REAL-TIME HANDWIRITNG RECOGNITION USING TENSORFLOW AND RASPBERRY PI 4

PREPARED BY
Aris Jamal

In partial fulfillment of the requirements for
CPE 4903

PREPARED BY: Aris Jamal
APPROVED BY:

DECEMBER 02, 2021

KENNESAW STATE UNIVERSITY

ABSTRACT

Real time handwriting digit recognition done with creating a Convolutional Neural Network (CNN) on a Raspberry Pi. The Raspberry Pi will be a standalone system that can work on its own only requiring external power source. The Pi has two attachments, a LED sense hat, and a camera. The camera will take in handwritten digits, process the information through the network and then output the correct digit on the LED sense hat. The training samples were taken from MNIST and trained using TensorFlow and Keras API to create a python code with anaconda distribution. The application of the system would be to use the raspberry pi to read and correctly recognize handwritten digits without the use of any external systems.

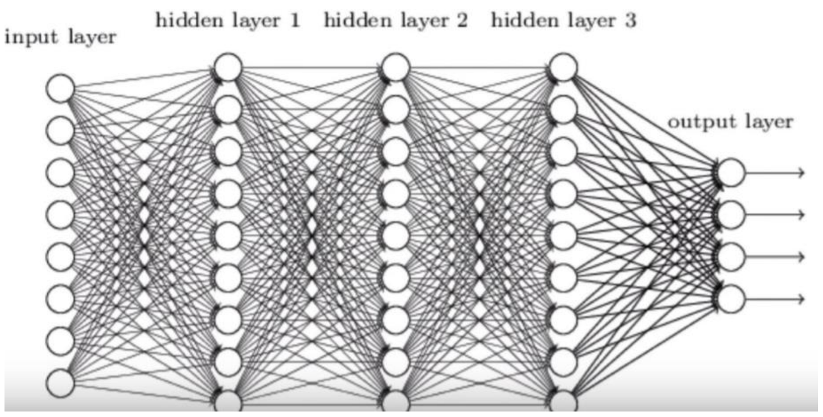
TABLE OF CONTENTS

1. NEURAL NETWORKS

1.1 NEURAL NETWORK BASIS

Neural networks are computer systems that can be modeled to create and predict almost anything. These networks have recently come back in mainstream applications rapidly. One might be using a network without even realizing; Spotify to listen to music, using artificial intelligence in cellphones to check the weather, or biometric recognition. In figure 1 below, a simple diagram of layers and nodes is shown.

Figure 1 – Multi-layer network

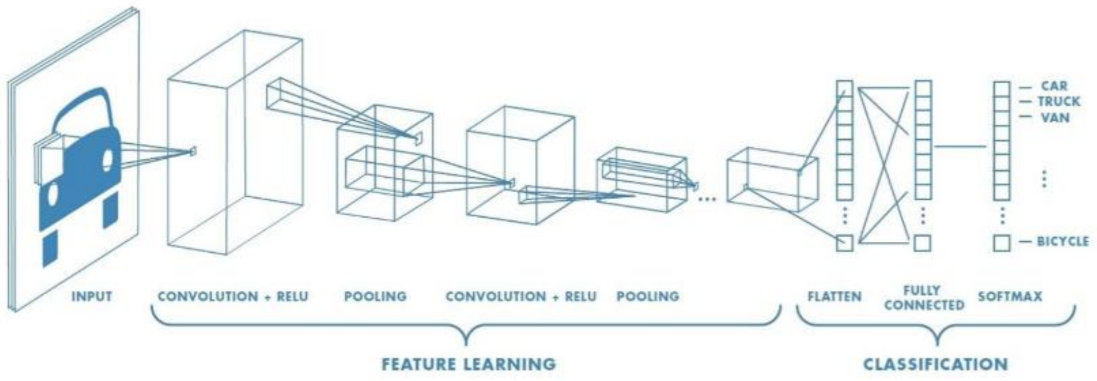


The basis of a neural network is very similar to neurons in one's brain. Just as neurons are connected to one another sending electro chemical signals to the brain, a neural network uses weights that communicate with each layer to get an output.

