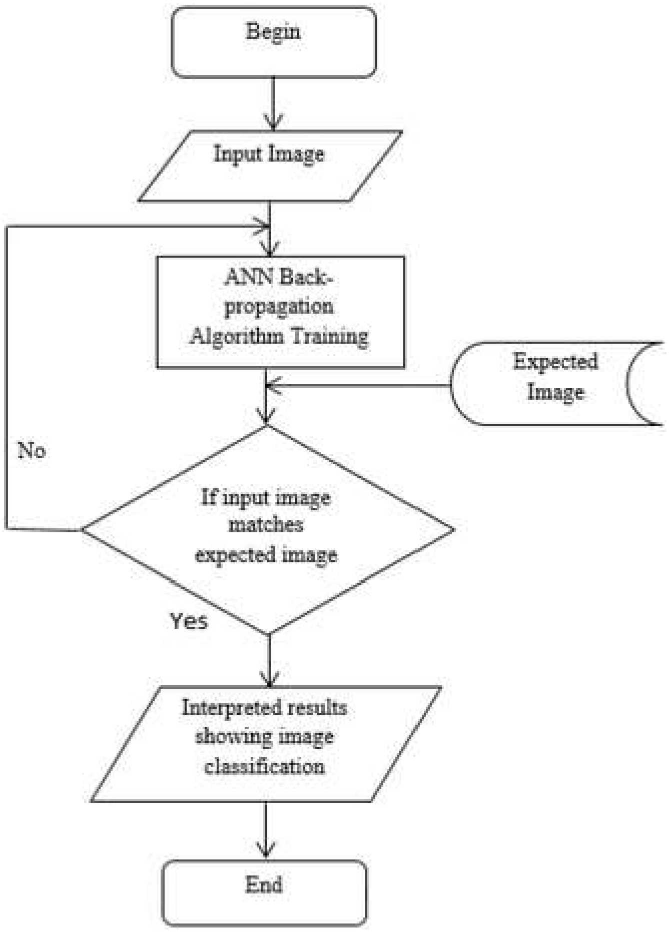
# Luanne Seaman CPE4903 Final Project 5/6/2022

## Introduction

Neural networks refer to computing systems that are modeled after biological neural networks. They consist of an input layer, one or more hidden layers, and an output layer. We give the neural network the data we want it to analyze in the input layer. Inputs can come in many forms like text, speech, and images. Different neural networks are better at working with different types of data and/or forms of pattern recognition. An artificial neural network works by receiving inputs, applying initialized weights (usually set to 0), finding the error between the expected and actual weights, and using that error as feedback to update the weights. The updating of weights is done through the backpropagation algorithm. Convolutional neural networks (CNNs) are a subclass of neural networks that are mainly used in image processing. CNNs can identify characteristics independently. Thus, they can be trained to identify and classify images without an external user supplying information about, for example, what makes a cat a cat and a dog a dog. For this project, a digit recognition system was created using a Raspberry Pi 3B+, an ArduCam camera, and a SenseHat.

