

# **HUMAN ACTIVITY PREDICTION**

Machine Learning Project

2020.05.05

## Table of Contents

1	Project Introduction .....	1
1.1	Accelerometer .....	1
1.2	Human Activity Prediction.....	2
2	Feature Extraction .....	2
3	Long Short Term Memory (LSTM).....	3
3.1	Introduction .....	3
3.2	Technical Background.....	4
4	LSTM over RNN .....	7
5	Code Breakdown.....	8
5.1	PyCharm (ML) Code.....	8
5.1.1	Raw Data Inspection .....	8
5.1.2	Data Import .....	9
5.1.3	Data Analyses .....	10
5.1.4	Preprocessing the Data.....	14
5.1.5	Splitting Data .....	15
5.1.6	Building the Model .....	15
5.1.7	Training the Model .....	17
5.1.8	Trained Model Evaluation .....	19
5.1.9	Freezing the Trained Model.....	21
5.2	Android Code .....	22
5.2.1	Main Activity.....	22
5.2.2	Model Import and Prediction .....	25
5.2.3	User Interface.....	25
5.2.4	Application in Action.....	26
6	Used Technologies.....	29
7	Possible Areas for Future Improvements.....	29
8	References.....	29

## Table of Figures

Figure 1.1: Accelerometer .....	1
Figure 1.2: x,y,z Axis.....	1
Figure 1.3 : Human Activity .....	2
Figure 3.1 : RNN Structure.....	3
Figure 3.2 : LSTM Structure.....	4
Figure 3.3 : Cell State .....	4
Figure 3.4 : Sigmoid Layer & Pointwise Operation .....	5
Figure 3.5 : Forget Gate Layer.....	5
Figure 3.6 : Creating New Cell State .....	6
Figure 3.7 : Updating the Cell State.....	6
Figure 3.8 : Finalizing the Output.....	7
Figure 5.1 : Input data.....	8
Figure 5.2 : Import Data.....	9
Figure 5.3 : Analyzing Data.....	10
Figure 5.4 : Check the Imported Data.....	10
Figure 5.5 : Training Examples by User Bar Graph .....	11
Figure 5.6 : Sitting Axis Chart.....	11
Figure 5.7 : Standing Axis Chart .....	12
Figure 5.8 : Walking Axis Chart.....	12
Figure 5.9 : Jogging Axis Chart.....	13
Figure 5.10 : Downstairs Axis Chart .....	13
Figure 5.11 : Upstairs Axis Chart .....	14
Figure 5.12 : Reshaping Data.....	14
Figure 5.13 : Output of Reshaping Data .....	15
Figure 5.14 : Splitting Data.....	15
Figure 5.15 : Printing Split Data .....	15
Figure 5.16 : Building the Model Part 1 .....	16
Figure 5.17 : Building the Model Part 2 .....	16
Figure 5.18 : Building the Model Part 3 .....	17
Figure 5.19 : Training the Model Part 1 .....	17
Figure 5.20: Training the Model Part 2 .....	18
Figure 5.21 : Final Accuracy and Loss .....	18
Figure 5.22 : Use of Pickle .....	19
Figure 5.23 : Plotting Accuracy and Loss.....	19
Figure 5.24 : Accuracy and Loss Chart .....	20
Figure 5.25 : Building the Confusion Matrix.....	20
Figure 5.26 : Confusion Matrix .....	21
Figure 5.27 : Freezing the Trained Model .....	21
Figure 5.28 : Main Android Part 1 .....	22
Figure 5.29 : Main Android Part 2.....	22

Figure 5.30 : Main Android Part 3.....	23
Figure 5.31 : Main Android Part 4.....	23
Figure 5.32 : Main Android Part 5.....	24
Figure 5.33 : Main Android Part 6.....	24
Figure 5.34 : Model Import and Prediction .....	25
Figure 5.35 : User Interface .....	26
Figure 5.36 : Live Application Part 1 .....	27
Figure 5.37 : Live Application Part 2 .....	28

# 1 Project Introduction

## 1.1 Accelerometer

Accelerometer is type of an electromagnetic hardware module [Figure 1.1] which can measure acceleration of a moving object or a person. Smart phones in present market contains an accelerometer module to satisfy various requirements. Mobile phones have applications such as compass, sensor-based car games and auto rotate (feature). Mentioned applications and functions need the data of x, y and z axis [Figure 1.2] of the accelerometer (tri axial accelerometer) to determine the required behavior. Sensitivity of these modules are very high. Then comes the question what are the possible research areas that these features can be used. One heavily researched area is about predicting human activities using accelerometer data.

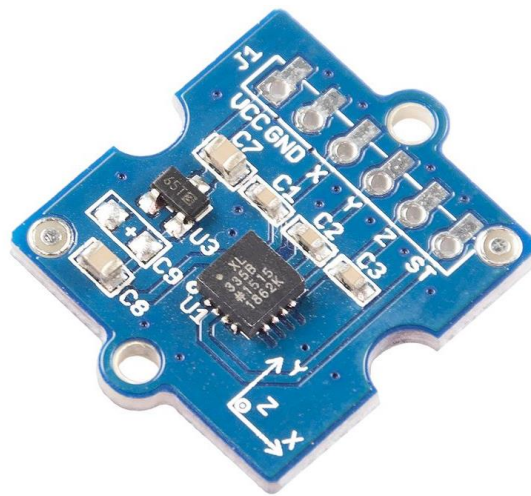
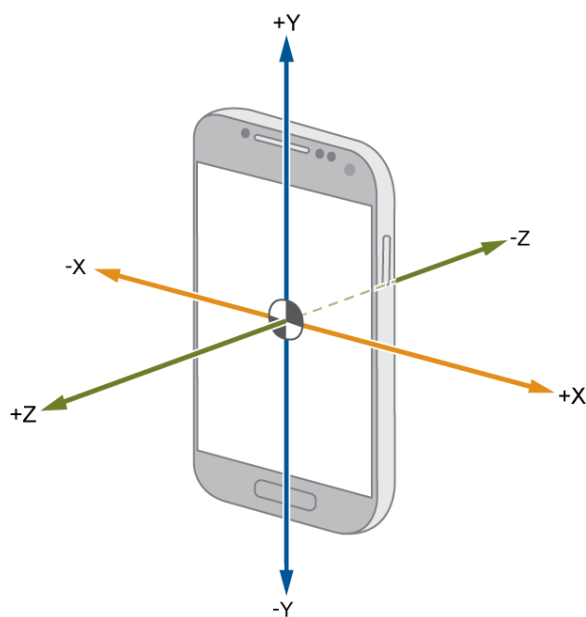


Figure 1.1: Accelerometer



z-axis - captures the forward movement of the leg  
y-axis - captures the upward and downward motion  
x-axis - captures horizontal movement of the user's leg

Figure 1.2: x,y,z Axis

## 1.2 Human Activity Prediction

What is human activity prediction? There are multiple answers to this question. It is a known fact that social media giants are tracking user activities by misusing accelerometer data. Though the techniques are same, the approach used here is to predict human activities for a particular time with the awareness of the user. Through this project there are 6 different activities that can be identified namely walking, jogging, sitting, standing, walking upstairs and walking downstairs.

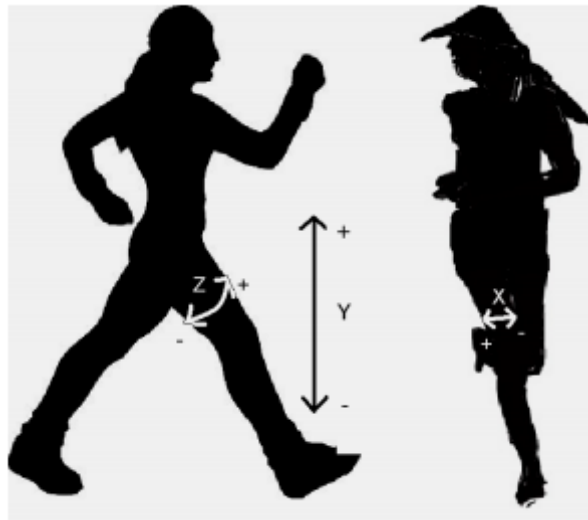


Figure 1.3 : Human Activity

## 2 Feature Extraction

The concept of this machine learning project and the dataset were taken from a previous research [1] that have been conducted to predict human activities. Research data was gathered from 36 people and multiple android devices including the Nexus One, HTC Hero, and Motorola Backflip was used to proceed with the data gathering process. At first, raw data should be transformed to apply classification algorithms. Below are the features based on the raw accelerometer readings as mentioned in the research [1].

- Average: Average acceleration (for each axis)
- Standard Deviation: Standard deviation (for each axis)
- Average Absolute Difference
- Average Resultant Acceleration: Average of the square roots of the sum of the values of each axis squared  $\sqrt{(x_i^2 + y_i^2 + z_i^2)}$  over the ED
- Time between Peaks
- Binned Distribution

The researches have prepared data in supervised learning approach by turning data into examples with a labeled activity. Models were built to predict user activities using three classification algorithms in WEKA data mining suite namely decision trees (J48), logistic regression and multilayer neural networks.

In this machine learning project, we'll be using the same dataset [2] and will achieve a higher accuracy using the deep learning technique known as long short term memory.

### 3 Long Short Term Memory (LSTM)

#### 3.1 Introduction

In deep learning there are three main types of neural networks.

- 1 Artificial Neural Networks (ANN)
- 2 Convolution Neural Networks (CNN)
- 3 Recurrent Neural Networks (RNN)

ANN – Mostly used and suitable for number based projects

CNN – Mostly used and suitable for image based projects

RNN – Mostly used and suitable for time series based projects

Since our x, y and z axis data are dependent features with time, it is clear that RNN is the possible candidate for our project. All RNN neural networks has a repeating module which enables to hold a memory for a short period of time. In RNN it'll be a single tanh layer as shown in the below image [Figure 3.1].