1.2 CONVOLUTIONAL NEURAL NETWORK

The notable difference in a convolutional neural network (CNN) from other networks is the network's ability to communicate with itself. The neurons are set up in layers, and each layer takes data from previous neurons or nodes, which in turn are being sent to the layer ahead. As seen in figure 1, a network out in the real world can become very complicated very quickly. Most networks that are used professionally contain hundreds if not thousands of nodes and layers. In this pi project, the CNN is used for image processing. The CNN picks multiple features from the input images to recognize, which in this case are digits, and uses the nodes to predict the correct output. In figure 2 below, the broad outline of CNN is shown. CNN's use filters to specify certain regions on the image. These are shown in figure 2. There are multiple filters that are used, and several different types of 'pooling' used which take small slices of bigger arrays that make up the image to have smaller amounts of data.

Figure 2 – Feature learning and classification in CNN



2. HANDWRITING SYSTEM

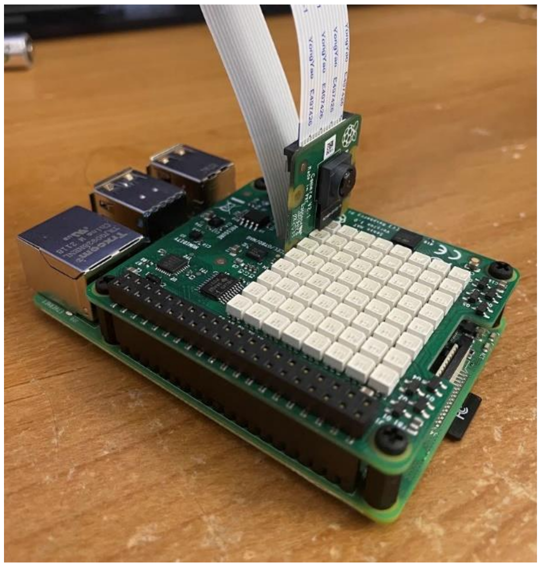
2.1 HARDWARE

The hardware for the physical system includes the raspberry pi 4 with at least 4 gigabytes of memory. Wi-Fi and Bluetooth need to be included in the Pi model for ease but there are work arounds. A LED sensehat is also required which will be plugged on top of the pi and secured with screws. The sensehat has many features such as gyroscope, accelerometer, magnetometer, temperature, barometric pressure sensor and a humidity sensor. A Raspberry Pi camera will be necessary, and it can be easily plugged in using a connector. Final components will include a SD card to flash the Raspbian operating system and micro-HDMI cable, along with external screen an external USB mouse/keyboard.

2.2 RASPBERRY PI SET UP

The pi will require the Raspbian image downloaded onto the SD card which will then be flashed onto the pi itself. Once the image is flashed, the pi software will need to be updated and upgraded via Linux command prompts. These commands include sudo apt update/ sudo apt upgrade. There are two options to do this, either on the physical pi itself using an external monitor/ keyboard or by SSH to communicate with the pi via external computer. After this initial pi setup, free software such as VNC viewer can be used to stream the pi onto another computer. This will allow the pi to be acting on its own. A new directory is set up within the pi and all required libraries are installed. These libraries include TensorFlow, Matplotlib, NumPy, OpenCV, etc. Some of the packages and libraries need to be updated with pip commands. Once the pi is set up, next steps can be achieved. Figure 3 below shows the finished Pi.