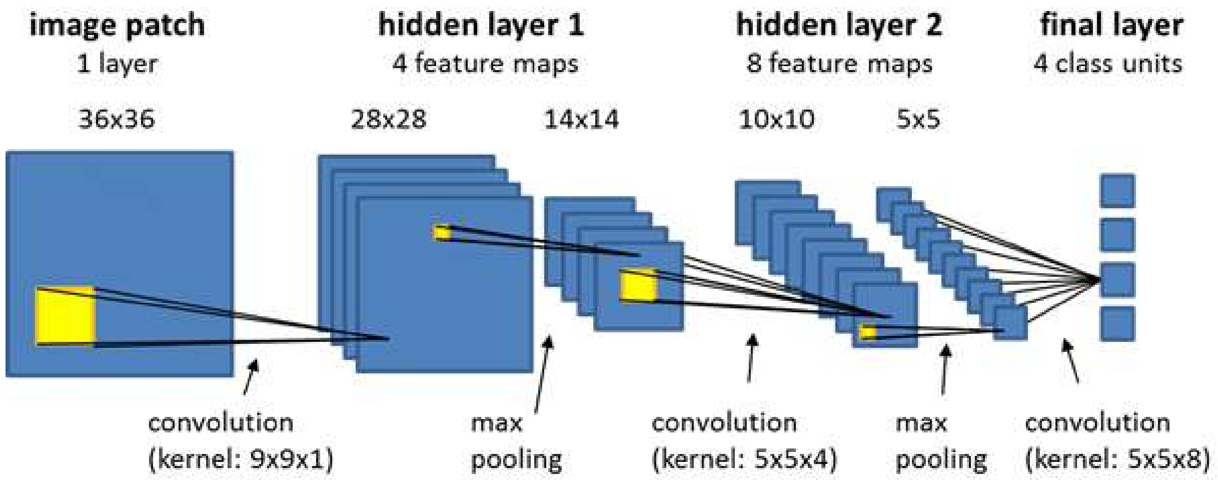

Begin
Input Image
ANN Back-propagation Algorithm Training
Expected Image
No
If input image matches expected image
Yes
Interpreted results showing image classification
End

Flow of Backpropagation algorithm from: https://link.springer.com/article/10.1007/s10586-018-2369-7/figures/3 (https://link.springer.com/article/10.1007/s10586-018-2369-7/figures/3)

