Real-Time HandWriting Recognition Using TensorFlow and Raspberry PI

Prepared by:

Jason Greaves-Taylor

In partial fulfillment of the requirements for CPE 4903

Instrutor: Hai Ho

Data: December 08, 2021

Kennesaw State University

Abstract

This paper documents work done to create an embedded system running Tensorflow 2 and OpenCV in order to predict and classify hand writen numeral characters via image capture. The overall purpose of this work was to show that a small, low powered system could provide accurate, robust results using a trained convoluted neural network in real time. The hardware items used were a Raspberry Pi 3 B+, a SenseHat, and a PiCamera. The software used were Tensorflow and OpenCV via Python.

The results showed that such a system could be created and operated while providing reliable and accurate results. The system captures images via camera, converts them into an OpenCV object, filters them to be in a negative binary format, then passes that data through a pre-trained convoluted neural network. The network then predicts the output and gives a confidence interval above 50%, usually above 70% in most instances, as to the numeral that was captured with a ~90% accuracy rating.

Introduction

Getting Started

The Real-Time Handwriting Recognition System was developed as an embedded system which had three physical components and two code processes. The system took in images using a PiCamera attached to a Raspberry Pi 3 B+. The PI would then run a pretrained neural network for the purposes of classifing the objects in the images that were captured. Once classifcation was completed, it would return a confidence value to the Linux Terminal and the SenseHat's LED will display the number that it predicts is in the image. The system will take a picture every time the user presses the "C" key and quits the process when the user presses the "Q" key.

Theory

Neural Networks and Machine Learning is the process of using the mathematical concept of systems of equations also known as linear algebra to approximate the concept of learning as experienced by humans. The derivations used to create the equations are pretty mathematically intensive but given how complex the human brain is this is to be expected when trying to apporoximate it on a number crunching machine.

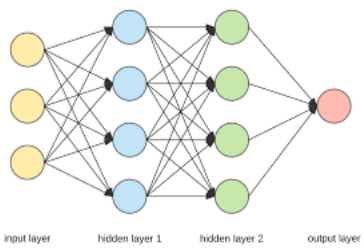

Figure 1: Image of a typical Neural network

The above image is an example the typical neural network architecture. In the most basic neural network there is an input layer where the intial data is introduced the network, a hidden layer which would the process the data finally an output layer which would spit out result. The above picture is slightly more complicated than the most basic neural network as it has 2 hidden layers where the data being introduced would be processed twice as opposed just once

before going to output. As the complexity of the opperation that you wish to have your neural network tackle increases so does you number of hidden layers until you get to applications like DALLE which according to nimblebox.ai has upwards of 64 hidden layers.

Typically the applications of neural networks fall under three major categories. These categories are: Prediction, Classification and Generation. For this project our neural network fell under the Classification category and we used type of Neural Network called a Convolutional Neural Network (CNN). This neural network is widely used for image/video detection and is known for being pretty computationally efficient when compared to other types of neural network such as the Multi-Layer Perceptrons (MLP).

Procedure

Intial Set up

The very first steps for creating the software side of the system used was to prepare the Raspberry Pi by downloading the dependencies, Tensorflow 2, NumPy, OpenCV as well as the SenseHat module. Additionally, one of the steps taken was to downgrade the Python version present on the Raspberry Pi to Python version 3.7.12 because as of November 2022, tensorflow does not work with the most recent version of Python (Python 3.9.2). Once these intital steps were complete we moved on to building the model.

Creating the Model

The code below was used to create the CNN model used to classify the handwritten numbers.

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
from sens_hat import SenseHat
MySH = SenseHat()


# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

# Load the data and split it between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")


# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)



model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
```

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()



batch_size = 128
epochs = 15

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

```
batch_size = 128
epochs = 15

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)


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

model.save('/home/pi/project/handrecog/keras_convnet_adam')
```

Figure 2: Code used for creating the model

The data that was loaded came from an online database called the Modified National Institue of Standards and Technology (MNIST). This database is a large number of handwritten digits that was specifically created to be used in the training of image processing for recognizing handwritten numbers. By loading the database in to the model we don't have require to download all the labeled images so we can distribute our system easily if we so desired. Additionally, this made it easier to train our system on our Raspberry Pi, a system with limited resources.

Using The Model

In addition to the code for creating the model another script was created for the purpose controlling system by actively detecting digits using the Raspberry Pi Camera and using the model to compare the images to the labeled data. Once the model predicted a potential digit, the predicted digit would be displayed on the SenseHat module.