Figure 3 – Raspberry Pi with all components

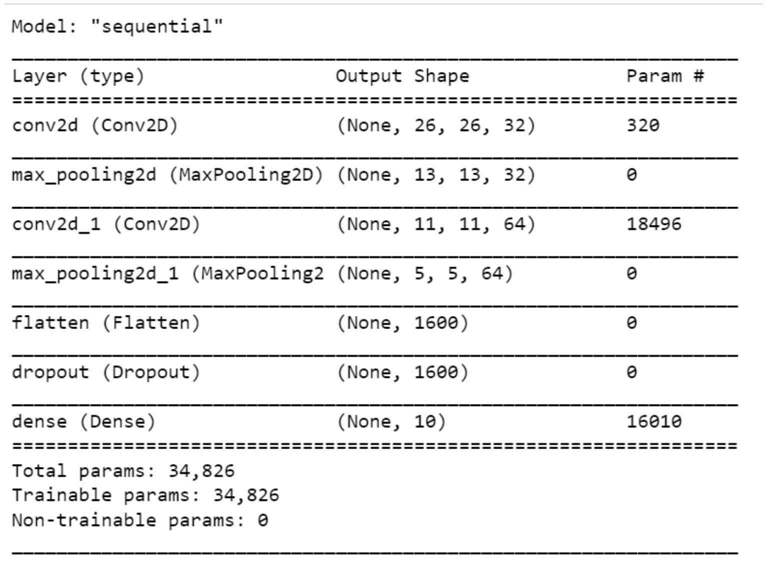


3. NEURAL NETWORK

3.1 CREATING THE MODEL

To train the handwriting digit system that is being created for this project, data samples will be used from MNIST. MNIST provides all the samples that will be used for training the network. Since this project makes use of a convolutional neural network, this will need to be specified when using Keras. The difficult part of the training will be to work through all the data and give the network proper parameter sizes. There also needs to be a dropout in the code to prevent overtraining and have the network freely create accurate predictions. The training can be done either on the pi or another computer. The pi will take quite a while to create the model while the computer will be quicker. The MNIST data can be loaded with a simple load command. Training will be done using python and Keras. ReLu will be the activation functions except the last which will be a SoftMax function. Sample of the Keras output is shown below.

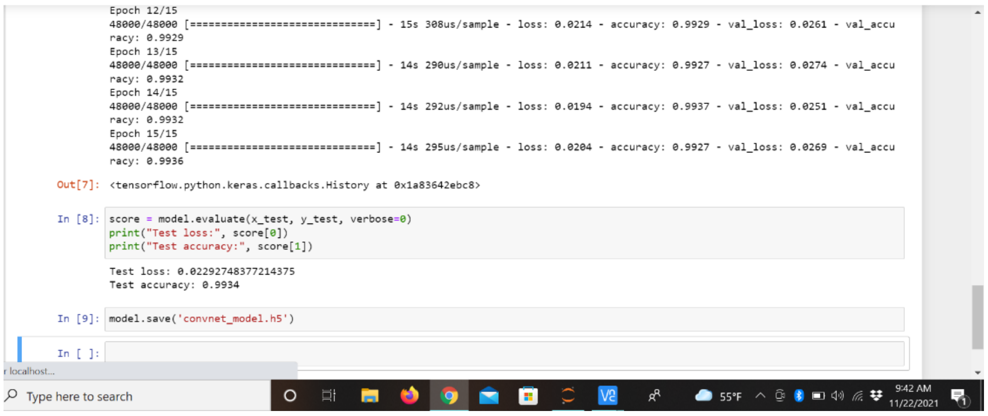
Figure 4 – Keras Output



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dropout (Dropout)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 10)                16010
=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

The trainable parameters equal 34,826. After initializing an appropriate number of epochs along with the correct batch size, the training can be run. A screenshot of the training can be seen below in figure 5.

Figure 5 – Training With MNIST



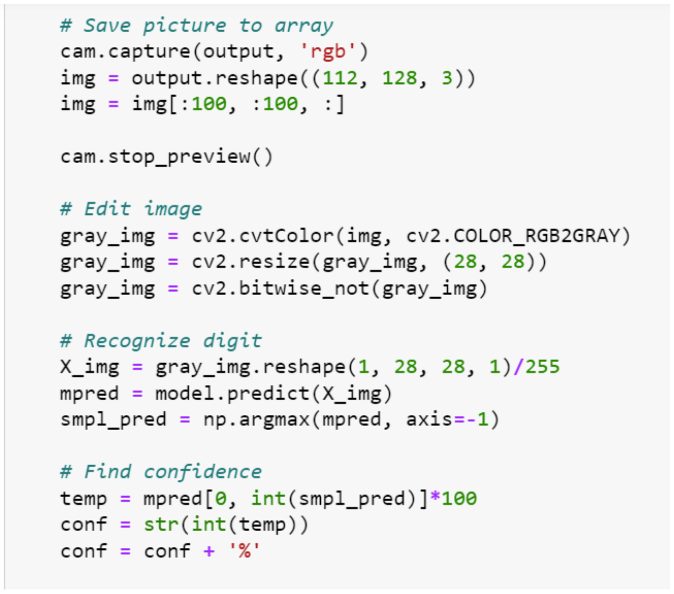
```
Epoch 12/15
48000/48000 [==============================] - 15s 308us/sample - loss: 0.0214 - accuracy: 0.9929 - val_loss: 0.0261 - val_accu
racy: 0.9929
Epoch 13/15
48000/48000 [==============================] - 14s 290us/sample - loss: 0.0211 - accuracy: 0.9927 - val_loss: 0.0274 - val_accu
racy: 0.9932
Epoch 14/15
48000/48000 [==============================] - 14s 292us/sample - loss: 0.0194 - accuracy: 0.9937 - val_loss: 0.0251 - val_accu
racy: 0.9932
Epoch 15/15
48000/48000 [==============================] - 14s 295us/sample - loss: 0.0204 - accuracy: 0.9927 - val_loss: 0.0269 - val_accu
racy: 0.9936

Out[7]: <tensorflow.python.keras.callbacks.History at 0x1a83642ebc8>

In [8]: score = model.evaluate(x_test, y_test, verbose=0)
        print("Test loss:", score[0])
        print("Test accuracy:", score[1])

        Test loss: 0.02292748377214375
        Test accuracy: 0.9934

In [9]: model.save('convnet_model.h5')

In [ ]:
```