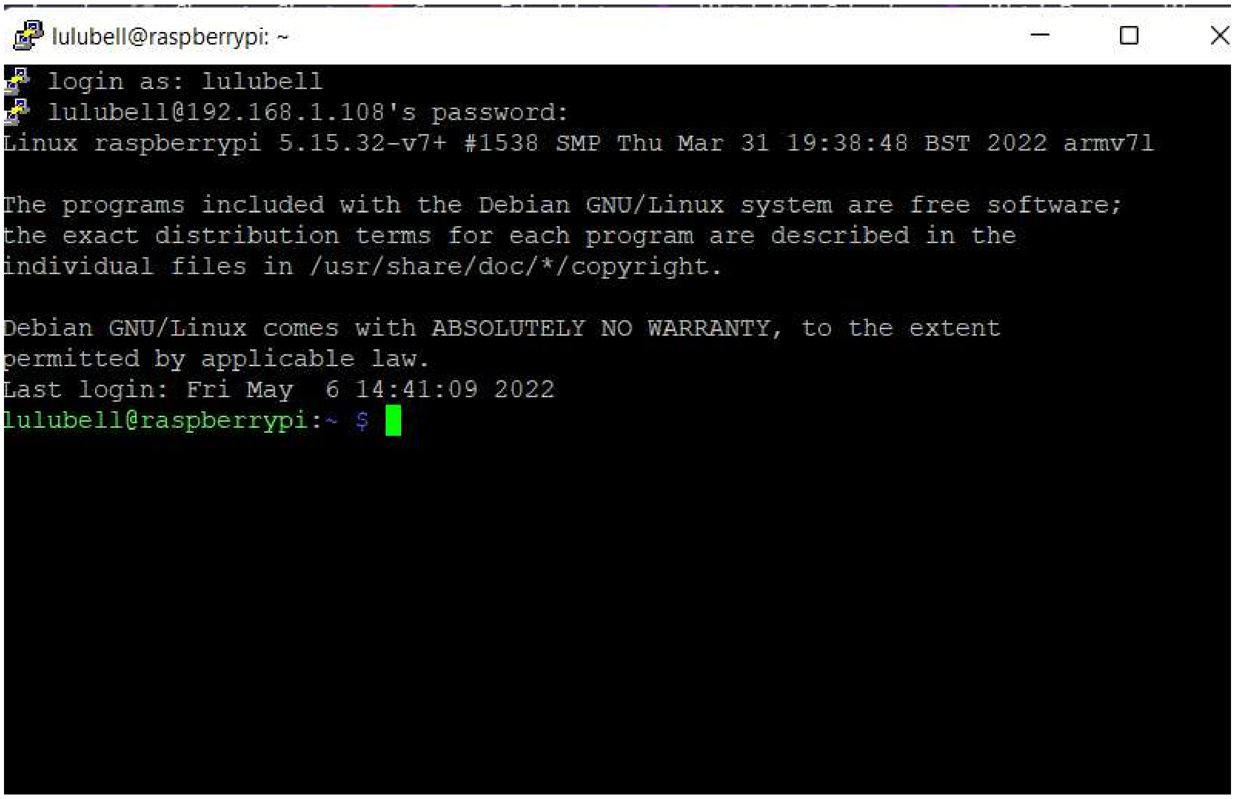
## Theory

CNNs consist of an input layer, convolutional layers, pooling layers, fully connected layers, and an output layer. Convolutional layers scan the image for patterns and return a high, positive value when a pattern is detected and a zero or negative number when no pattern is detected. After the convolutional layer, the layer's output is passed through an activation function like sigmoid or ReLu. Next, a pooling layer scales down the convolutional layer’s output by passing through a x by x matrix (size defined by the programmer) and selecting the largest value (for max pooling) or finding the average of values (average pooling).

Next, there are the fully connected layers. A fully connected layer is basically a standard neural network. Prior to entering this layer, the outputs from the features analysis layers are flattened into a column vector. The fully connected layer ascribes weights to each output. Each output is multiplied by its respective weight and added to a bias vector, b. The calculations in the fully connected layer can be defined as g(Wx+b) where g is the activation function, W is the weight matrix with dimensions (# neurons in previous layer, # neurons in current layer), X is the input matrix with dimensions (# neurons in previous layer, 1), and b is the bias vector with dimensions (# neurons in previous layer, 1). At the final layer, the output is passed through a softmax activation function. The final output is a vector containing probabilities of the image belonging to each class.

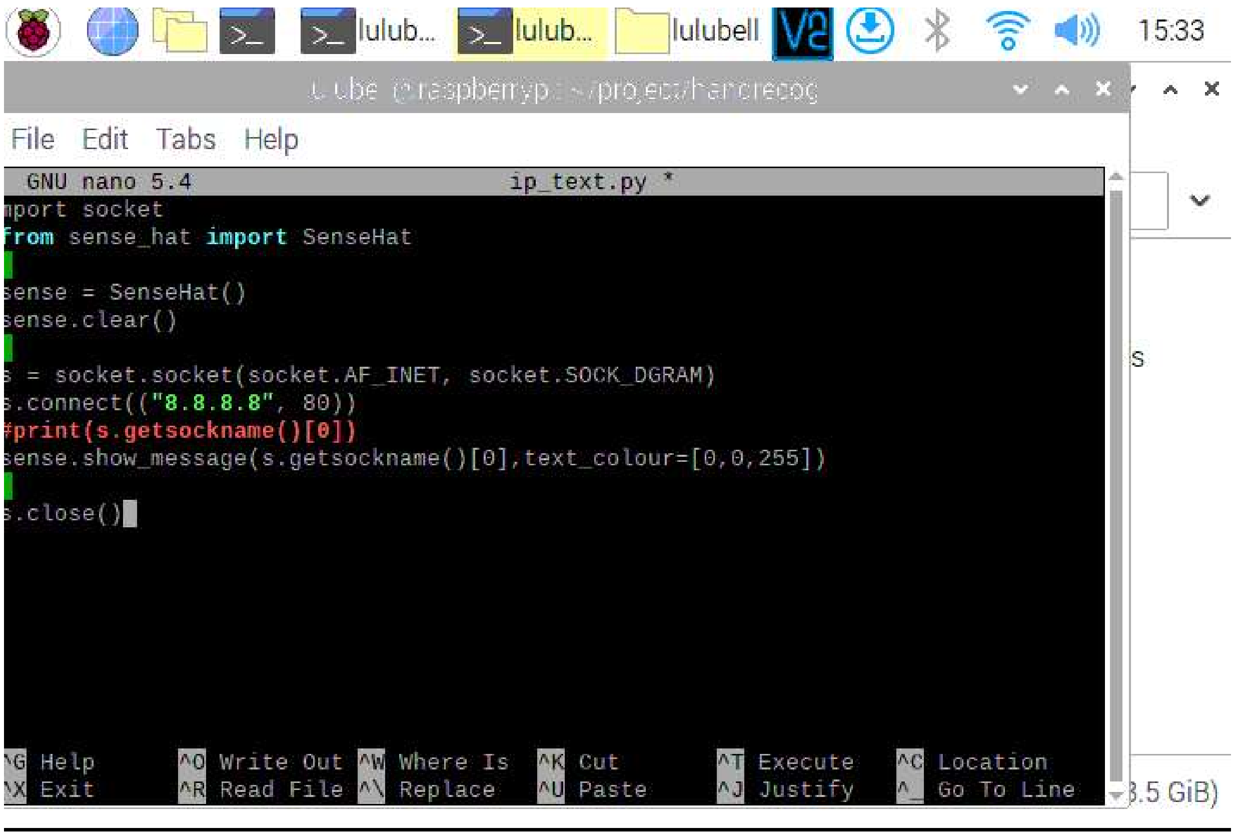


CNN Diagram from:
https://docs.ecognition.com/eCognition_documentation/User%20Guide%20Developer/8%20Classific
%20Deep%20Learning.htm
(https://docs.ecognition.com/eCognition_documentation/User%20Guide%20Developer/8%20Classifi
%20Deep%20Learning.htm)



```
login as: lulubell
lulubell@192.168.1.108's password:
Linux raspberrypi 5.15.32-v7+ #1538 SMP Thu Mar 31 19:38:48 BST 2022 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri May  6 14:41:09 2022
lulubell@raspberrypi:~ $
```

SSH connection to Raspberry Pi



```
 GNU nano 5.4                            ip_text.py *
mport socket
from sense_hat import SenseHat

sense = SenseHat()
sense.clear()

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.connect(("8.8.8.8", 80))
#print(s.getsockname()[0])
sense.show_message(s.getsockname()[0],text_colour=[0,0,255])

s.close()

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line
```

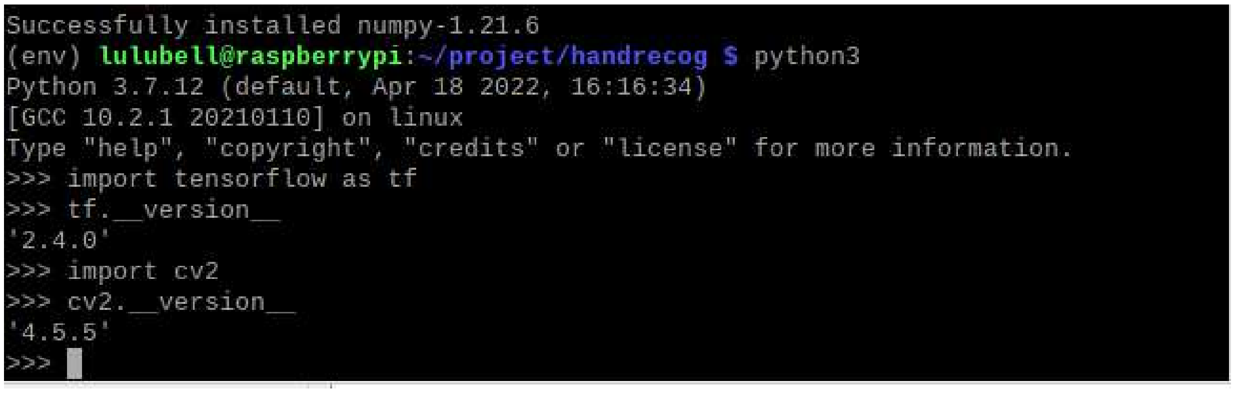
Script to scroll IP address on SenseHat

## Procedure

The first steps were downloading Raspberry Pi Imager and downloaded the 32-bit Raspberry Pi OS to a micro SD card. Then, I connected a mouse, keyboard, and monitor to the Raspberry Pi and connected it to power. Following that, I entered “ifconfig” into the Raspberry Pi’s terminal to find its IP address. Finally, I entered the “sudo raspi-config” and enabled Camera, SSH, I2C, and VNC by going to “Preferences” then “Raspberry Pi Configuration” followed by “Interfaces.”