```
from sense_hat import SenseHat # SenseHat
from picamera import PiCamera # Camera
from skimage import img_as_ubyte
from skimage.color import rgb2gray
import time
from time import sleep
import datetime
import argparse
import imutils
from imutils.video import VideoStream
import os, cv2, itertools # cv2 -- OpenCV
import tensorflow as tf # Tensorflow
import numpy as np # Numpy

# Assigning SenseHat and PiCamera
sense = SenseHat()  # Initialize SenseHat
#camera = PiCamera() # Initialize Camera
#camera.color_effects = (128,128) # Set camera to black and white mode

#Scaled&Thresh-Pilled
ROWS = 28
COLS = 28

#Load model
print('Loading Neural Network...\n')
model = tf.keras.models.load_model('/home/pi/project/handrecog/keras_convnet_adam/')
#model = tf.keras.models.load_weights('keras_convent_adam')

# Construct arguments for PiCamera.
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--picamera", type=int, default=-1, help="Should the PiCam be used?")
args = vars(ap.parse_args())

# Initialize VideoStream from PiCamera
vidStream = VideoStream(usePiCamera=args["picamera"] > 0).start()
time.sleep(2.0)
```

```
time.sleep(2.0)

# Read Image Function Using OpenCV and Create Binary Output
def read_negative_image(sample_img):

        print('Applying filter to image...')

        #Reads image in using OpenCV grayscale the converts to uint8
        img_gray =  rgb2gray(sample_img)
        img_gray8 = img_as_ubyte(img_gray)

        #Binary Filter and Scaling using the OTSU method
        (thresh, img_bw) = cv2.threshold(img_gray8, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        img_resized = cv2.resize(img_bw,(28,28))

        #Invert the colors
        img_bw_invert = 255 - img_resized
        img_final = img_bw_invert.reshape(1,ROWS,COLS,1)
        #Write and Show Filtered Image (For Testing Purposes)
        #print('Filter Applied!')
        #cv2.imwrite('samp_BW.png', sample_img)
```

```
def read_negative_image(sample_img):

        print('Applying filter to image...')

        #Reads image in using OpenCV grayscale the converts to uint8
        img_gray =  rgb2gray(sample_img)
        img_gray8 = img_as_ubyte(img_gray)

        #Binary Filter and Scaling using the OTSU method
        (thresh, img_bw) = cv2.threshold(img_gray8, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        img_resized = cv2.resize(img_bw,(28,28))

        #Invert the colors
        img_bw_invert = 255 - img_resized
        img_final = img_bw_invert.reshape(1,ROWS,COLS,1)
        #Write and Show Filtered Image (For Testing Purposes)
        #print('Filter Applied!')
        #cv2.imwrite('samp_BW.png', sample_img)
        #cv2.imshow('Binary Input', sample_img)

        run_network(img_final)

# Predict the classification of the image.
def run_network(img_final):

        #Array printout of each possible digit.
        answer = model.predict(img_final)
        print(answer)

        #Digit with the greatest possibility is the prediction
        prediction = answer[0].tolist().index(max(answer[0].tolist()))
        print('The predicted digit is: ', prediction)

        #Get confidence value
        confidences = np.squeeze(model.predict_proba(img_final))
        topClass = np.argmax(confidences)
        topConf = confidences[topClass]
```

```
        run_network(img_final)

# Predict the classification of the image.
def run_network(img_final):

        #Array printout of each possible digit.
        answer = model.predict(img_final)
        print(answer)

        #Digit with the greatest possibility is the prediction
        prediction = answer[0].tolist().index(max(answer[0].tolist()))
        print('The predicted digit is: ', prediction)

        #Get confidence value
        confidences = np.squeeze(model.predict_proba(img_final))
        topClass = np.argmax(confidences)
        topConf = confidences[topClass]
        print(topClass)
        print(confidences)

        #Update Sensehat LED display when over the 50% threshold
        if topConf > .5:

                print('Confidence is: ', topConf)
                ledDigit = str(prediction)
                sense.show_letter(ledDigit)

############################################################
# Main Function - Takes pictures when 'C' is pressed.
#                 Quits function when 'Q' is pressed.
#                 Waits for keyboard input to break loop.
############################################################

def main():
        #Updates frames from VideoStream to Window Preview
        while True:
```

```
        print(confidences)