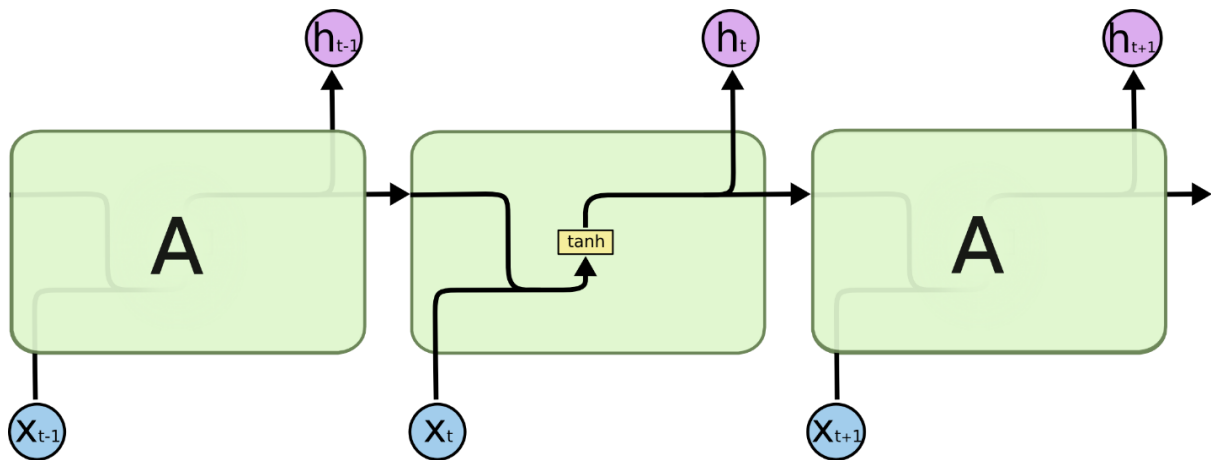


Figure 3.1 : RNN Structure

The problem with the typical RNN is the short period of memory holding time and the respective low accuracy that can be generated when the application needs more context of the past data. In other words RNN cannot handle the output, when the gap between the past memories increase. As an example, suppose the task is to build a machine learning program which predicts the next word in a sentence.

First Sentence – The sky is\_\_\_\_\_

Second Sentence - He was born in Japan. He is fluent in \_\_\_\_\_

We can see that the first sentence need less information of the past data to determine the next word as ‘Blue’. But in the second sentence in order to fill that the person is fluent in Japanese the algorithm should be able to remember the first sentence (more past data) as well. Hence, the prediction accuracy of the RNN will be decreased.

To address this question an enhanced version of RNN known as LSTM was introduced by Hochreiter & Schmidhuber in 1997. So, the LSTM is capable of learning long term dependencies. LSTM also have the repeating module structure like RNN, but it has four interacting neural network layers than a one, which enables it to hold more dependencies of the data [Figure 3.2].

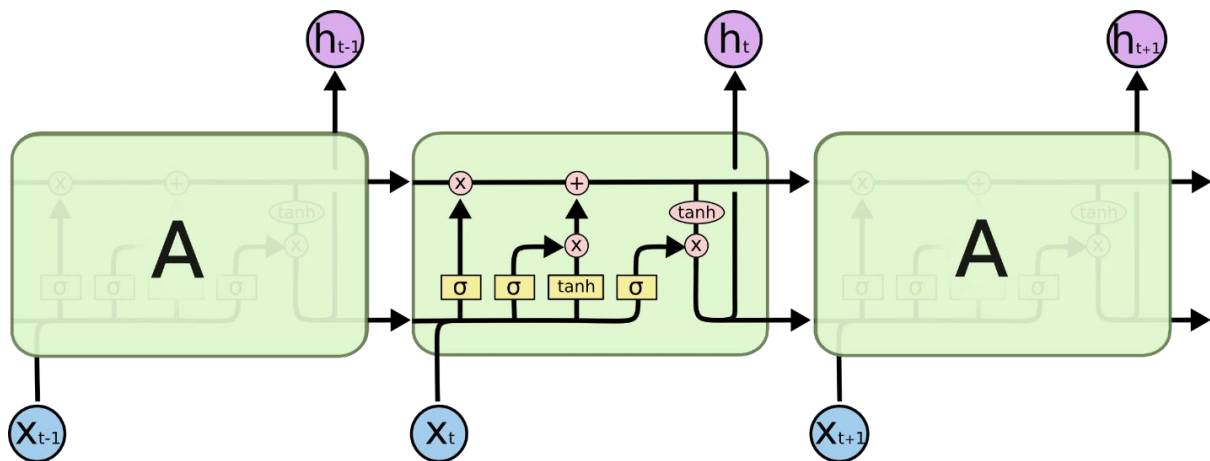


Figure 3.2 : LSTM Structure

### 3.2 Technical Background

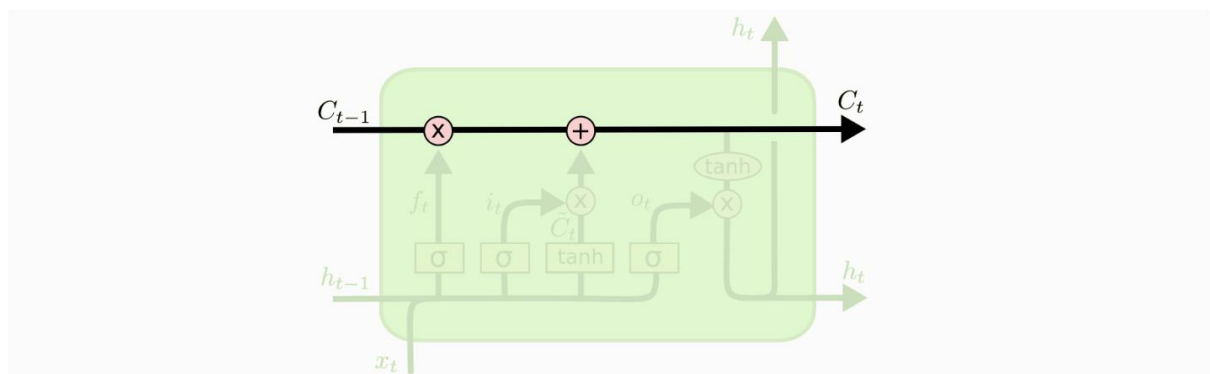


Figure 3.3 : Cell State

In LSTM all the changes are tracked with the feature called ‘Cell state’. Cell state is like a line [Figure 3.3] covering all the interactions which occur in the network. All the changes



will get updated to maintain an updated cell state. Also, the final output will be extracted from the cell state thus it's mandatory to handle the cell state carefully to achieve a higher level of accuracy in the output. So how does the LSTM network communicate about the changes with the cell state? Gates are the solution. Gates can be considered as a space to pass only the required and authorized information. Gates are using sigmoid neural net layer and a pointwise multiplication operation [Figure 3.4] to update the cell state.

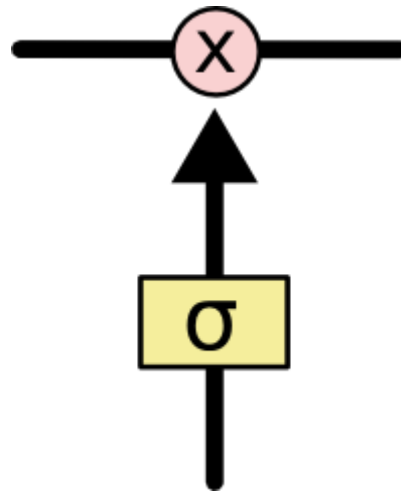
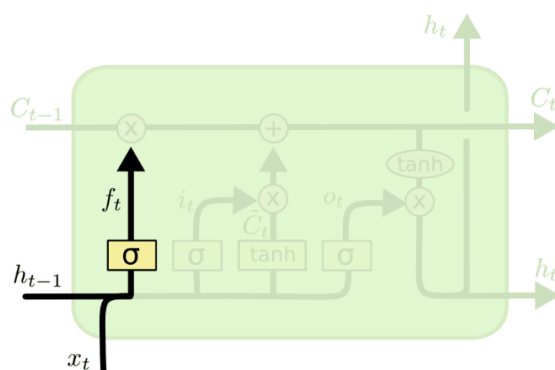


Figure 3.4 : Sigmoid Layer & Pointwise Operation

In a LSTM network there are three gates to protect the process from unacceptable situations.

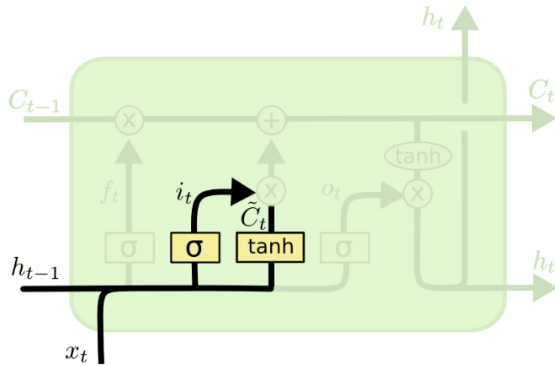


$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

Figure 3.5 : Forget Gate Layer

Since the connection between the cell state and network is explained, let's dive into the LSTM network. Its need to categorize the information that are eligible and not eligible to proceed to the cell state. Above image [Figure 3.5] shows the equation which determines the information for the neglecting process (values that are going to get thrown away). This is known as the forget gate layer (sigmoid layer). Output of the previous instance and the new

input is passed through the sigmoid layer along with the weight and bias to generate the forget gate layer.



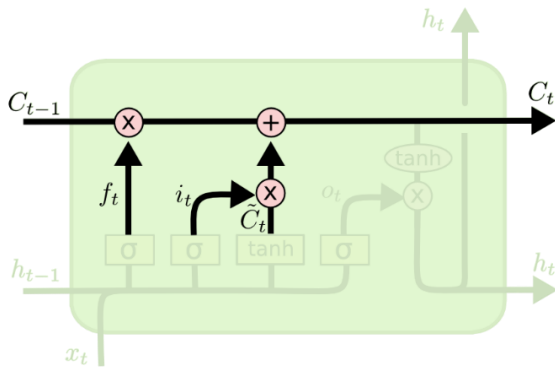
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Figure 3.6 : Creating New Cell State

So far the LSTM network has decided what to neglect and the next step is to look at the information that are going to the cell state. This process is executed by two equations which can be named as input gate layer and tanh layer equation. Input gate layer is responsible for deciding the data that can be updated to the cell state and the tanh layer will create a vector for these new values [Figure 3.6].

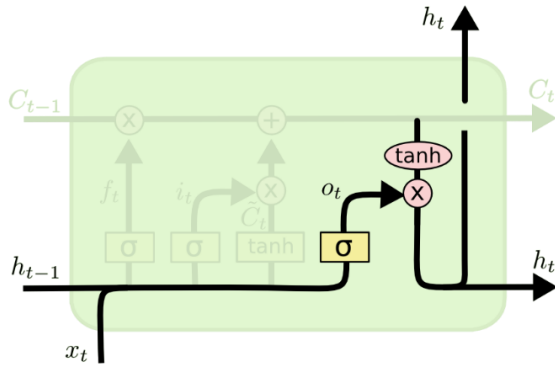
LSTM network is finished with categorizing the eligible and non-eligible information and it can proceed with updating the cell state [Figure 3.7].



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 3.7 : Updating the Cell State

New cell state ( $c(t)$ ) will be the addition of  $f(t) * c(t-1)$  and  $i(t) * \tilde{c}(t)$ .  $f(t)$  is the output of the forget gate layer section that we have discussed above [Figure 3.5].  $C(t-1)$  is the old cell state. Other section of the addition is nothing but the multiplication of the previous two equations [Figure 3.6].



