KENNESAW STATE UNIVERSITY

Real-Time Handwriting Recognition Using Tensorflow and Raspberry Pi

Jorge Diaz Rodriguez

28 November 2022

Contents

1 Team Members:

- Jorge Diaz

2 Abstract

Real-Time Handwriting Recognition project prompts students to make a program that recognizes a digit when a camera connected to a raspberry pi is shown a digit on a white sheet of paper. The objective was accomplished by constructing a convolutional neural network(CNN). The CNN was constructed using the API Keras. Open CV library was then used to establish an interface between the camera and the raspberry pi.

3 Introduction

3.1 History

Neural Networks were first introduced as a model to explain how neurons in the brain function, and in 1943 it was portrayed as a simple electrical circuit by neuro-physiologist Warren McCulloch. As research on the topic advanced in 1959 at Stanford Bernard Widrow and Marcian Hoff devloped the first Neural Network MADALINE to be applied to a real world problem it had an adaptive filter that could filter echo from phone lines. Although standard neural networks were revolutionary, there was a cap to the precision that could be acquired through a standard neural network. Due to this neural networks that designed with different layers started to be made like the Neocognitron which was developed in the 1980's which introduced concepts such as feature extraction, pooling layers and the use of convolution in neural networks, and proposed classification and recognition. This was the beginning what would come to be known as a Convultional Neural Network which has the ability to take an input and assign weights and biases to aspects of the input to be able to differentiate between it and other inputs.

3.2 Objective

The objective of this paper is to create a convolutional neural network that can take an input from a camera seeing a paper with a number on it which then has to send an input to the CNN which should be able to recognize the numbers shown to it, and display it on a sense hat which is an a add on for the raspberry pi that gives it array sensing capabilities.

4 Bill of Materials

- Raspberry Pi
- Raspberry Pi Sense HAT
- Raspberry Pi Camera
- Micro SD Card
- Raspberry Pi Power Source
- Micro HDMI
- Screen with HDMI port

5 Convolutional Neural Network

Convolutional Neural Networks are deep learning that can take input images and assign weights and biases based on aspects of the image.

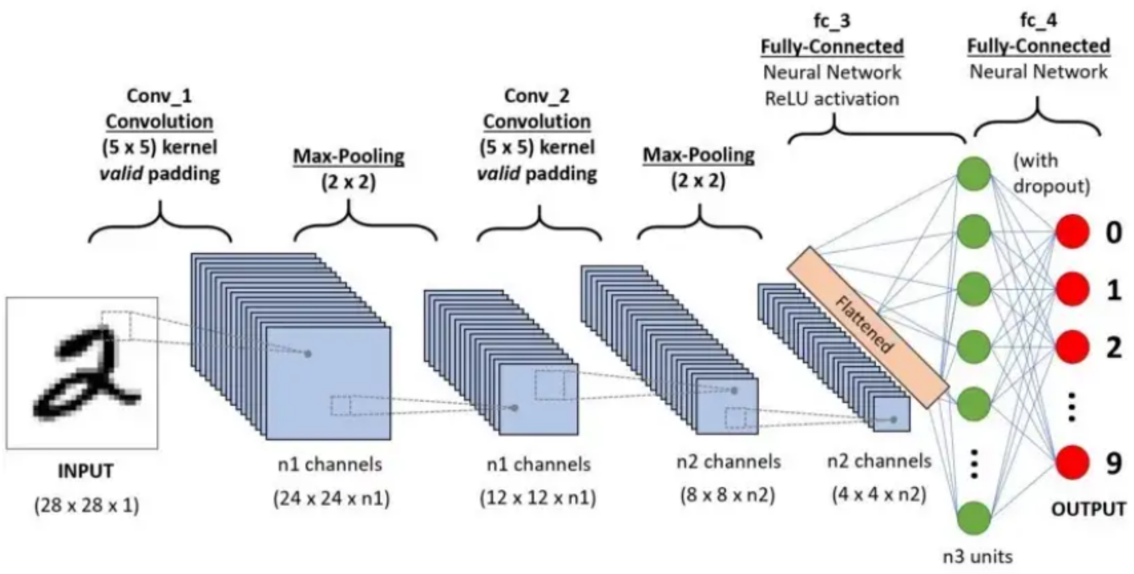
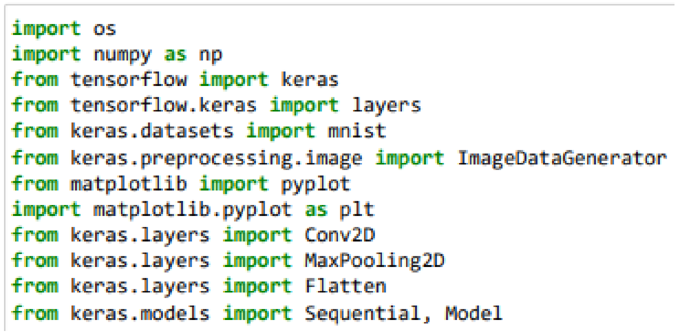