        #Update Sensehat LED display when over the 50% threshold
        if topConf > .5:

                print('Confidence is: ', topConf)
                ledDigit = str(prediction)
                sense.show_letter(ledDigit)

#########################################################
# Main Function - Takes pictures when 'C' is pressed.
#                 Quits function when 'Q' is pressed.
#                 Waits for keyboard input to break loop.
#########################################################

def main():
        #Updates frames from VideoStream to Window Preview
        while True:
                try:

                        #Resizes image input from VideoStream to 800px wide, auto adjust height.
                        frame = vidStream.read()
                        frame = imutils.resize(frame, width=400)

                        ##Creates a timestamp on the image.
                        #timestamp = datetime.datetime.now()
                        #ts = timestamp.strftime("%A %d %B %Y %I:%M:%S%p")
                        #cv2.putText(frame, ts, (10, frame.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX,
                        #       0.35, (0, 0, 255), 1)

                        #Show frame in Window Preview
                        cv2.imshow("Window Preview", frame)
                        key = cv2.waitKey(1) & 0xFF

                        #Key catch statement. Refer to Main Function comment for controls.
                        if key == ord("q"):
                                break
```

```
                        #Resizes image input from VideoStream to 800px wide, auto adjust height.
                        frame = vidStream.read()
                        frame = imutils.resize(frame, width=400)

                        ##Creates a timestamp on the image.
                        #timestamp = datetime.datetime.now()
                        #ts = timestamp.strftime("%A %d %B %Y %I:%M:%S%p")
                        #cv2.putText(frame, ts, (10, frame.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX,
                        #       0.35, (0, 0, 255), 1)

                        #Show frame in Window Preview
                        cv2.imshow("Window Preview", frame)
                        key = cv2.waitKey(1) & 0xFF

                        #Key catch statement. Refer to Main Function comment for controls.
                        if key == ord("q"):
                                break
                                #KillAllWindows.exe
                                cv2.destroyAllWindows()
                                vidStream.stop()
                        elif key == ord("c"):
                                cv2.imwrite("number.jpg", frame)
                                sample_img = cv2.imread("number.jpg")
                                read_negative_image(sample_img)
                        else:
                                pass
                except KeyboardInterrupt:
                        #KillAllWindows.exe
                        cv2.destroyAllWindows()
                        vidStream.stop()