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

Figure 3.8 : Finalizing the Output

To generate the final output, updated cell state will be passed through the  $\tanh$  layer while multiplying it with the output of the sigmoid layer. This happens because we are extracting only the required features from the cell state to produce the output. These last functions are in our codes as well. All the intermediate operations are hidden based on our implementation of LSTM with Tensorflow.

#### 4 LSTM over RNN

Inputs of the accelerometer gets changed rapidly with the time and sometimes the device may produce inaccurate data. The capability of learning long term dependencies of LSTM will helps to provide a more accurate output over RNN.

## 5 Code Breakdown

### 5.1 PyCharm (ML) Code

#### 5.1.1 Raw Data Inspection

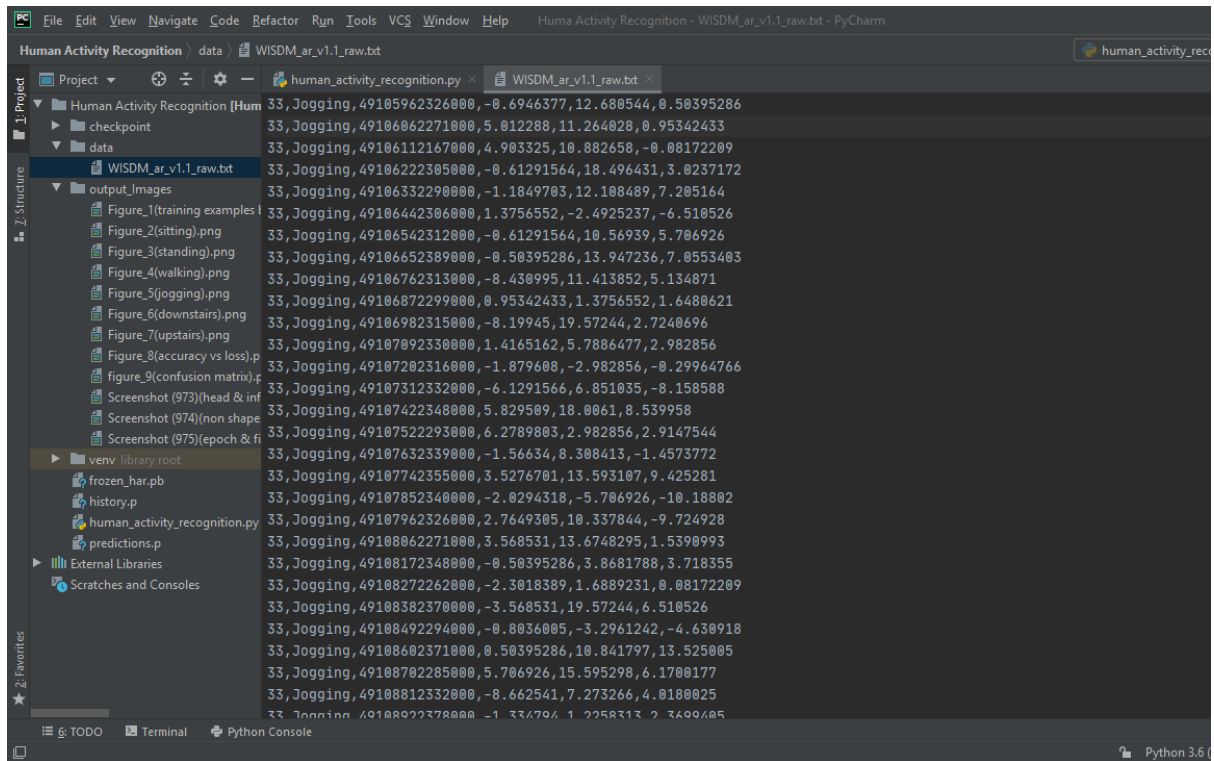
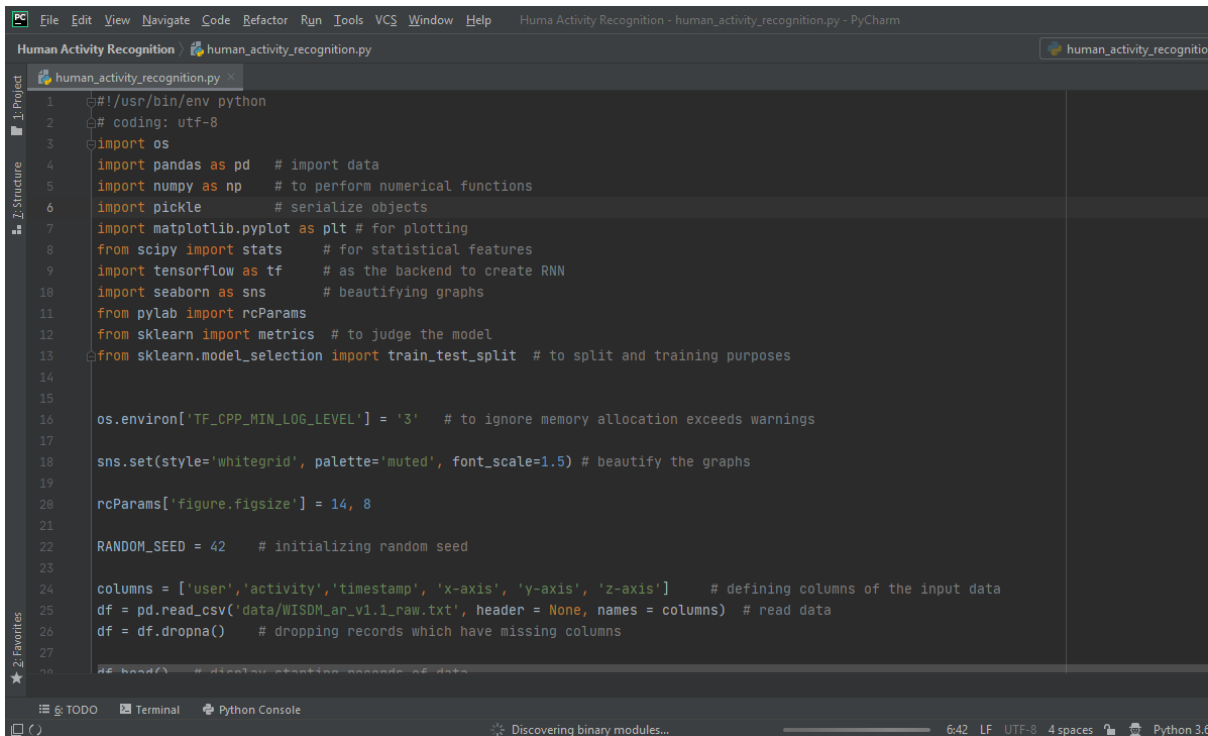


Figure 5.1 : Input data

It's a must to check the dataset and get an understanding about what we are dealing with at the first place. It was observed that the data was collected from 36 subjects and there are six columns in the text file. Six columns can be labeled as user id, activity, time, x, y and z axis data. Activities are walking, standing, sitting, jogging, walking downstairs and walking upstairs.

## 5.1.2 Data Import



```
1  #!/usr/bin/env python
2  # coding: utf-8
3  import os
4  import pandas as pd # import data
5  import numpy as np  # to perform numerical functions
6  import pickle       # serialize objects
7  import matplotlib.pyplot as plt # for plotting
8  from scipy import stats # for statistical features
9  import tensorflow as tf # as the backend to create RNN
10 import seaborn as sns # beautifying graphs
11 from pylab import rcParams
12 from sklearn import metrics # to judge the model
13 from sklearn.model_selection import train_test_split # to split and training purposes
14
15
16 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # to ignore memory allocation exceeds warnings
17
18 sns.set(style='whitegrid', palette='muted', font_scale=1.5) # beautify the graphs
19
20 rcParams['figure.figsize'] = 14, 8
21
22 RANDOM_SEED = 42 # initializing random seed
23
24 columns = ['user', 'activity', 'timestamp', 'x-axis', 'y-axis', 'z-axis'] # defining columns of the input data
25 df = pd.read_csv('data/WISDM_ar_v1.1_raw.txt', header = None, names = columns) # read data
26 df = df.dropna() # dropping records which have missing columns
27
28 # df.head() # display starting records of data
```

Figure 5.2 : Import Data

It is needed to import relevant dependencies as the first step. While running the code multiple warnings were displayed saying the memory allocation exceeded. Since it's just a warning and the output terminal should be clear to differentiate the outputs, used `os.environ[]` to ignore these errors. Pandas was used to read the text file and the `dropna()` [Figure 5.2] command was used to drop any lines with missing data. Tensorflow will be used as the backend to create the LSTM neural network. So, the importing task is completed but how can we ensure that the data were imported successfully?

### 5.1.3 Data Analyses

```
28 df.head() # display starting records of data
29 df.info() # display a summary of loaded data frame
30
31 df['activity'].value_counts().plot(kind='bar', title='Training examples by activity type');
32 df['user'].value_counts().plot(kind='bar', title='Training examples by user');
33
34 # plot activities by x,y,z
35 def plot_activity(activity, df):
36     data = df[df['activity'] == activity][['x-axis', 'y-axis', 'z-axis'][:200]
37     axis = data.plot(subplots=True, figsize=(16, 12),
38                     title=activity)
39     for ax in axis:
40         ax.legend(loc='lower left', bbox_to_anchor=(1.0, 0.5))
41
42 plot_activity("Sitting", df)
43 plot_activity("Standing", df)
44 plot_activity("Walking", df)
45 plot_activity("Jogging", df)
46 plot_activity("Downstairs", df)
47 plot_activity("Upstairs", df)
```