Figure from [8]

A convolutional neural network is generally consists of three different types of layers, convolutional, max pooling and the fully connected layers.

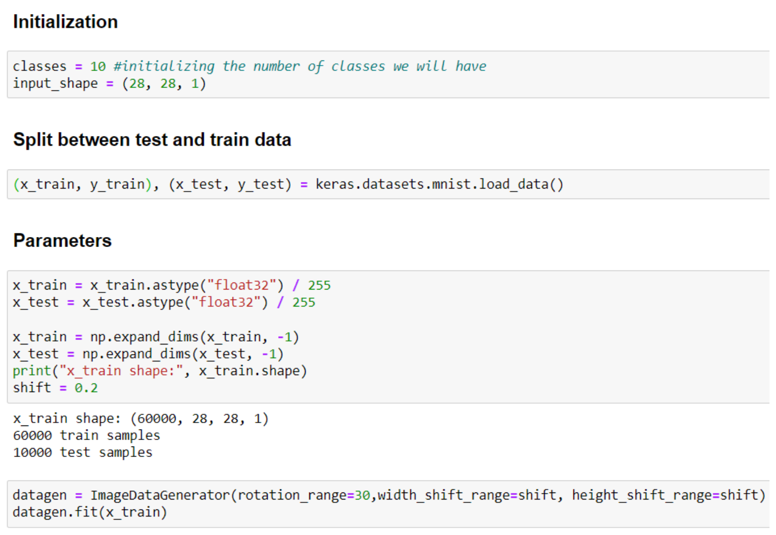
- Convolutional Layer: This layer is essentially what differentiates a Convolutional Neural networks from other Neural Networks, it contains filters that convolve with the image inputted and creates an activation map.
- Pooling Layer: Pooling layers reduce the number of parameters and computation down by reducing the dimensions of the feature maps.
- Fully Connected Layer: This layer takes the result from the convolution and pooling layer process and reaches a classification/decision.

5.1 Code



```
import os
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
from keras.datasets import mnist
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
import matplotlib.pyplot as plt
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.models import Sequential, Model
```

To begin the CNN code the libraries that are going to be used have to be imported which is shown on the image above. The image below shows the variables being initialized split between train and test data. After that the parameters are fixed to be used for the code.



Initialization

```
classes = 10 #initializing the number of classes we will have
input_shape = (28, 28, 1)
```