if __name__=="__main__":
        main()
```

Figure 3: Code used for controlling

This code had three main functions. The first function takes control of the PiCamera by creating a live video preview in a window. This live video allows the user so line up the camera properly with the hand written digit to get a clear picture. To capture the image for the model, the program waits in a while loop for the user to input "C" using the keyboard and when the user wishes to stop they input "Q" on the keyboard.

The second function was to take the image and reverse it from black text on a white background to white text on a black background. This function was added because the labeled data is in the form of white text on a black background. It works by saving the image into memory and doing several transformations. Including the res

The last function is use to call the model. With the image processed, it is fed to the model in the model.predict function to use the pre-trained weights created in the previous script used for train the model. The predict fuction will return a value if the threshold is greater than 50%, the SenseHat's Led display will be actuated to display the predicted digit.

Design

This system was designed to be pretty simplistic as it only required three hardware compontents. These compontents are realitve easy to come by and can be sourced at Microcenter or Amazon. Once again these components are the Raspberry PI 3 Model B+, the PiCamera and the SenseHat. Below is a picture of the system that was used to test this system.

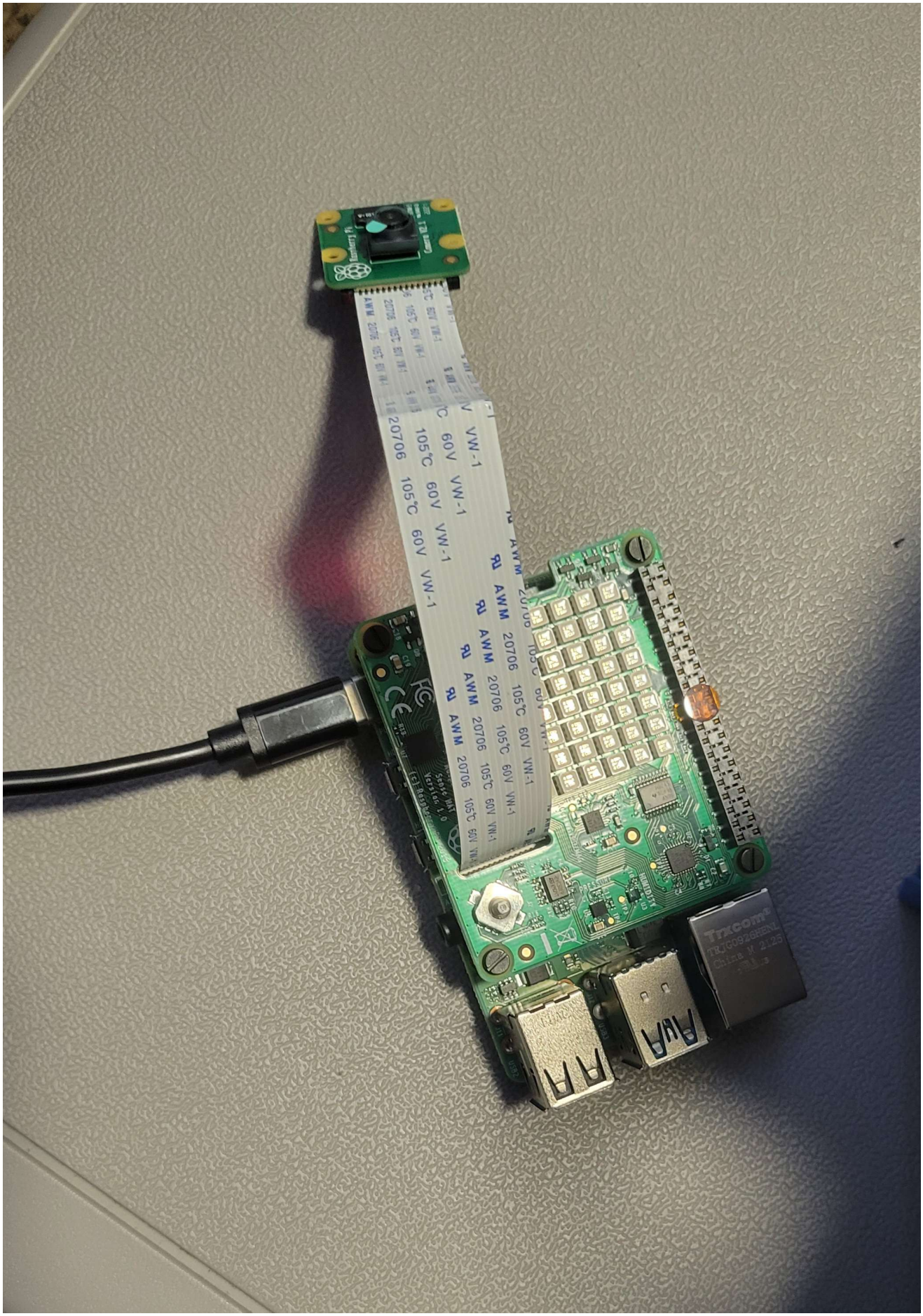
AWM 20706 105°C 60V VW-1

Figure 4: Image of completed system

Hardware

The Raspberry Pi 3 B+ has a Cortex - A53 ARM processor running at 1.4 GHz, and 1GB of DDR2 RAM and an extended 40- pin GPIO header. The SenseHat was attached to the pi using the GPIO header which allows for the use of the LED display. The Pi camera was attached to the PI with a UxCell ribbon cable to the Pi's camera port.

Design Limitations

While the system works pretty well in ideal situations, in non ideal situations the system struggles. It really struggles if there are any shadows on the page. This is because the images in the database that it was trained from were taken in perfect conditions which means for you to use the system effectively you need a clean image.

To help combat this the preview window in the live video stream from the camera was used in conjuction with requiring the user to confirm that they wanted to capture the image. This helps the user to try to take as clean an image as possible with as few shadows.

Another design limitation is the resources avalible on the raspberry pi for training. Whilst it is possible to train the model on the pi, it takes an incredibly long time to complete that training. So to combat this in future versions, if you had to retrain the model, I would recommend that training is done on a laptop or a desktop computer. This would allow for the training to be completed more quickly.

Data

Data

```
The following data was taken from both the training and testing of the model.It shows the accuracy and loss of both sets of
data. The model was trainned using 2 hidden layers of sizes 32 and 64 respectively. Additionally, Each hidden layer had a ReLU
activation fuction. The outputs was then flattened and with a random drop out of 50 percent to prevent overfitting of the data
during each interation. Finally, the outputs at the very end were detemined using a 'softmax' activation fuction.