Figure 5.3 : Analyzing Data

So after importing the data we can use head and info functions [Figure 5.3Figure 5.4] to display the head values and to grab a summary of our imported dataset.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1098203 entries, 0 to 1098203
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   user        1098203 non-null  int64
1   activity    1098203 non-null  object
2   timestamp   1098203 non-null  int64
3   x-axis      1098203 non-null  float64
4   y-axis      1098203 non-null  float64
5   z-axis      1098203 non-null  float64
dtypes: float64(3), int64(2), object(1)
memory usage: 58.7+ MB
```

Figure 5.4 : Check the Imported Data

The above output ensures that the data got imported correctly and we can also see the size of the imported dataset and it is possible to compare with the real text file size to further ensure that the import is a success.

It is always better to understand about the dataset by creating charts [Figure 5.3], since it makes the process of comparison and identification easier. As shown in the below image [Figure 5.5] user with the id number 20 has contributed for over 400,000 records while the most of the users have only maintained a value below 50000. This graph suggests that the data gathering process was not happened in a justified manner because one or two persons have more influence against the dataset. However, because of the variations that reside in the recorded data, it will be enough to build the required model.

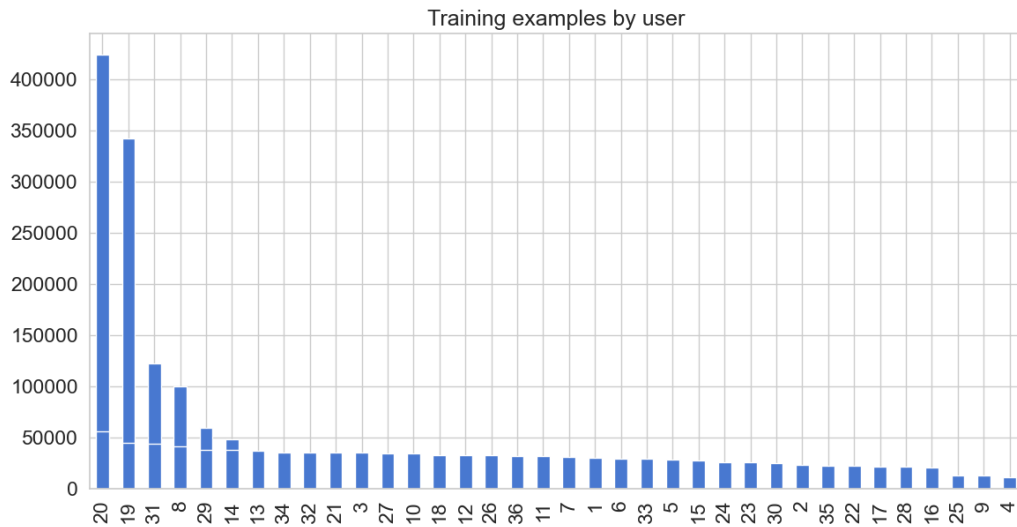


Figure 5.5 : Training Examples by User Bar Graph

Below are the charts related to each activity with each axis. Variations of the six different activities ensure that it is reasonable to use the imported data to make a model. Most of the activities are showing repetitive movements [Figure 5.8, Figure 5.9], which can be used to easily differentiate the activities. Even though the sitting [Figure 5.6] and standing [Figure 5.7] charts don't have sudden accelerations or much of a periodic behavior, the two activities can be identified by x,y and z relative magnitudes. Also, for most of the activities the y axis shows the largest accelerations. The reason for that is the Earth's gravitational pull. This forces the accelerometer to measure a value of  $9.8 \text{ m/s}^2$  in the direction of the Earth's center.



Figure 5.6 : Sitting Axis Chart

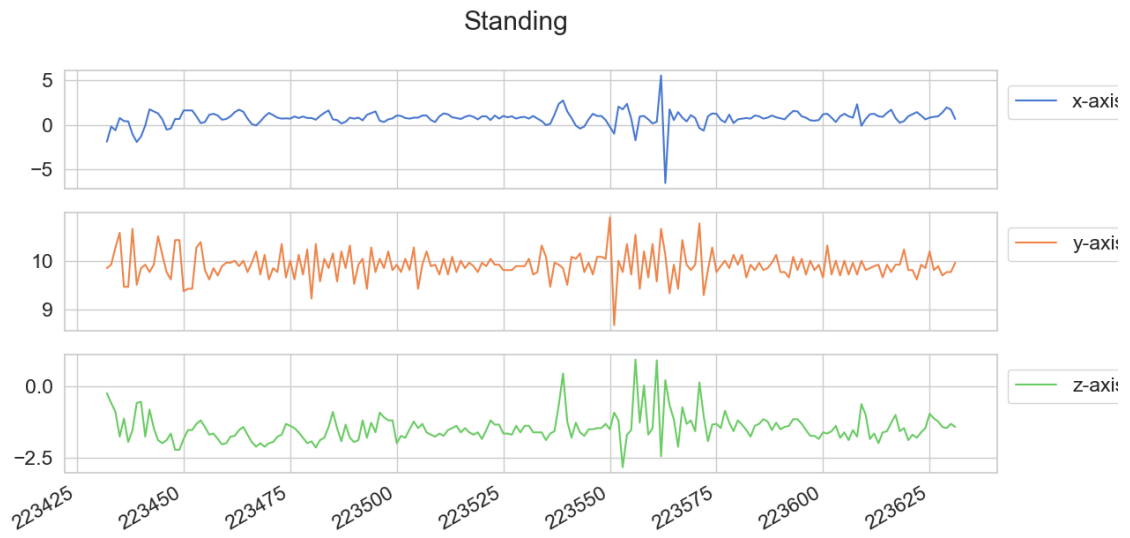


Figure 5.7 : Standing Axis Chart



Figure 5.8 : Walking Axis Chart



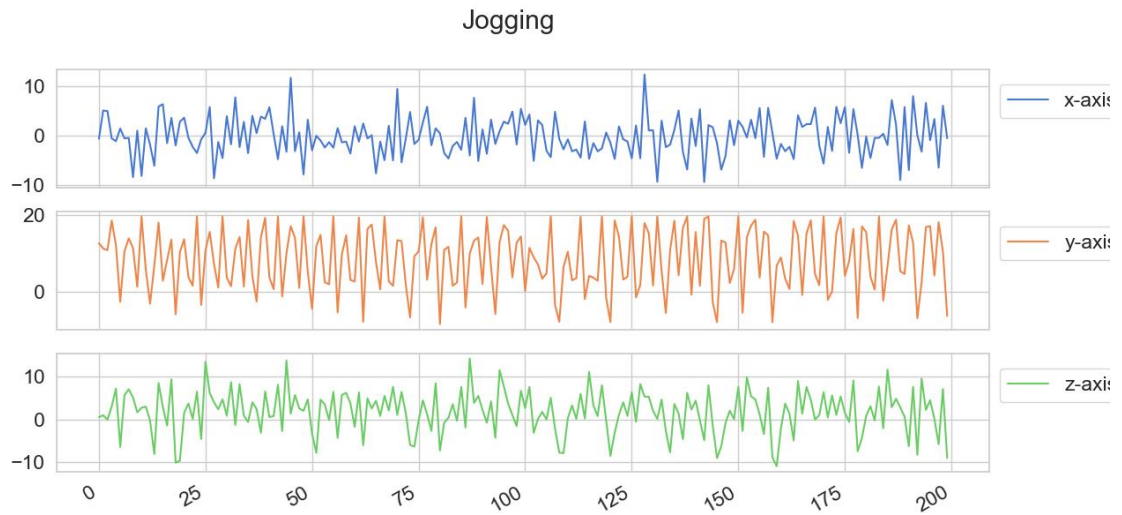


Figure 5.9 : Jogging Axis Chart

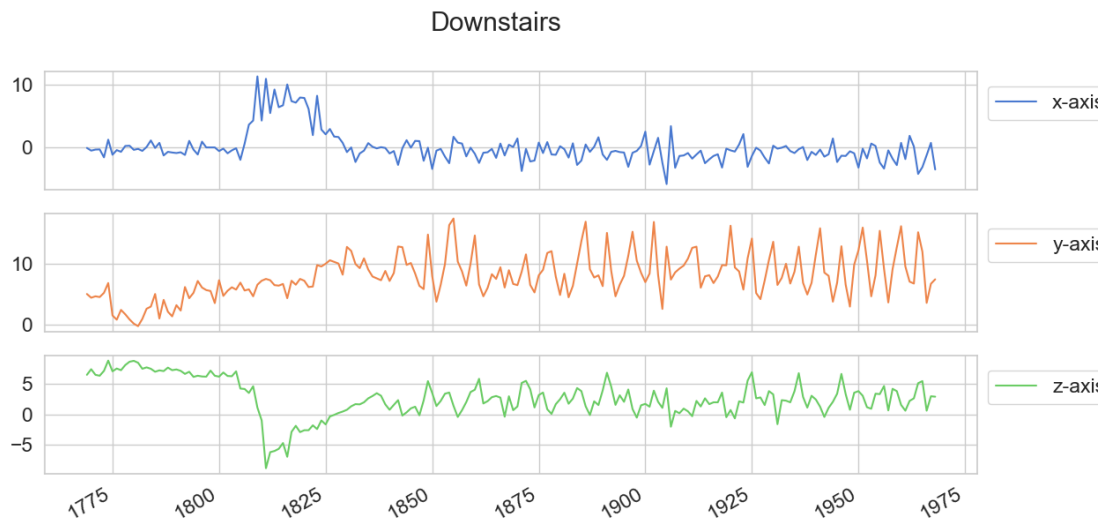


Figure 5.10 : Downstairs Axis Chart

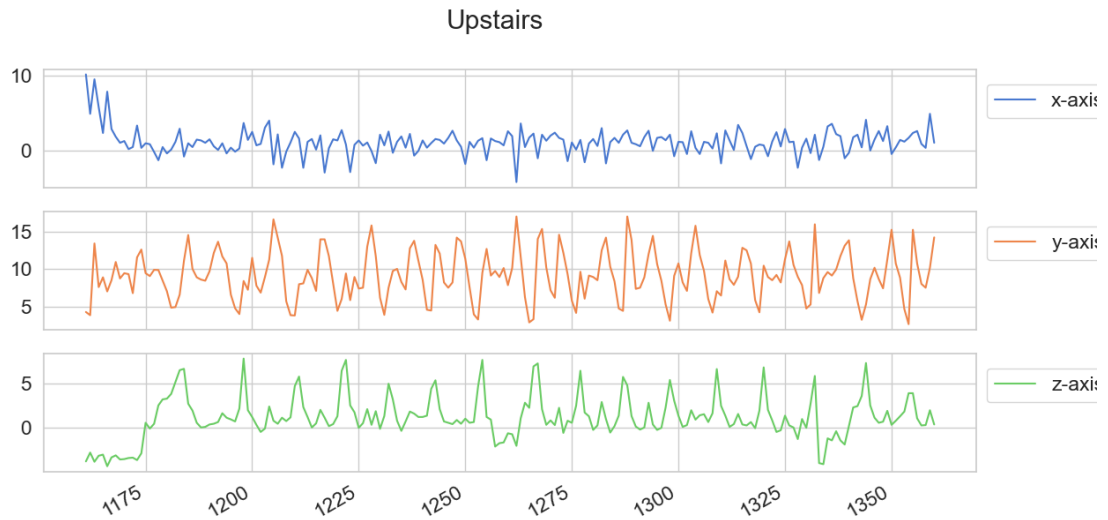


Figure 5.11 : Upstairs Axis Chart

#### 5.1.4 Preprocessing the Data