Split between test and train data

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

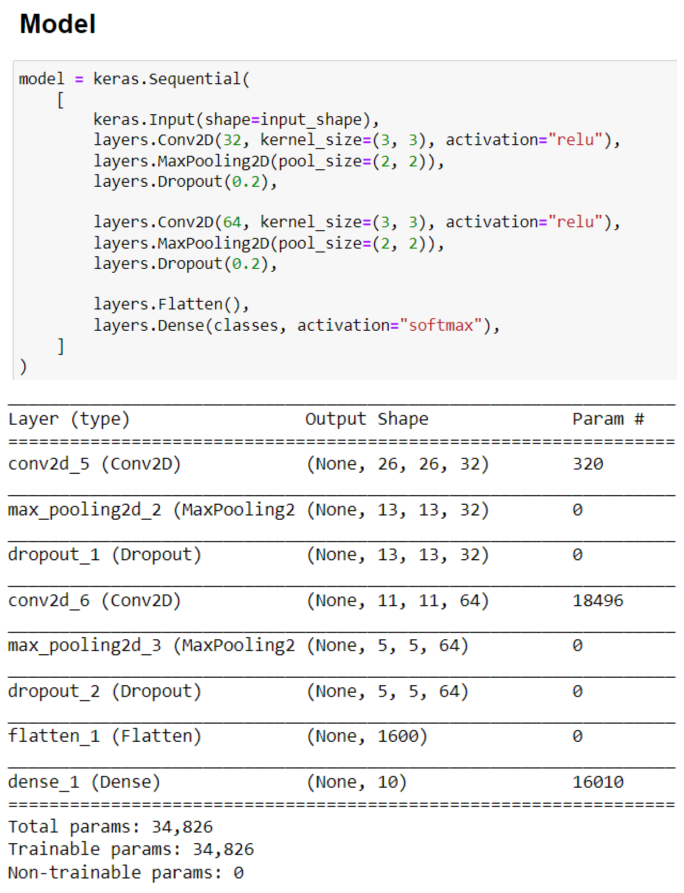
Parameters

```
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
shift = 0.2
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

```
datagen = ImageDataGenerator(rotation_range=30,width_shift_range=shift, height_shift_range=shift)
datagen.fit(x_train)
```

The code below was used to make the CNN, it consists of two convolutional layers, two pooling layers, and the fully connected layer. Below that is the summary of the CNN.

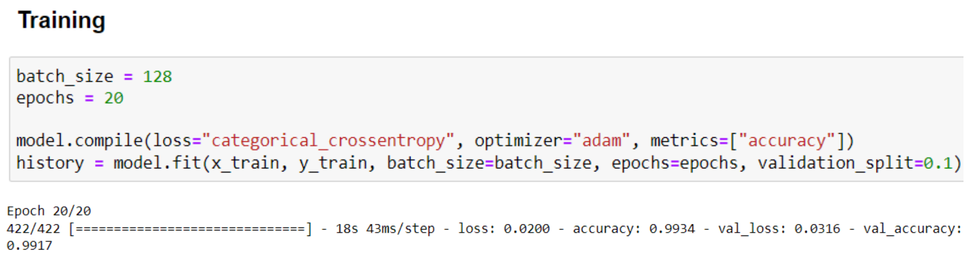


Model

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Dropout(0.2),

        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Dropout(0.2),

        layers.Flatten(),
        layers.Dense(classes, activation="softmax"),
    ]
)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                16010
=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
```

Before, the CNN was constructed. Below is the code that trains the model using the model constructed before with the training variables and testing it with the data that was designated as test data which is where the accuracy also comes from.



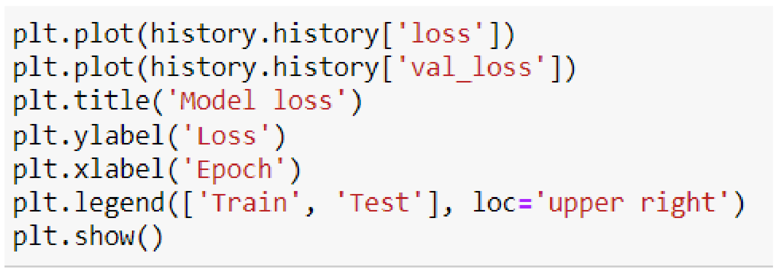
Training

```
batch_size = 128
epochs = 20

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

```
Epoch 20/20
422/422 [==============================] - 18s 43ms/step - loss: 0.0200 - accuracy: 0.9934 - val_loss: 0.0316 - val_accuracy:
0.9917
```

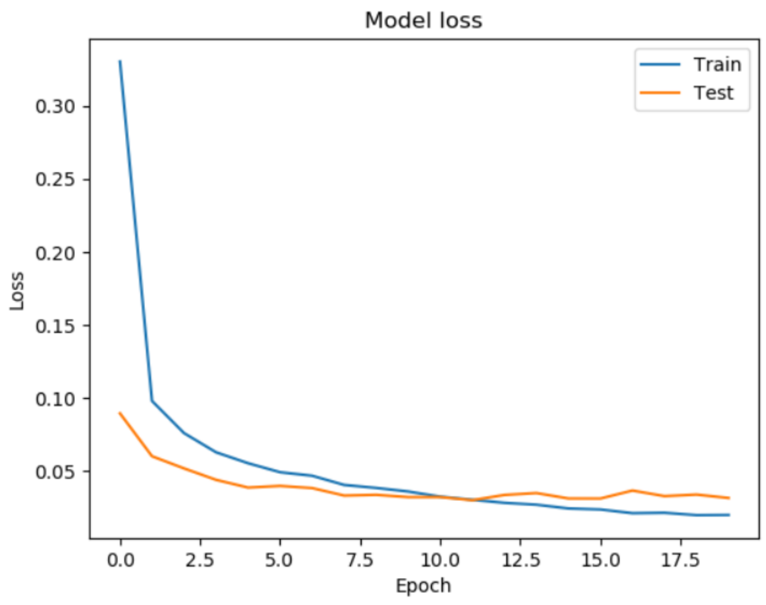
6 Accuracy

The code below takes the loss of the model and plots it.



```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()
```

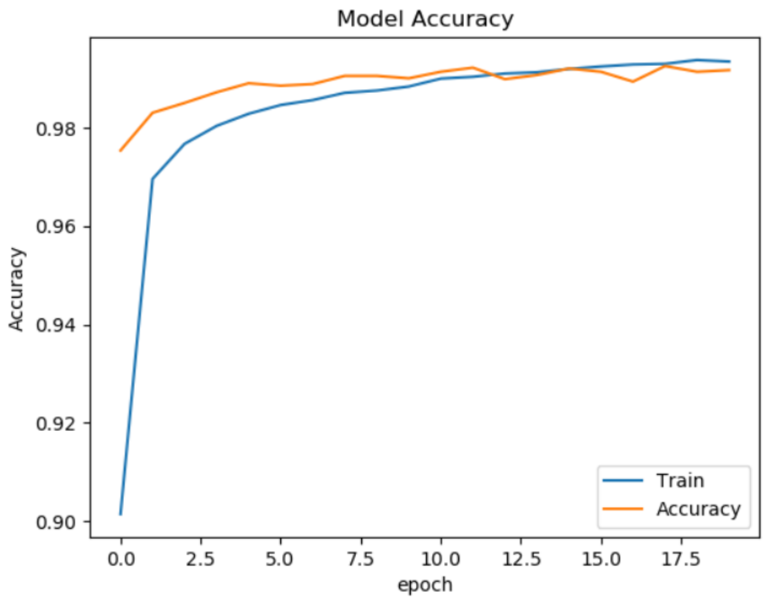
The image below shows the Loss of the model as epochs proceeded which as expected the loss went down as more it proceeded.



Below is the code that plots the accuracy of the model.

