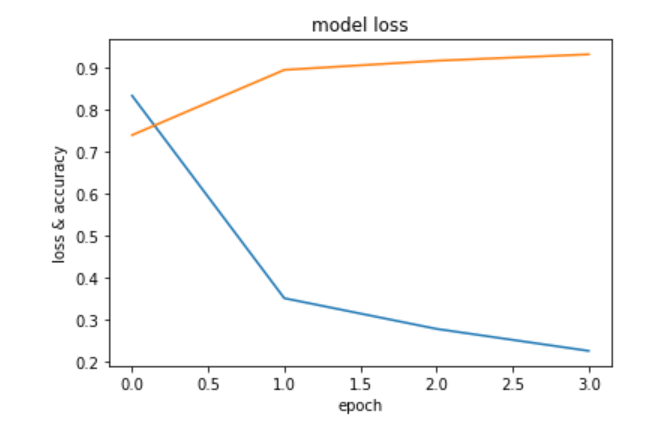
Graph is something that makes the user read the code more accurately and easily. In the following graph accuracy and loss is shown. 

Figure 12 Graph for the training and loss.

5. CONCLUSION

This small project shows the amazing things that Machine Learning can help achieve. The complications in this project were mostly the part where we are required to play with the image and resize it to the desired input. Which helps us to learn more about OpenCv and how this library can be implemented in python. We also get to better understand:

Image processing, Convolution Neural Network, Hardware implementation, Real-time recognition, working with linux based camera systems, and how Machine Learning tools hold the power to shape the future.

bibliography

1. https://github.com/Q-point/PiCamera-Digit-Recognizer/blob/master/PiCameraApp.py