```

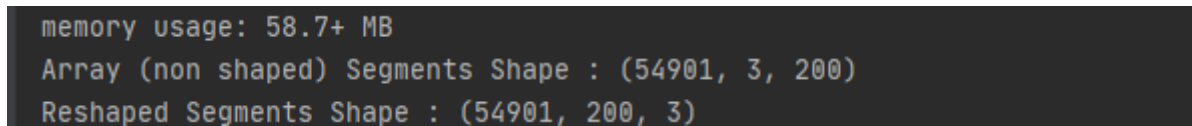
human_activity_recognition.py
49 # separating data to blocks and labelling to feed into LSTM network
50 N_TIME_STEPS = 200
51 N_FEATURES = 3
52 step = 20
53 segments = []
54 labels = []
55 for i in range(0, len(df) - N_TIME_STEPS, step):
56     xs = df['x-axis'].values[i: i + N_TIME_STEPS]
57     ys = df['y-axis'].values[i: i + N_TIME_STEPS]
58     zs = df['z-axis'].values[i: i + N_TIME_STEPS]
59     label = stats.mode(df['activity'][i: i + N_TIME_STEPS])[0][0]
60     segments.append([xs, ys, zs])
61     labels.append(label)
62
63
64 print("Array (non shaped) Segments Shape :", np.array(segments).shape) # checking the data shape
65
66
67 reshaped_segments = np.asarray(segments, dtype= np.float32).reshape(-1, N_TIME_STEPS, N_FEATURES)
68 labels = np.asarray(pd.get_dummies(labels), dtype= np.float32) # labelling activities to a float version
69
70 print("Reshaped Segments Shape :", reshaped_segments.shape)
71

```

Figure 5.12 : Reshaping Data

Imported data have to go through a transformation prior to feeding to the LSTM network. Because there are certain requirements for every neural network algorithm to provide a better output. Firstly, the data should be separated as 200 sized data blocks or as data chunks. By using the 'for' loop, we are labeling activity sets with 200 records with the highest available activity. Suppose 150 records of a 200 block belongs to walking and 50 represents other activities. Then we are going to label the mentioned 200 set as a walking set. This process is similar to creating the examples as mentioned in the feature extraction [Feature Extraction] section. The scipy, stats will be used to generate this labelling process by using the mode [Figure 5.12] function. After printing the current shape of the data, it was realized that the

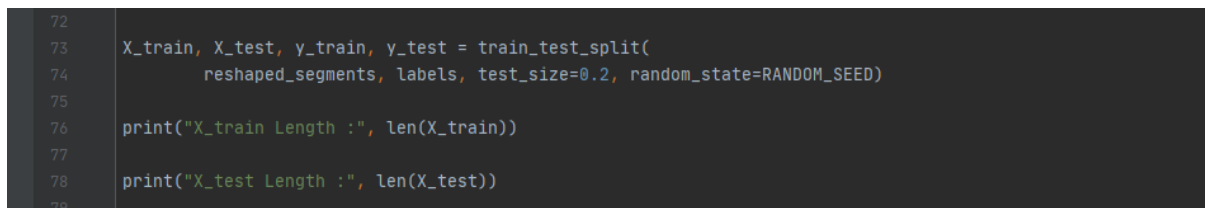
data is arranged in a messy and an unacceptable manner. In the below screenshot we can see that there are 54901 records, 3 rows and 200 columns [Figure 5.13]. That doesn't look like something that can be fed to a neural network. Using the functions of numpy, it was possible to reshape and after printing the reshaped output in the terminal, it was observed that the column number and the row number got interchanged. Now we are ready to proceed with the reshaped data to proceed with the splitting.



```
memory usage: 58.7+ MB
Array (non shaped) Segments Shape : (54901, 3, 200)
Reshaped Segments Shape : (54901, 200, 3)
```

Figure 5.13 : Output of Reshaping Data

### 5.1.5 Splitting Data



```
72
73 X_train, X_test, y_train, y_test = train_test_split(
74     reshaped_segments, labels, test_size=0.2, random_state=RANDOM_SEED)
75
76 print("X_train Length :", len(X_train))
77
78 print("X_test Length :", len(X_test))
79
```

Figure 5.14 : Splitting Data

We have to train the model to predict and also need to test its accuracy and loss counts before releasing it to the public. Imported and reshaped data can be split as train and test data. At the line 74, `test_size=0.2` suggests that 20% of the whole data will be used to test [Figure 5.14] the model and rest of the 80% data will be used to train the model. After checking the train and test values by using a simple print command, we can see that it satisfies the percentage requirements [Figure 5.15].



```
X_train Length : 43920
X_test Length : 10981
```

Figure 5.15 : Printing Split Data

### 5.1.6 Building the Model

After creating the test and the train data, it's time to build our model. First we have to define our weights and biases because as we saw in our LSTM equations [Technical Background], it's a must to proceed with the neural network. After transposing and reshaping the data, relu will be used as the activation function in the hidden layer [Figure 5.16]. This is where the magic happens. Then it's time to create two LSTM layers. After stacking it together, we can build the neural network by using these stacked LSTM layers.

```

80 # Building the model
81 N_CLASSES = 6
82 N_HIDDEN_UNITS = 64 # unit size for each LSTM layer
83
84 def create_LSTM_model(inputs):
85     W = {
86         'hidden': tf.Variable(tf.random_normal([N_FEATURES, N_HIDDEN_UNITS])),
87         'output': tf.Variable(tf.random_normal([N_HIDDEN_UNITS, N_CLASSES]))
88     }
89     biases = {
90         'hidden': tf.Variable(tf.random_normal([N_HIDDEN_UNITS], mean=1.0)),
91         'output': tf.Variable(tf.random_normal([N_CLASSES]))
92     }
93
94     X = tf.transpose(inputs, [1, 0, 2])
95     X = tf.reshape(X, [-1, N_FEATURES]) # transforming and reshaping to feed into the model
96     hidden = tf.nn.relu(tf.matmul(X, W['hidden']) + biases['hidden']) # using relu as the activation function for the hidden layer - this is the LSTM main fu
97     hidden = tf.split(hidden, N_TIME_STEPS, 0) # splitting data to 200
98
99     # Stack 2 LSTM layers
100     lstm_layers = [tf.contrib.rnn.BasicLSTMCell(N_HIDDEN_UNITS, forget_bias=1.0) for _ in range(2)] # creating 2 lstm layers
101     lstm_layers = tf.nn.rnn_cell.MultiRNNCell(lstm_layers) # stacking layers
102
103     outputs, _ = tf.nn.nn.static_rnn(lstm_layers, hidden, dtype=tf.float32) # creating the LSTM network from 2 layers
104

```

Figure 5.16 : Building the Model Part 1

The output will be returned at the end of the create\_LSTM\_model function [Figure 5.17].

```

105     # return the output of the whole RNN
106     lstm_last_output = outputs[-1]
107
108     return tf.matmul(lstm_last_output, W['output']) + biases['output']
109
110
111 tf.reset_default_graph()
112 # feed data to tensorflow model as x and y
113 X = tf.placeholder(tf.float32, [None, N_TIME_STEPS, N_FEATURES], name="input")
114 Y = tf.placeholder(tf.float32, [None, N_CLASSES])
115
116
117 pred_Y = create_LSTM_model(X) # calling the function
118
119 pred_softmax = tf.nn.softmax(pred_Y, name="y_") # softmax as the activation function for the output layer
120

```

Figure 5.17 : Building the Model Part 2

The default graphs should be reset because there might be data which can affect our future chart outputs. Then, we can call the LSTM model function by feeding the data (x = inputs). We are using softmax [Figure 5.17] as the output layer's activation function since it is capable of providing values as a probability which can summed up to one. That way it will be easy for the users to identify the outputs related to specific activities.

```

121
122 L2_LOSS = 0.0015 # to prevent from over fitting
123
124 l2 = L2_LOSS * \
125     sum(tf.nn.l2_loss(tf_var) for tf_var in tf.trainable_variables())
126
127 loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred_Y, labels=Y)) + l2
128
129
130
131 LEARNING_RATE = 0.0025
132
133 optimizer = tf.train.AdamOptimizer(learning_rate=LEARNING_RATE).minimize(loss) # setting adam optimizer for learning
134
135 correct_pred = tf.equal(tf.argmax(pred_softmax, 1), tf.argmax(Y, 1)) # finding the highest probability prediction
136 accuracy = tf.reduce_mean(tf.cast(correct_pred, dtype=tf.float32))
137
138 N_EPOCHS = 50
139 BATCH_SIZE = 1024
140

```

Figure 5.18 : Building the Model Part 3

There's an overfitting problem in most of the networks and the issue can be addressed by using reduce mean functions and by setting Adam optimizer for learning [Figure 5.18]. After that epochs and batch size should be decided to start with the training process.

### 5.1.7 Training the Model

We are going to save the data to the disk and to continually save everything we are creating a dictionary to hold the current data.

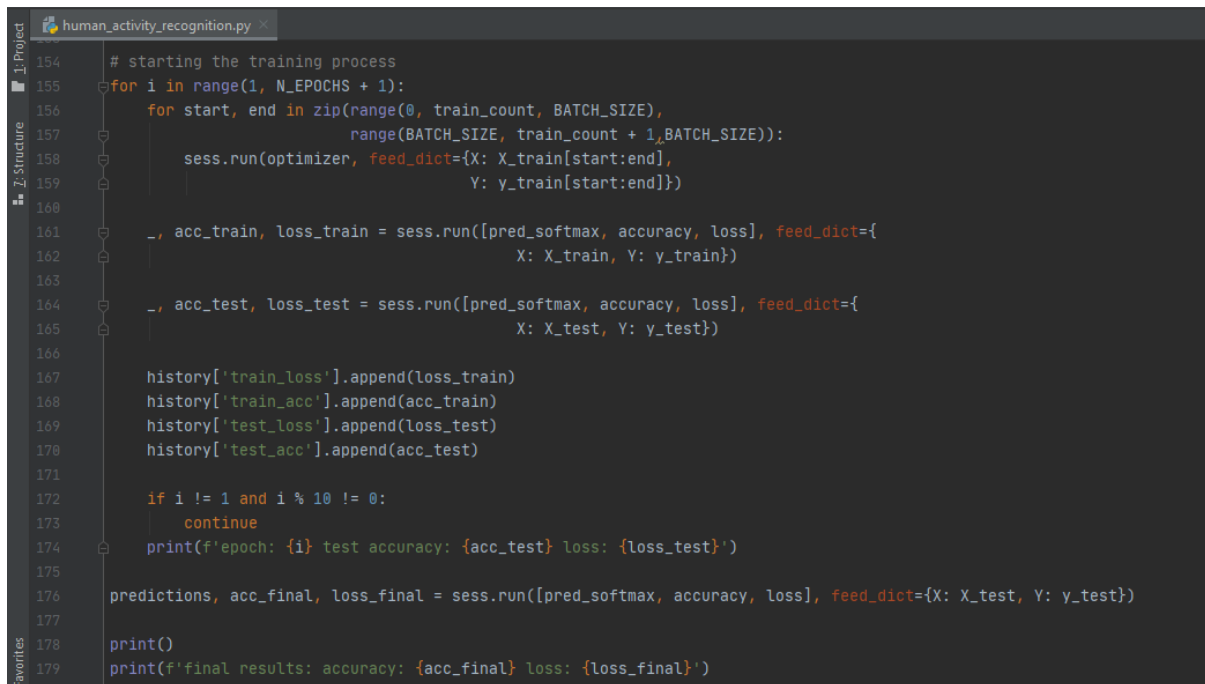
```