```python
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Accuracy'], loc='lower right')
plt.show()
```

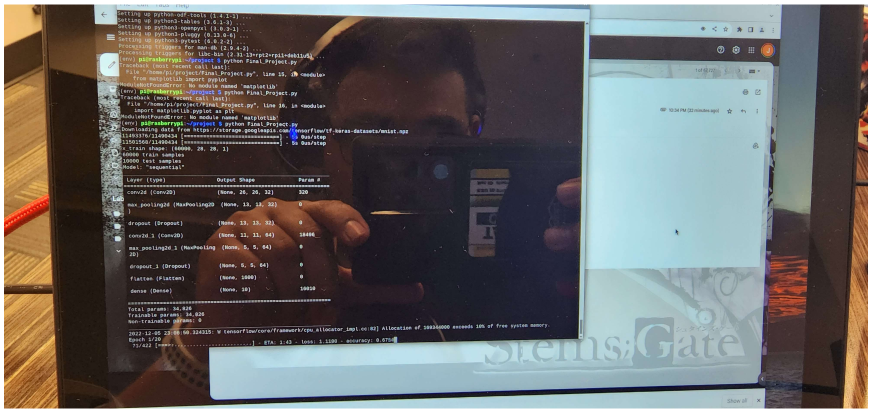
The image below shows the accuracy of the system in comparison to the testing data.



The model should be made on a normal desktop because the raspberry pi will take a long time due to its limited processing speed. To save the model use the code below.

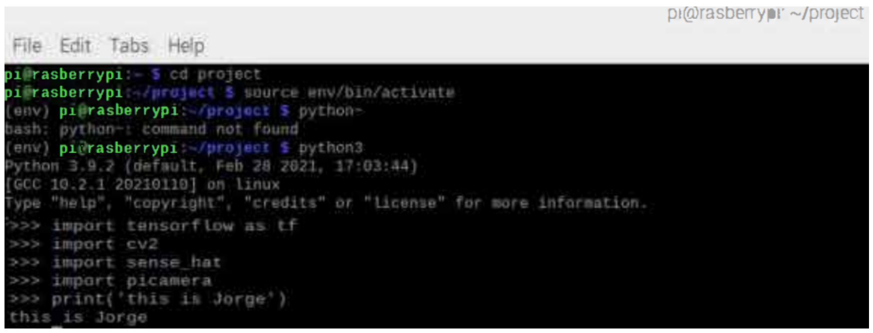
```python
model.save('final_trained_model')
```

The model can also be ran on a raspberry as can be see below.



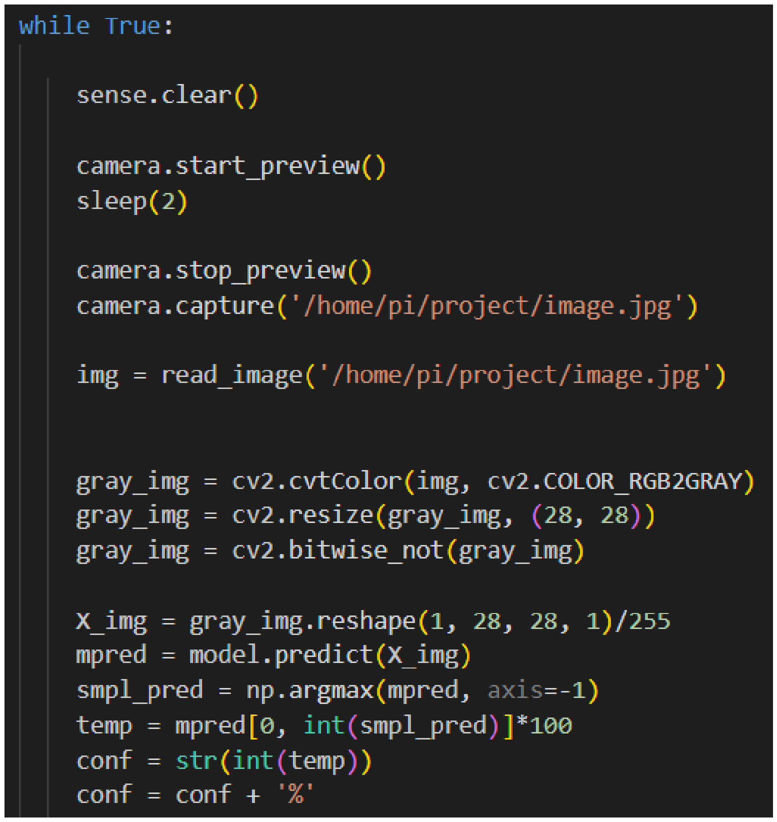
```
Setting up python-odf-tools (1.4.1-1) ...
Setting up python3-tables (3.6.1-3) ...
Setting up python3-openpyxl (3.0.3-1) ...
Setting up python3-pluggy (0.13.0-6) ...
Setting up python3-pytest (6.0.2-2) ...
Processing triggers for man-db (2.9.4-2) ...
Processing triggers for libc-bin (2.31-13+rpt2+rpi1+deb11u5) ...
(env) pi@rasberrypi:~/project $ python Final_Project.py
Traceback (most recent call last):
  File "/home/pi/project/Final_Project.py", line 15, in <module>
    from matplotlib import pyplot
ModuleNotFoundError: No module named 'matplotlib'
(env) pi@rasberrypi:~/project $ python Final_Project.py
Traceback (most recent call last):
  File "/home/pi/project/Final_Project.py", line 16, in <module>
    import matplotlib.pyplot as plt
ModuleNotFoundError: No module named 'matplotlib'
(env) pi@rasberrypi:~/project $ python Final_Project.py
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 5s 0us/step
11501568/11490434 [==============================] - 5s 0us/step
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
 )