Using another computer, I opened PuTTY, entered the Pi’s IP address, and established an SSH connection. I logged in using the default username and password. At this point, I was able to connect remotely to the Raspberry Pi. Following that, I ran the “sudo apt-get update” and “sudo apt-get install sense-hat” command to make sure all the Pi’s software is up-to-date and install the SenseHat package, respectively. Finally, I ran the following script (shown in the cell below) to display the IP address on the SenseHat.

The next big step was installing OpenCV and TensorFlow on the Pi. OpenCV is a library of software that is mainly focused on computer vision. TensorFlow is a widely used open-source software library used for machine learning. Because of software incompatibility, Python had to first be downgraded to 3.7.12. Next, I installed a virtual environment to work within for this project. Then, I installed OpenCV and TensorFlow in the virtual environment along with the packages for the SenseHat, camera, and dependencies.



```
Successfully installed numpy-1.21.6
(env) lulubell@raspberrypi:~/project/handrecog $ python3
Python 3.7.12 (default, Apr 18 2022, 16:16:34)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> tf.__version__
'2.4.0'
>>> import cv2
>>> cv2.__version__
'4.5.5'
>>> 
```

TensorFlow and OpenCV installation

Now, it was time to train the Pi. I used the following algorithm shown below. The training model had an accuracy of 98.0% percent. The training model was saved as "mnist_trained_model.h5." Then, I ran another algorithm that takes pictures with the ArduCam, processes them, and feeds them into the training model for classification. It performs the following steps: -Converting the image to grayscale -Converting to uint8 range -Determining threshold using Otsu's method (determining difference between foreground and background pixels) -Resizing image -Inverting image to black background -Feeding image into trained neural network -Printing answer and accuracy on SenseHat

In [6]:
```
# Training Model Algorithm


## import required packages

from __future__ import print_function
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Dropout, Input, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.utils import to_categorical


batch_size = 64
num_classes = 10
epochs = 12


## Determine image dimensions

img_rows, img_cols = 28, 28

## Divide MNIST data into train and test sets


(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

## Resize and process data
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
print("Training matrix shape", X_train.shape)
print("Testing matrix shape", X_test.shape)

Y_train = keras.utils.to_categorical(Y_train, num_classes)
Y_test = keras.utils.to_categorical(Y_test, num_classes)

## Create and define model
model = Sequential()
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(10))
model.add(Activation('softmax'))
```

```
## Compile model and print summary
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics = ['accu

model.summary()

## Fit and evaluate model
history = model.fit(X_train, Y_train, batch_size=128, epochs=4)
score = model.evaluate(X_test, Y_test,verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
Yhat = (model.predict(X_test) > 0.5).astype("float32")
Yhat = Yhat.reshape(-1,1)  # column vector
Y_test = Y_test.reshape(-1,1)  # column vector

print(X_test.shape)

## Resize X and Y for CNN
X_train2 = X_train.reshape(X_train.shape[0], 28, 28, 1)   #samples, w, h, channel
X_test2 = X_test.reshape(X_test.shape[0], 28, 28, 1)
Y_train = Y_train.reshape(-1,1)
Y_test = Y_test.reshape(-1,1)

print('Shape of X_train2 is {}'.format(X_train2.shape))
print('Shape of X_test2 is {}'.format(X_test2.shape))
print('Shape of Y_train is {}'.format(Y_train.shape))
print('Shape of Y_test is {}'.format(Y_test.shape))


## Convert Y to binary matrices associated with classes 0 to 9


nb_classes = 10
Y_train = to_categorical(Y_train, nb_classes)
Y_test = to_categorical(Y_test, nb_classes)
print('Shape of Y_train is {}'.format(Y_train.shape))
print('Shape of Y_test is {}'.format(Y_test.shape))


## Define CNN Model


model_cnn = Sequential()
model_cnn.add(Conv2D(32, (2, 2), input_shape = (28, 28, 1), activation = 'relu'))
model_cnn.add(MaxPooling2D(pool_size = (2,2)))
model_cnn.add(Dropout(0.2))
model_cnn.add(Flatten())
model_cnn.add(Dense(units = 200, activation = 'relu'))
model_cnn.add(Dense(units = 10, activation = 'softmax'))


## Compile and Evaluate Model


model_cnn.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics

print(model_cnn.summary())
```

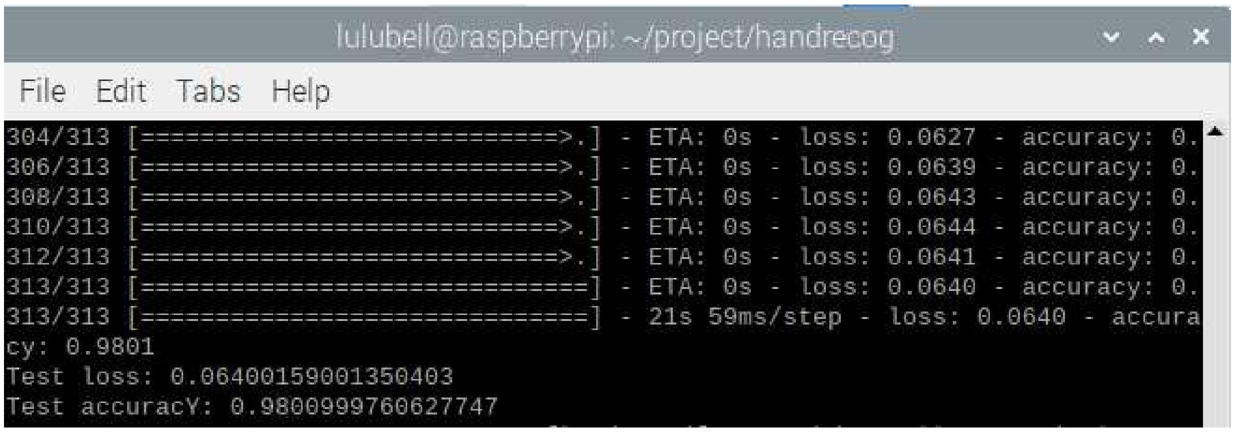
```
history = model_cnn.fit(X_train2,Y_train,epochs=5,batch_size=200,validation_split

score = model_cnn.evaluate(X_test2, Y_test, verbose=1)


## Save model

model.save('mnist_trained_model.h5')
```

```
(10000, 784)
Shape of X_train2 is (60000, 28, 28, 1)
Shape of X_test2 is (10000, 28, 28, 1)
Shape of Y_train is (600000, 1)
Shape of Y_test is (100000, 1)

Shape of Y_train is (600000, 10)
Shape of Y_test is (100000, 10)
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 27, 27, 32)        160

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 dropout_2 (Dropout)         (None, 13, 13, 32)        0

 flatten (Flatten)           (None, 5408)              0
```