Figure 3 is showing training with 15 epochs or iterations. The test accuracy is 99%. The code is created, and the convolutional model will be created with an accompanying h5 file which contains the weights for testing.

3.2 DATA AND ANALYSIS

Since the test will be done on the pi, the model that was created from training the network also needs to be on the pi. This can be easily moved using any file transfer software. Once the pi has the model, code needs to be created to read the model, gather the weights, use the camera to read in the test image and then output the correct digit on the sensehat. Since the pi is not a supercomputer, the image resolution will be lowered to adequately be compatible with the hardware on the pi. The tricky part is to create the test image in the form of the MNIST samples. The MNIST samples are contain dark backgrounds with white digits, while the pi camera takes regular pictures. This can be implemented in the python testing code to invert the picture taken. Some of the sample code is included below in figure 6.

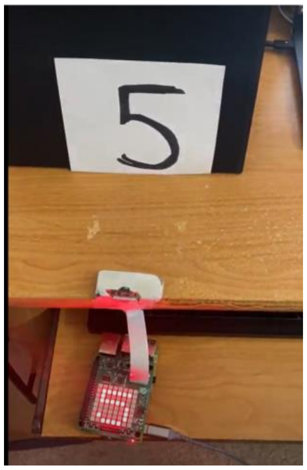
Figure 6 - Sample code used to change input image



```
# Save picture to array
cam.capture(output, 'rgb')
img = output.reshape((112, 128, 3))
img = img[:100, :100, :]

cam.stop_preview()

# Edit image
gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
gray_img = cv2.resize(gray_img, (28, 28))
gray_img = cv2.bitwise_not(gray_img)

# Recognize digit
X_img = gray_img.reshape(1, 28, 28, 1)/255
mpred = model.predict(X_img)
smpl_pred = np.argmax(mpred, axis=-1)

# Find confidence
temp = mpred[0, int(smpl_pred)]*100
conf = str(int(temp))
conf = conf + '%'
```

Figure 2 is code that is used for changing the color of the input test image. The image is then given lower resolution and converted into an array. Once the code is written it can be saved in the same directory where the python libraries were installed. The code is ready to run and be tested against physical handwritten digits. When the code is run, the camera takes the picture and writes out the output on the sensehat. The data to test was four input digits written on a white piece of paper with the digits 0, 1, 5, and 8. The pi recognized each digit correctly with varying amounts of confidence. The output should look like figure 7 which is below.

Figure 7 – Sensehat Output



4. CONCLUSION

The project realistically replicates professionally used libraries and programs like Keras and TensorFlow to create the machine learning algorithm. Using a Pi provides many freedoms not found in other boards, as well as the implementation of many attachments like the camera and senshat. The input samples are taken from the MNIST train data which includes hundreds of handwritten digits. The neural network that is being used is a convolutional network and the accuracy should be close to 99 percent accurate. Two codes are needed, one to create the network and another to test on the pi. After the setup of the pi, it will be able to run only a power source. For the pi to correctly test the image, the image needs to be almost perfectly set up as any disruption can alter the output reading and the correct output should be printed on the sensehat. As can be seen from the project, running code on the pi is feasible, however limitations in hardware are already obvious. The test images can be finicky and need to be positioned almost perfectly for the pi to correctly recognize.