 dropout (Dropout)           (None, 13, 13, 32)        0

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 5, 5, 64)          0

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 10)                16010

=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
2022-12-05 23:06:50.324315: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 169344000 exceeds 10% of free system memory.
Epoch 1/20
 73/422 [===>..........................] - ETA: 1:43 - loss: 1.1190 - accuracy: 0.6754
```

To run the model on raspberry pi tensorflow 2, open cv, sensehat, and picamera had to be installed.

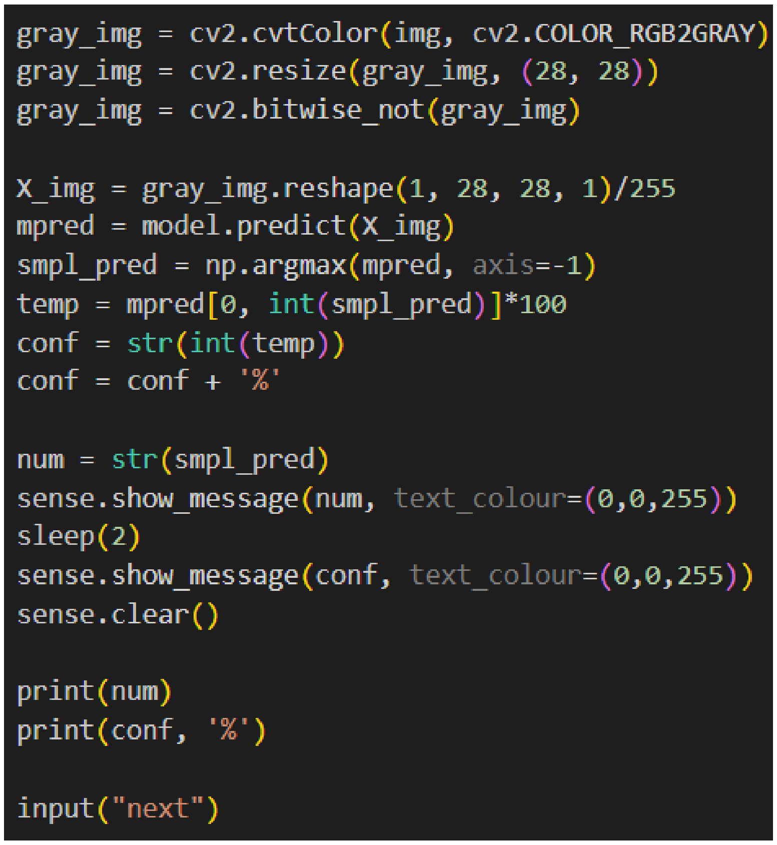


```
pi@rasberrypi:~ $ cd project
pi@rasberrypi:~/project $ source env/bin/activate
(env) pi@rasberrypi:~/project $ python~
bash: python~: command not found
(env) pi@rasberrypi:~/project $ python3
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> import cv2
>>> import sense_hat
>>> import picamera
>>> print('this is Jorge')
this is Jorge
```

Once this was completed the code below was ran to have the raspberry pi's camera read the image taken from the camera. Using this code the sense hat will show the pixels in blue due to using the (0,0,255) which is the color blue in pixels.