human_activity_recognition.py x
140
141 saver = tf.train.Saver() # to save the training process in disk
142
143 # saving everything in a dictionary to use to plotting and perform analysis of the network
144 history = dict(train_loss=[],
145               train_acc=[],
146               test_loss=[],
147               test_acc=[])
148
149 sess=tf.InteractiveSession() # create a session to start with training the LSTM model
150 sess.run(tf.global_variables_initializer())
151
152 train_count = len(X_train)

```

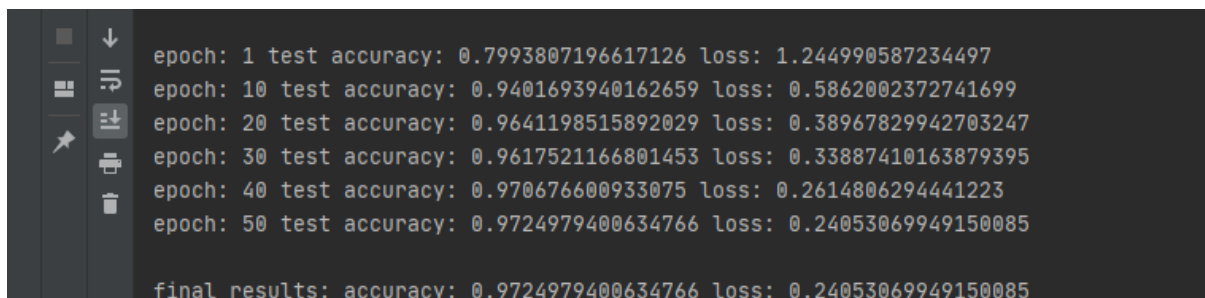
Figure 5.19 : Training the Model Part 1

Then we can start with the training process by using a 'for' loop and it will run satisfying the batch size and the number of epochs. As mentioned we have to save everything to our dictionary by appending the data to it. Also the data will be printed by 10 epochs [Figure 5.20], since the value differences of close by epochs will be very low. Output will be clearer and the progress will be more identifiable by printing the epochs by 10 because of the less number of terminal outputs [Figure 5.21].



```
154 # starting the training process
155 for i in range(1, N_EPOCHS + 1):
156     for start, end in zip(range(0, train_count, BATCH_SIZE),
157                           range(BATCH_SIZE, train_count + 1, BATCH_SIZE)):
158         sess.run(optimizer, feed_dict={X: X_train[start:end],
159                                         Y: y_train[start:end]})
160
161     _, acc_train, loss_train = sess.run([pred_softmax, accuracy, loss], feed_dict={
162                                         X: X_train, Y: y_train})
163
164     _, acc_test, loss_test = sess.run([pred_softmax, accuracy, loss], feed_dict={
165                                         X: X_test, Y: y_test})
166
167     history['train_loss'].append(loss_train)
168     history['train_acc'].append(acc_train)
169     history['test_loss'].append(loss_test)
170     history['test_acc'].append(acc_test)
171
172     if i != 1 and i % 10 != 0:
173         continue
174     print(f'epoch: {i} test accuracy: {acc_test} loss: {loss_test}')
175
176 predictions, acc_final, loss_final = sess.run([pred_softmax, accuracy, loss], feed_dict={X: X_test, Y: y_test})
177
178 print()
179 print(f'final results: accuracy: {acc_final} loss: {loss_final}')
```

Figure 5.20: Training the Model Part 2



```
epoch: 1 test accuracy: 0.7993807196617126 loss: 1.244990587234497
epoch: 10 test accuracy: 0.9401693940162659 loss: 0.5862002372741699
epoch: 20 test accuracy: 0.9641198515892029 loss: 0.38967829942703247
epoch: 30 test accuracy: 0.9617521166801453 loss: 0.33887410163879395
epoch: 40 test accuracy: 0.970676600933075 loss: 0.2614806294441223
epoch: 50 test accuracy: 0.9724979400634766 loss: 0.24053069949150085

final results: accuracy: 0.9724979400634766 loss: 0.24053069949150085
```

Figure 5.21 : Final Accuracy and Loss

As shown in the above image, the program is displaying accuracy levels and the loss values for the displayed epochs. Tada! We were able to achieve a better accuracy near to 98% with our LSTM approach. Also notice that the loss count is getting decreased with time ensuring that our model will perform well in predicting future inputs. We can also use pickle to serialize our algorithm and save it to the disk for future use. We are going to use the data in the created history dictionary to assist with that operation. Then we can deserialize it back to the model to proceed with the prediction [Figure 5.22].

```

181
182
183 pickle.dump(predictions, open("predictions.p", "wb"))
184 pickle.dump(history, open("history.p", "wb"))
185 tf.train.write_graph(sess.graph_def, '.', './checkpoint/har.pbtxt')
186 saver.save(sess, save_path='./checkpoint/har.ckpt')
187 sess.close()
188
189
190 history = pickle.load(open("history.p", "rb"))
191 predictions = pickle.load(open("predictions.p", "rb"))
192

```

Figure 5.22 : Use of Pickle

### 5.1.8 Trained Model Evaluation

After training the model and testing for accuracy and all, we can plot charts and see the final outcome in a visualized manner to have a better understanding.

```

human_activity_recognition.py x
193
194 plt.figure(figsize=(12, 8))
195
196 plt.plot(np.array(history['train_loss']), "r--", label="Train loss")
197 plt.plot(np.array(history['train_acc']), "g--", label="Train accuracy")
198
199 plt.plot(np.array(history['test_loss']), "r-", label="Test loss")
200 plt.plot(np.array(history['test_acc']), "g-", label="Test accuracy")
201
202 plt.title("Training session's progress over iterations")
203 plt.legend(loc='upper right', shadow=True)
204 plt.ylabel('Training Progress (Loss or Accuracy values)')
205 plt.xlabel('Training Epoch')
206 plt.ylim(0)
207
208 plt.show()

```

Figure 5.23 : Plotting Accuracy and Loss

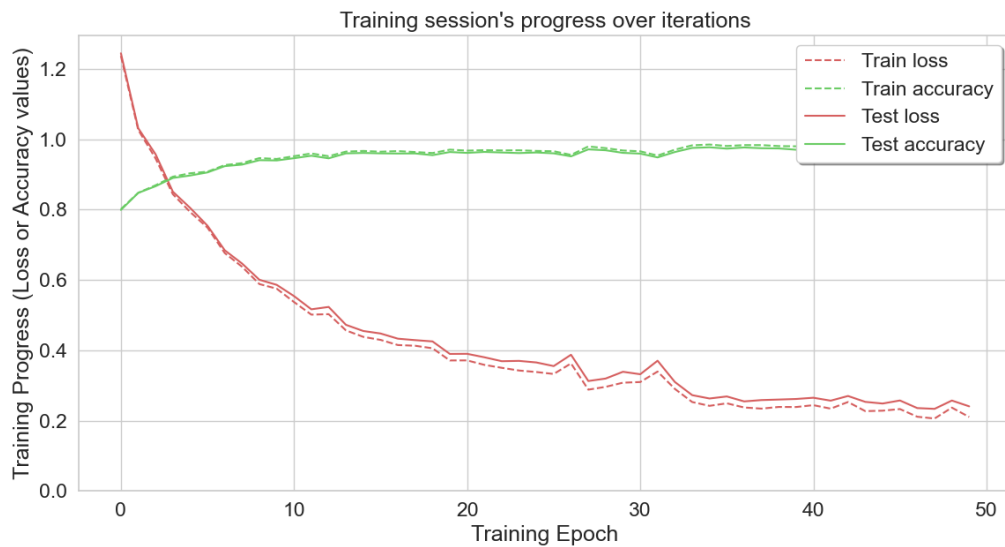


Figure 5.24 : Accuracy and Loss Chart

As shown in the above chart the model is performing well with an accuracy level almost close to 98%. Also notice the significant decrease in the loss count.

```

209
210 # confusion matrix
211 LABELS = ['Downstairs', 'Jogging', 'Sitting', 'Standing', 'Upstairs', 'Walking']
212
213
214 max_test = np.argmax(y_test, axis=1)
215 max_predictions = np.argmax(predictions, axis=1)
216 confusion_matrix = metrics.confusion_matrix(max_test, max_predictions)
217
218 plt.figure(figsize=(16, 14))
219 sns.heatmap(confusion_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d")
220 plt.title("Confusion matrix")
221 plt.ylabel('True label')
222 plt.xlabel('Predicted label')
223 plt.show()

```

Figure 5.25 : Building the Confusion Matrix

After that we can build the confusion matrix to see the conflicts and confusion levels which occur when predicting the correct activity. We can expect the same behavior when running the application because it will provide the probabilities as percentages such as 98% for walking, 0.1% for jogging etc. even though it's a walking activity. In other words, not a single activity will be predicted with a 100% accuracy.



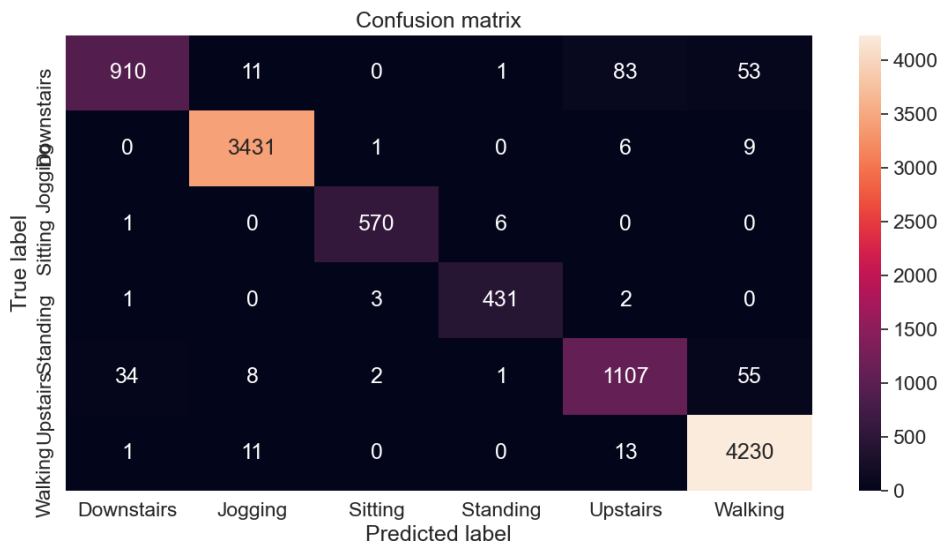


Figure 5.26 : Confusion Matrix

According to the confusion matrix, can further ensure that the accuracy levels of the model is acceptable. Downstairs and upstairs activities have a significant conflict with other activities. By looking at the matrix [Figure 5.26] it can be assumed that the predictions regarding downstairs and upstairs won't be accurate as other activities.

### 5.1.9 Freezing the Trained Model

```

224
225 # exporting the model to use in android app
226 from tensorflow.python.tools import freeze_graph
227
228 MODEL_NAME = 'har'
229
230 input_graph_path = 'checkpoint/' + MODEL_NAME + '.pbtxt'
231 checkpoint_path = './checkpoint/' + MODEL_NAME + '.ckpt'
232 restore_op_name = "save/restore_all"
233 filename_tensor_name = "save/Const:0"
234 output_frozen_graph_name = 'frozen_' + MODEL_NAME + '.pb'
235
236 freeze_graph.freeze_graph(input_graph_path, input_saver="",
237                           input_binary=False, input_checkpoint=checkpoint_path,
238                           output_node_names="y_", restore_op_name="save/restore_all",
239                           filename_tensor_name="save/Const:0",
240                           output_graph=output_frozen_graph_name, clear_devices=True, initializer_nodes="")
241
242
243

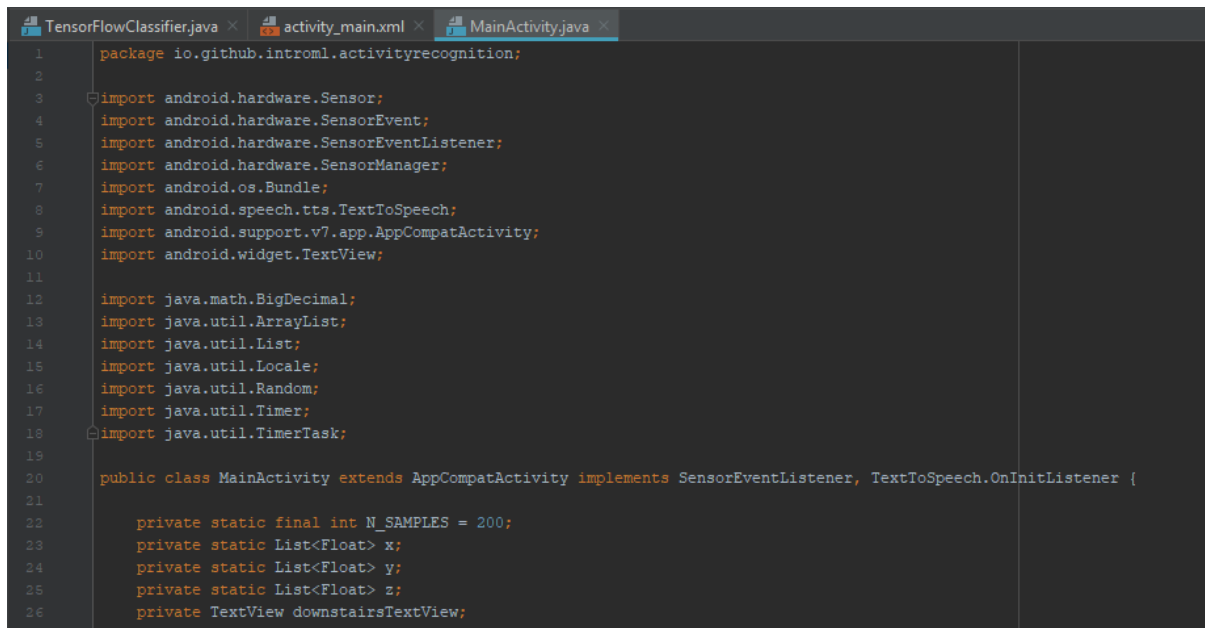
```

Figure 5.27 : Freezing the Trained Model

After doing all the hard work, now we just need to freeze the trained model so it can be used in the android application to predict the activities. We are going to freeze the model by using the saved files related to pickle in the folder named checkpoint. After running this section, the generated frozen\_har.pb (protocol buffer) can be used in the android application to perform the real time prediction.

## 5.2 Android Code

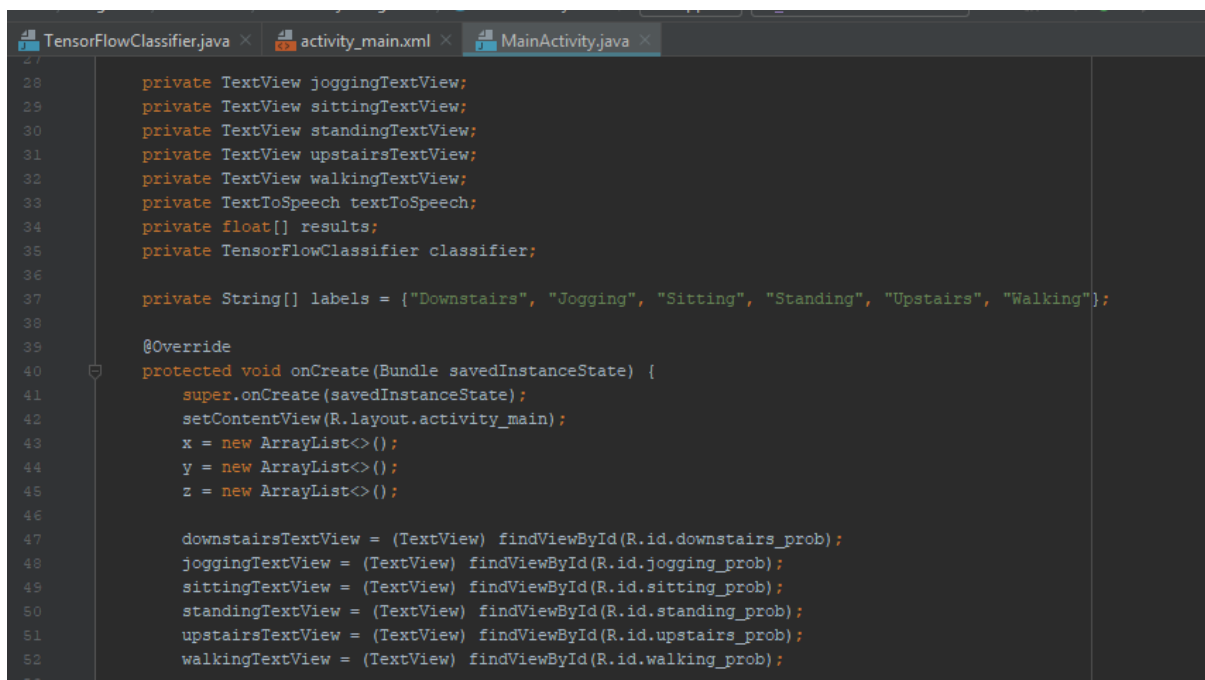
### 5.2.1 Main Activity



```
1 package io.github.introml.activityrecognition;
2
3 import android.hardware.Sensor;
4 import android.hardware.SensorEvent;
5 import android.hardware.SensorEventListener;
6 import android.hardware.SensorManager;
7 import android.os.Bundle;
8 import android.speech.tts.TextToSpeech;
9 import android.support.v7.app.AppCompatActivity;
10 import android.widget.TextView;
11
12 import java.math.BigDecimal;
13 import java.util.ArrayList;
14 import java.util.List;
15 import java.util.Locale;
16 import java.util.Random;
17 import java.util.Timer;
18 import java.util.TimerTask;
19
20 public class MainActivity extends AppCompatActivity implements SensorEventListener, TextToSpeech.OnInitListener {
21
22     private static final int N_SAMPLES = 200;
23     private static List<Float> x;
24     private static List<Float> y;
25     private static List<Float> z;
26     private TextView downstairsTextView;
```

Figure 5.28 : Main Android Part 1

Android application is built with a basic interface and features, since our main focus was to build the model and increase the accuracy of the predictions. In the above image [Figure 5.28] importing packages and list creation to hold the input data is displayed as the content.



```
28 private TextView joggingTextView;
29 private TextView sittingTextView;
30 private TextView standingTextView;
31 private TextView upstairsTextView;
32 private TextView walkingTextView;
33 private TextToSpeech textToSpeech;
34 private float[] results;
35 private TensorFlowClassifier classifier;
36
37 private String[] labels = {"Downstairs", "Jogging", "Sitting", "Standing", "Upstairs", "Walking"};
38
39 @Override
40 protected void onCreate(Bundle savedInstanceState) {
41     super.onCreate(savedInstanceState);
42     setContentView(R.layout.activity_main);
43     x = new ArrayList<>();
44     y = new ArrayList<>();
45     z = new ArrayList<>();
46
47     downstairsTextView = (TextView) findViewById(R.id.downstairs_prob);
48     joggingTextView = (TextView) findViewById(R.id.jogging_prob);
49     sittingTextView = (TextView) findViewById(R.id.sitting_prob);
50     standingTextView = (TextView) findViewById(R.id.standing_prob);
51     upstairsTextView = (TextView) findViewById(R.id.upstairs_prob);
52     walkingTextView = (TextView) findViewById(R.id.walking_prob);
53 }
```

Figure 5.29 : Main Android Part 2

After declaring the variables [Figure 5.29], need to map those variables with the id of the interface elements by using `findViewById` function. So that the created variables can refer to these sections and the output can be set accordingly to the id.

```

53
54         classifier = new TensorFlowClassifier(getApplicationContext());
55
56         textToSpeech = new TextToSpeech(this, this);
57         textToSpeech.setLanguage(Locale.US);
58     }
59
60     @Override
61     public void onInit(int status) {
62         Timer timer = new Timer();
63         timer.scheduleAtFixedRate(new TimerTask() {
64             @Override
65             public void run() {
66                 if (results == null || results.length == 0) {
67                     return;
68                 }
69                 float max = -1;
70                 int idx = -1;
71                 for (int i = 0; i < results.length; i++) {
72                     if (results[i] > max) {
73                         idx = i;
74                         max = results[i];
75                     }
76                 }
77
78                 textToSpeech.speak(labels[idx], TextToSpeech.QUEUE_ADD, null, Integer.toString(new Random().nextInt()));
79             }

```

Figure 5.30 : Main Android Part 3

Also the application has a voice based mechanism to speak out the predicted activity. Above image functions are used to implement its behavior with delays [Figure 5.30].

```

90         }, 2000, 5000);
91     }
92
93     protected void onPause() {
94         getSensorManager().unregisterListener(this);
95         super.onPause();
96     }
97
98     protected void onResume() { //registering listener again for the sensor
99         super.onResume();
100         getSensorManager().registerListener(this, getSensorManager().getDefaultSensor(Sensor.TYPE_ACCELEROMETER), SensorManager.SENSOR_DELAY_GA
101     }
102
103     @Override
104     public void onSensorChanged(SensorEvent event) {
105         activityPrediction();
106         x.add(event.values[0]);
107         y.add(event.values[1]);
108         z.add(event.values[2]);
109     }

```

Figure 5.31 : Main Android Part 4

Also it is mandatory to update x, y and z values when the sensor values get changed [Figure 5.31].