In [8]:
```
## Pi Camera Algorithm


from skimage.io import imread
from skimage.transform import resize
from sense_hat import SenseHat
import numpy as np
from skimage import data, io
from matplotlib import pyplot as plt
from skimage import img_as_ubyte
from skimage.color import rgb2gray
import cv2
import datetime
import argparse
import imutils
import time
import tensorflow as tf
from tensorflow import keras
from time import sleep
from imutils.video import VideoStream
from tensorflow.keras.models import load_model

## Import training model

model=load_model('mnist_trained_model.h5')
sense = SenseHat()
sense.show_message('Ready')
blue = (0, 0, 255)


#Specify ArduCam as input
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--picamera", type=int, default=0,
        help="Choose whether to use ArduCam")
args = vars(ap.parse_args())

#Initialize camera stream
vs = VideoStream(usePiCamera=args["picamera"] > -1).start()
time.sleep(2.0)

#Process captured frames

def ImagePreProcess(im_orig):
        cv2.imwrite("num.jpg", im_orig)
        im_orig = cv2.imread("num.jpg")
        im_gray = rgb2gray(im_orig)
        img_gray_u8 = img_as_ubyte(im_gray)
        (thresh, im_bw) = cv2.threshold(img_gray_u8, 128, 255, cv2.THRESH_BINARY
        img_resized = cv2.resize(im_bw,(28,28))
        im_gray_invert = 255 - img_resized
        im_final = im_gray_invert.reshape(1,28,28,1)
        predict = model.predict(im_final)
        if predict > 0.5:
            ans = model.predict(im_final)
            ans = ans[0].tolist().index(max(ans[0].tolist()))
            sense.show_message(str(ans), str(round(max(ans[0].tolist(),2))), text
#Loop through captured frames

def main():

        while True:
                try:
                        # Resize frame
                        frame = vs.read()
                        frame = imutils.resize(frame, width=400)
                        ImagePreProcess(frame)

                        key = cv2.waitKey(1) & 0xFF

                        #Exit by pressing 0.  Initialize preprocessing by pressir
                        if key == ord("0"):
                                break
                                # do a bit of cleanup
                                cv2.destroyAllWindows()
                                vs.stop()

                        else:
                                pass

                except KeyboardInterrupt:

                        cv2.destroyAllWindows()
                        vs.stop()



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

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Input In [8], in <cell line: 15>()
     13 import datetime
     14 import argparse
---> 15 import imutils
     16 import time
     17 import tensorflow as tf

ModuleNotFoundError: No module named 'imutils'
```

## Data and Analysis



```
304/313 [============================>.] - ETA: 0s - loss: 0.0627 - accuracy: 0.
306/313 [============================>.] - ETA: 0s - loss: 0.0639 - accuracy: 0.
308/313 [============================>.] - ETA: 0s - loss: 0.0643 - accuracy: 0.
310/313 [============================>.] - ETA: 0s - loss: 0.0644 - accuracy: 0.
312/313 [============================>.] - ETA: 0s - loss: 0.0641 - accuracy: 0.
313/313 [==============================] - ETA: 0s - loss: 0.0640 - accuracy: 0.
313/313 [==============================] - 21s 59ms/step - loss: 0.0640 - accura
cy: 0.9801
Test loss: 0.06400159001350403
Test accuracY: 0.9800999760627747
```

Training Results

This project involved a good amount of troubleshooting. I had to routinely change the Pi Camera argument integer between 1 and 0. The system would present an error saying it was unable to find a camera at the previous index. Moreover, I had to specify my keras imports by adding "tensorflow.keras" to each instance. Additionally, I dealt with an "AttributeError: 'str' object has no attribute 'decode'" error. The most common solution was to downgrade h5py. However, this caused a cascade of dependency issues. To solve this, I had to navigate to "~/project/env/lib/python3.7/site-packages/tensorflow/python/keras" and remove each instance of ".decode('utf-8')." Lastly, I had issues connecting with the Sense Hat. This was resolved by reinstalling the Sense Hat package and running the following commands to re-download the EEPROM flash tool: -git clone https://github.com/raspberrypi/hats.git (https://github.com/raspberrypi/hats.git) -cd hats/eepromutils -make

## Conclusion

Overall, this reinforced how many running parts there are in neural network applications. There were times I had to make adjustments to the algorithms when they had been working correctly in the last test. Creating an application that can run continuously is an arduous effort that requires a great deal of testing and quality analysis as well as consistent updating. Still, it was satisfying to see the application come together and work properly.