```
while True:

    sense.clear()

    camera.start_preview()
    sleep(2)

    camera.stop_preview()
    camera.capture('/home/pi/project/image.jpg')

    img = read_image('/home/pi/project/image.jpg')


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

    X_img = gray_img.reshape(1, 28, 28, 1)/255
    mpred = model.predict(X_img)
    smpl_pred = np.argmax(mpred, axis=-1)
    temp = mpred[0, int(smpl_pred)]*100
    conf = str(int(temp))
    conf = conf + '%'
```



```
gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
gray_img = cv2.resize(gray_img, (28, 28))
gray_img = cv2.bitwise_not(gray_img)

X_img = gray_img.reshape(1, 28, 28, 1)/255
mpred = model.predict(X_img)
smpl_pred = np.argmax(mpred, axis=-1)
temp = mpred[0, int(smpl_pred)]*100
conf = str(int(temp))
conf = conf + '%'

num = str(smpl_pred)
sense.show_message(num, text_colour=(0,0,255))
sleep(2)
sense.show_message(conf, text_colour=(0,0,255))
sense.clear()

print(num)
print(conf, '%')

input("next")
```

7 Conclusion

Neural networks allow for the combination of the ability to process information and create outputs based on the information. Neural Networks have three primary uses recognition, prediction and generation. Neural networks can be combined with existing machines to permit them to handle more ”intelligent” tasks, such as processing unorganized data by segregating and categorizing it.
This project used a Convolutional Neural Network which are network architectures used for deep learning, they primarily are used for finding patterns in images to recognize objects and/or classify/categorize.The MNIST data set was divided into a training and a testing data set. Keras was then used for the training of a Convolutional Neural Network, that had ten classes the numbers zero through nine. After making the neural network, a code was made that allowed for the raspberry pi to read data off of the pi camera and send the data through the CNN model that then allowed for the recognition of number written on a piece of paper. Although the network did not always work it did work more often than not Video. A problem that was encountered in getting the system to run was making sure to have compatible software, due using python 3.9.2 the standard pi camera and sense hat software did not function, until a work around was found. By constructing this project the user can strengthen if not acquire the ability to connect code to hardware, and also strengthen their own coding skills.

8 References

[1] Dr.Hai Ho's class notes

9 Bibliography

[1] https://towardsdatascience.com/convolutional-layers-vs-fully-connected-layers-364f05ab460b,November 13,2021

[2] https://towardsdatascience.com/demystifying-convolutional-neural-networks-using-class-activation-maps-fe94eda4cef1, September 26, 2019

[3] https://towardsdatascience.com/a-concise-history-of-neural-networks-2070655d3fec, August 13, 2016

[4] https://indiantechwarrior.com/fully-connected-layers-in-convolutional-neural-networks/: :text=Fully %20connected%20layers%20are%20an,features%2C%20and%20analyzing%20them%20independently.

[5] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, December 15, 2018

[6] https://medium.com/appyhigh-technology-blog/convolutional-neural-networks-a-brief-history-of-their-evolution-ee3405 August 31, 2021

[7] https://levelup.gitconnected.com/how-do-computers-communicate-with-each-other-50636acbeb4c, February 22, 2021

[8] https://towardsdatascience.com/reducing-the-carbon-foot-prints-of-cnns-at-the-cost-of-interactions-depthwise-pointwise-conv-5df850ea33a4, March 20, 2020