The model had 15 epochs with a 128 batches per epoch. This means that the model lasted for apporixmately 1920 iterations. Since
the goal of this system was the classification of images into multiple categories we used the categorical_crossentropy with a
the "Adam" optimizer. Below is the results of the trainning of my model.
```

```
(env) pi@greavestaylorpi:~/project/handrecog $ python3 keras-convet.py
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
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
_________________________________________________________________
2022-12-07 18:27:15.623493: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registered 1)
2022-12-07 18:27:15.638685: W tensorflow/core/platform/profile_utils/cpu_utils.cc:112] Failed to find bogomips or clock in /proc/cpuinfo; cannot determine CPU frequency
Epoch 1/15
422/422 [==============================] - 163s 380ms/step - loss: 0.7769 - accuracy: 0.7697 - val_loss: 0.0778 - val_accuracy: 0.9795
Epoch 2/15
422/422 [==============================] - 161s 382ms/step - loss: 0.1204 - accuracy: 0.9652 - val_loss: 0.0559 - val_accuracy: 0.9847
Epoch 3/15
422/422 [==============================] - 161s 382ms/step - loss: 0.0835 - accuracy: 0.9747 - val_loss: 0.0471 - val_accuracy: 0.9870
Epoch 4/15
422/422 [==============================] - 162s 383ms/step - loss: 0.0672 - accuracy: 0.9790 - val_loss: 0.0395 - val_accuracy: 0.9887
Epoch 5/15
422/422 [==============================] - 161s 382ms/step - loss: 0.0616 - accuracy: 0.9806 - val_loss: 0.0379 - val_accuracy: 0.9907
Epoch 6/15
422/422 [==============================] - 161s 383ms/step - loss: 0.0538 - accuracy: 0.9834 - val_loss: 0.0362 - val_accuracy: 0.9905
Epoch 7/15
422/422 [==============================] - 161s 382ms/step - loss: 0.0509 - accuracy: 0.9833 - val_loss: 0.0326 - val_accuracy: 0.9908
Epoch 8/15
422/422 [==============================] - 161s 382ms/step - loss: 0.0445 - accuracy: 0.9859 - val_loss: 0.0307 - val_accuracy: 0.9903
Epoch 9/15
422/422 [==============================] - 162s 383ms/step - loss: 0.0424 - accuracy: 0.9866 - val_loss: 0.0334 - val_accuracy: 0.9917
Epoch 10/15
422/422 [==============================] - 161s 382ms/step - loss: 0.0414 - accuracy: 0.9865 - val_loss: 0.0289 - val_accuracy: 0.9927
Epoch 11/15
422/422 [==============================] - 161s 382ms/step - loss: 0.0365 - accuracy: 0.9882 - val_loss: 0.0275 - val_accuracy: 0.9933
Epoch 12/15
422/422 [==============================] - 162s 383ms/step - loss: 0.0341 - accuracy: 0.9891 - val_loss: 0.0300 - val_accuracy: 0.9912
Epoch 13/15
422/422 [==============================] - 161s 382ms/step - loss: 0.0337 - accuracy: 0.9889 - val_loss: 0.0304 - val_accuracy: 0.9915
Epoch 14/15
422/422 [==============================] - 162s 383ms/step - loss: 0.0323 - accuracy: 0.9892 - val_loss: 0.0280 - val_accuracy: 0.9922
Epoch 15/15
422/422 [==============================] - 161s 382ms/step - loss: 0.0307 - accuracy: 0.9900 - val_loss: 0.0275 - val_accuracy: 0.9920
Test loss: 0.024310391396284103
```

Figure 5: Image of the results of training my model on the pi

Conclusion

In conclusion, The Real-Time Handwriting Recognition system worked pretty well and had a very high level of accuracy and was able to work in many locations. While there were limitations to consider, especially what type of lighting you have in the environment that you are trying to use the system in. However, once those limitations have been taken into consideration, being able to run this system on portable system with limited resources like a Raspberry Pi proves the potential for neural networks to used in everyday situations by the average user.

References

- DALL E 2: AI That Can Render Masterpieces from Text! (https://nimblebox.ai/blog/dalle-2-openai)
- (https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/)

  "Neural Network: Architecture, Components & Top Algorithms." UpGrad Blog, 22 Sept. 2022, (https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/)https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/ (https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/) Used for figure 1

In [ ]: