Deployment of ML for Gesture Recognition

Summer Research Internship

Prof. Tapas Kumar Maiti

Chirag Vaghasiya
DAIICT

Gujarat, India

201901003

Tushar Mangukiya
DAIICT

Gujarat, India

201901063

Aditya Ramanuj
DAIICT

Gujarat, India

201901138

Siddharth Jadav

DAIICT

Gujarat, India

201901215

I. Introduction

Machine learning and computer vision researchers are working on hand gesture recognition for human-computer interaction. Gesture recognition is a technology that reads and interprets motions or gestures as commands using sensors. The device with the capability of gesture recognition can help in multiple real-life applications. It has a broader range of applications, including in the healthcare, educational, and manufacturing sectors, among others. This paper involves a comparative study on Gesture recognition with the help of TinyML. TinyML is the same as ML, but it consumes less energy and costs. It can be used in implementing low energy systems.

II. Ease of use

As already discussed, a gesture-based interface makes interactions feel more natural. We are able to use technology quickly and with little effort due to the simplicity of simple hand motions. Our product has a great degree of flexibility as it can be carried easily anywhere, anytime.

It has optimal size and, therefore, can be carried by any individual. One such application is in the case of an old age person. When an aged person suffers a heart attack, he/she will fall suddenly to the ground, and such a downfall will create an extreme peak in the attached device. This unexpected peak will send the alert message to their guardian, and it can prevent further problems.

III. Objective

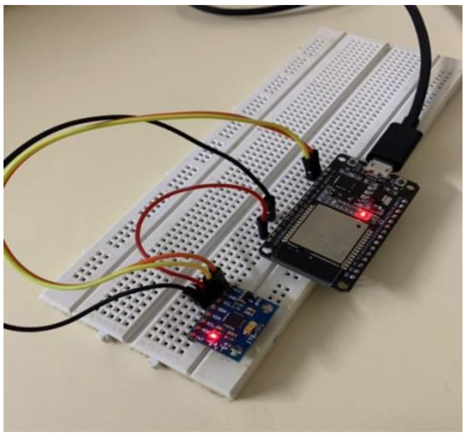
The main objective of gesture recognition research is to develop a system that can recognise particular human gestures and use them to communicate information or operate the device. As we know, it will detect and notifies an unexpected and catastrophic event of movement of the system.

IV. COMPONENTS

- ESP32 Wi-Fi model
- Accelerometer MPU6050
- Breadboard
- USB cable
- Jumper wires

V. EXPERIMENT

First, we started with the generation of the sine wave to learn how the Machine learning model works. We learned about a simple neural network to predict values of sine function x and its corresponding sin(x).



Setup of Arduino connection

The python code is verified in Arduino IDE. Then we uploaded that code to our microcomputer device. So now, the device is ready to collect the movement data. The device is connected to the laptop.

Now, live data collection is possible. So, as we move the device, the graph will be generated in the Arduino IDE. Accordingly, the code will produce the hex file of the movement.

Using Arduino IDE, we explored the code and its libraries. Now, code has enabled our tool to pass the real-time data of MPU6050 to our computer. The data we need to upload to the serial port using an Arduino IDE has a sampling frequency and be separated by "," in this example since we are providing acceleration data.

Here, we collected data on various movements such as leftward, rightward, up and downside. We use python to save and store the collected real-time data of the accelerometer. It will then generate a text file (.txt) of the collected data.

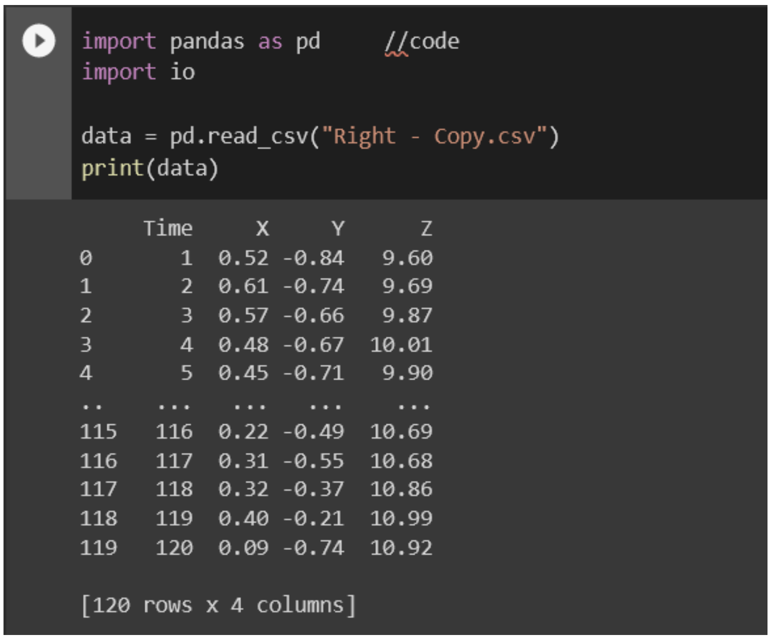
The next step is to convert .txt file into .csv file. In that conversion, we have to add column of time. It is the necessary step in plotting the graph (such as time -> x). Similarly, the process is repeated for all kinds of movements. Now, our data is ready for training.

The training procedure of data is done in Google collab. The following steps should be followed to train our data. Here steps for training the Right.csv file are given below.

A. *Training the Data*

a) Sample Data

Here, the Right - Copy.csv file contains the sample data from the accelerometer in x,y and z coordinates with respect to time. There are total 120 samples which are generated by moving the device in the right direction.



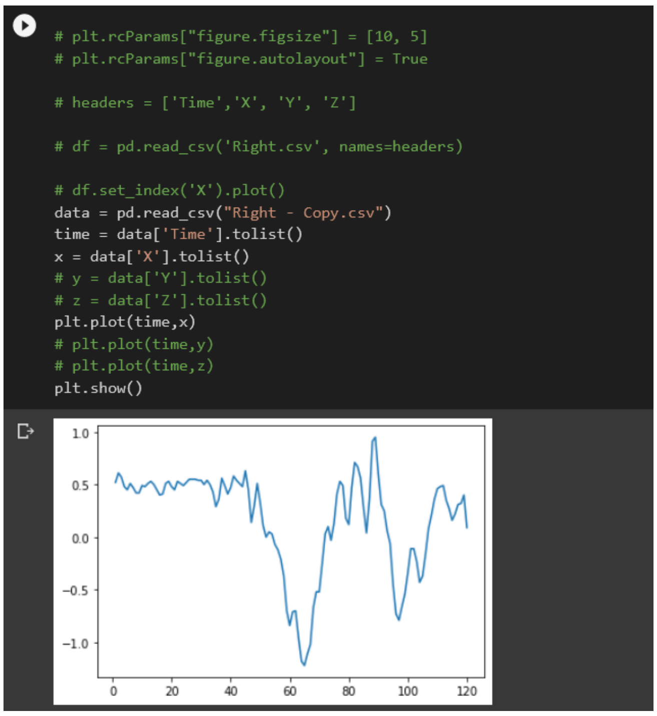
```
import pandas as pd      //code
import io

data = pd.read_csv("Right - Copy.csv")
print(data)
```

```
     Time     X     Y      Z
0       1  0.52 -0.84   9.60
1       2  0.61 -0.74   9.69
2       3  0.57 -0.66   9.87
3       4  0.48 -0.67  10.01
4       5  0.45 -0.71   9.90
..    ...   ...   ...    ...
115   116  0.22 -0.49  10.69
116   117  0.31 -0.55  10.68
117   118  0.32 -0.37  10.86
118   119  0.40 -0.21  10.99
119   120  0.09 -0.74  10.92

[120 rows x 4 columns]
```

b) We have created a list of all x,y and z coordinates. Then, the x against time plot is obtained.



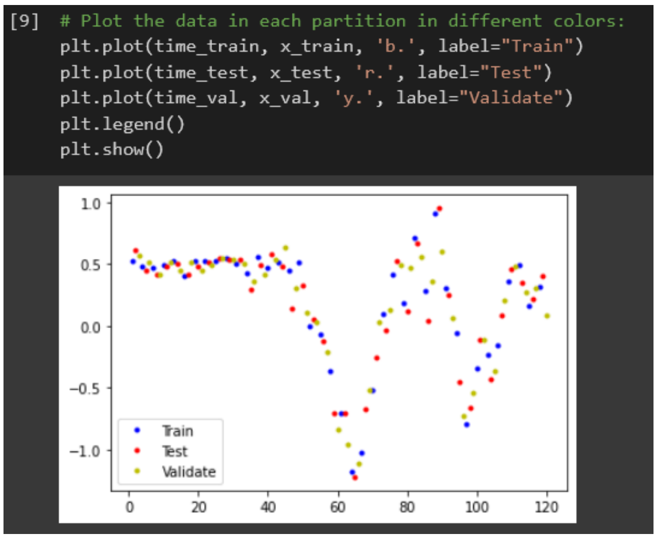
```
# plt.rcParams["figure.figsize"] = [10, 5]
# plt.rcParams["figure.autolayout"] = True

# headers = ['Time','X', 'Y', 'Z']

# df = pd.read_csv('Right.csv', names=headers)

# df.set_index('X').plot()
data = pd.read_csv("Right - Copy.csv")
time = data['Time'].tolist()
x = data['X'].tolist()
# y = data['Y'].tolist()
# z = data['Z'].tolist()
plt.plot(time,x)
# plt.plot(time,y)
# plt.plot(time,z)
plt.show()
```

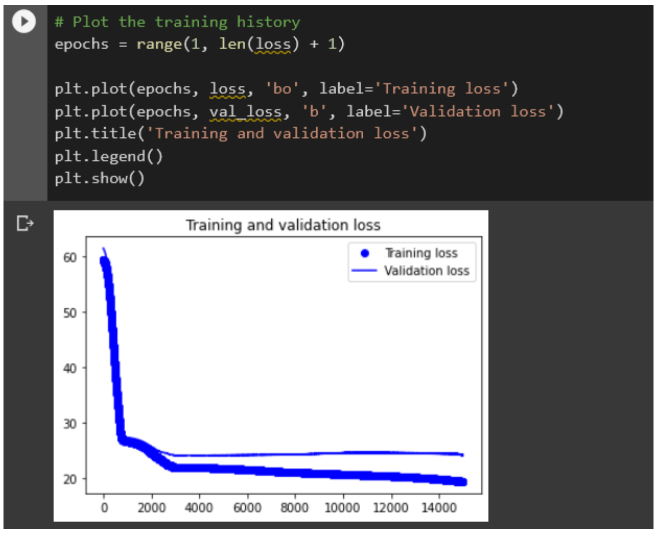
c) To train our data, we split the x dataset into three categories Train, Test and Validate. The data is plotted in each partition in different colours.



```
[9]  # Plot the data in each partition in different colors:
     plt.plot(time_train, x_train, 'b.', label="Train")
     plt.plot(time_test, x_test, 'r.', label="Test")
     plt.plot(time_val, x_val, 'y.', label="Validate")
     plt.legend()
     plt.show()
```

d) Training and Validation loss

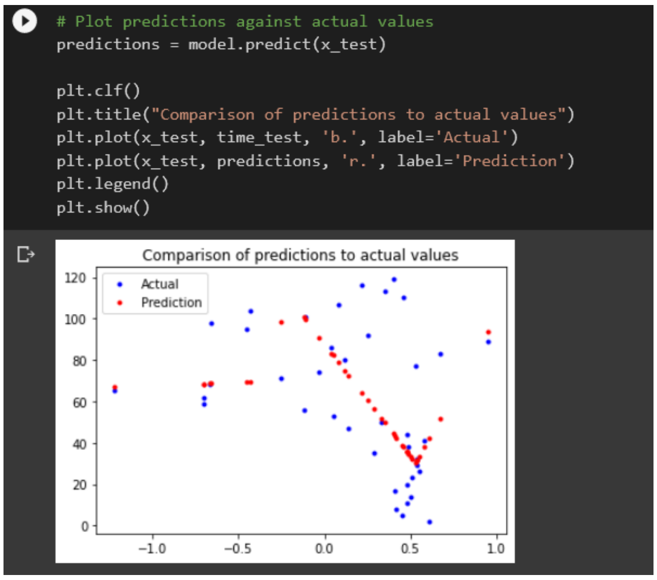
- Now, 15000 epochs are created. The larger the number of epochs, the more accurate result we get.
- The Training loss and Validation loss graph overlap as long as we consider a large number of samples. Here we have taken only 120 samples. Therefore there is some mismatch between training and validation loss.



```
# Plot the training history
epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

B. *Comparison between the predicted values and the original values*



```
# Plot predictions against actual values
predictions = model.predict(x_test)

plt.clf()
plt.title("Comparison of predictions to actual values")
plt.plot(x_test, time_test, 'b.', label='Actual')
plt.plot(x_test, predictions, 'r.', label='Prediction')
plt.legend()
plt.show()
```

After completing the training procedure, we have to convert it into a .h (hex) file, and our code will facilitate this need. This .h file will support in Arduino IDE and can detect the gesture of the device.

CONCLUSION

In this project, we learn to implement Machine Learning with the help of python. Usually, this kind of project is divided into two parts. One, which involves implementing Machine Learning with Python. As Python is a complex language and composed of various libraries, multiple threads and processes are applied to run the python code. Therefore, it will result in high throughput and need a high power source to run for sustainable time.

So, here as per our need, we opted for tinyML, which can be implemented efficiently in low energy systems like the ESP32 micro-computer, which is of 32 KB. The ESP32 micro-computer will fetch the code effectively with high speed. This will use low power consumption and give results with low throughput.

The code and related material are given in the below link:
https://drive.google.com/drive/folders/1s9Ge4BkghYPTpoWq34wp0oTxYp81LljQ?usp=sharing

REFERENCES

[1] https://youtu.be/BzzqYNYOcWc
[2] https://youtu.be/dU01M61RW8s
[3] https://www.hackster.io/Yukio/gesture-classification-with-esp32-and-tinyml-dab252
[4] https://studio.edgeimpulse.com/public/36985/latest