```

105
106     private void activityPrediction() {
107         if (x.size() == N_SAMPLES && y.size() == N_SAMPLES && z.size() == N_SAMPLES) {
108             List<Float> data = new ArrayList<>();
109             data.addAll(x);
110             data.addAll(y);
111             data.addAll(z);
112
113             results = classifier.predictProbabilities(toFloatArray(data));
114
115             downstairsTextView.setText(Float.toString(round(results[0], 2)));
116             joggingTextView.setText(Float.toString(round(results[1], 2)));
117             sittingTextView.setText(Float.toString(round(results[2], 2)));
118             standingTextView.setText(Float.toString(round(results[3], 2)));
119             upstairsTextView.setText(Float.toString(round(results[4], 2)));
120             walkingTextView.setText(Float.toString(round(results[5], 2)));
121
122             x.clear();
123             y.clear();
124             z.clear();
125         }
126     }

```

Figure 5.32 : Main Android Part 5

Finally, it's time to call the activity prediction function to start with the prediction. So the data taken as inputs will be passed to a list called 'data'. The list 'data' will be passed as the input to the predictProbabilities function. Let's identify what is inside the classifier function later. The rest of the code is setting up the returned array of values according to their called positions to display the relevant value in the application probability field [Figure 5.36]. Then the current values getting cleared to take new inputs to make the prediction. Below image explains two functions which were used to round up and convert the decimal values to an acceptable manner [Figure 5.33]. Now, let's see what happens to the data passed to the predictProbabilities function.

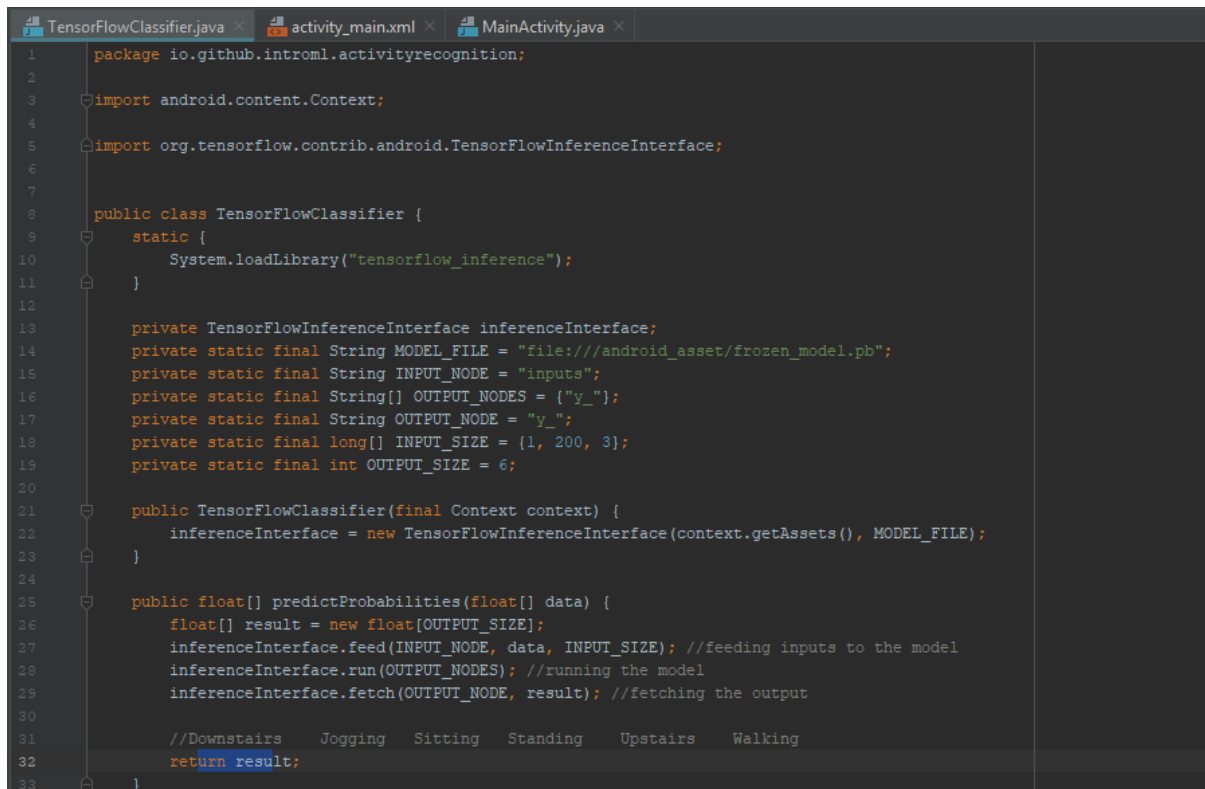
```

127
128     private float[] toFloatArray(List<Float> list) {
129         int i = 0;
130         float[] array = new float[list.size()];
131
132         for (Float f : list) {
133             array[i++] = (f != null ? f : Float.NaN);
134         }
135         return array;
136     }
137
138     private static float round(float d, int decimalPlace) {
139         BigDecimal bd = new BigDecimal(Float.toString(d));
140         bd = bd.setScale(decimalPlace, BigDecimal.ROUND_HALF_UP);
141         return bd.floatValue();
142     }
143
144     private SensorManager getSensorManager() {
145         return (SensorManager) getSystemService(SENSOR_SERVICE);
146     }
147
148 }
149

```

Figure 5.33 : Main Android Part 6

## 5.2.2 Model Import and Prediction



```
1 package io.github.introml.activityrecognition;
2
3 import android.content.Context;
4
5 import org.tensorflow.contrib.android.TensorFlowInferenceInterface;
6
7
8 public class TensorFlowClassifier {
9     static {
10         System.loadLibrary("tensorflow_inference");
11     }
12
13     private TensorFlowInferenceInterface inferenceInterface;
14     private static final String MODEL_FILE = "file:///android_asset/frozen_model.pb";
15     private static final String INPUT_NODE = "inputs";
16     private static final String[] OUTPUT_NODES = {"y_"};
17     private static final String OUTPUT_NODE = "y_";
18     private static final long[] INPUT_SIZE = {1, 200, 3};
19     private static final int OUTPUT_SIZE = 6;
20
21     public TensorFlowClassifier(final Context context) {
22         inferenceInterface = new TensorFlowInferenceInterface(context.getAssets(), MODEL_FILE);
23     }
24
25     public float[] predictProbabilities(float[] data) {
26         float[] result = new float[OUTPUT_SIZE];
27         inferenceInterface.feed(INPUT_NODE, data, INPUT_SIZE); //feeding inputs to the model
28         inferenceInterface.run(OUTPUT_NODES); //running the model
29         inferenceInterface.fetch(OUTPUT_NODE, result); //fetching the output
30
31         //Downstairs  Jogging  Sitting  Standing  Upstairs  Walking
32         return result;
33     }
34 }
```

Figure 5.34 : Model Import and Prediction

This is where all the magic happens because the frozen trained model is getting imported as shown in the 14<sup>th</sup> line. 200 rows, 3 columns and 6 activities were used in creating the LSTM network. Line 18<sup>th</sup> and line 19<sup>th</sup> indicates that the LSTM requirements are satisfied. Use of the tensorflow makes the rest of the part easier. To start predicting it is needed to feed the input to the model, run the model with the given input and grab the output as an array (result array). This returned array elements was called by their positions accordingly from the setText functions [Figure 5.32] to display the output to the user.

## 5.2.3 User Interface

The below shown image is the simple interface which was developed to display the output to a user. Probability column will display the relevant probability values according to the x, y and z data and the activity with the highest probability will be echoed to the user by the audio feature.

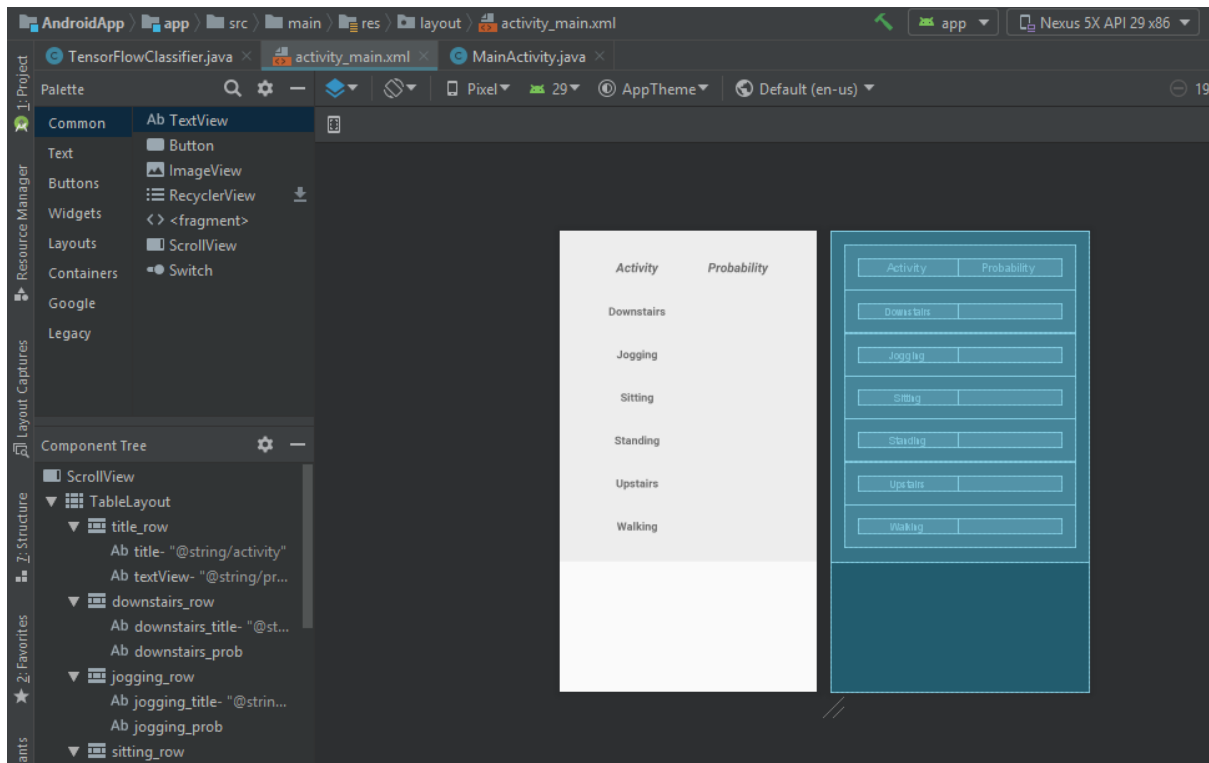


Figure 5.35 : User Interface

#### 5.2.4 Application in Action

Below two images were taken when the application was functioning. First image [Figure 5.36] is similar to a sitting situation since the phone is resting on a surface. The research was conducted by carrying the phone on the subjects' trouser pockets. Since the situation is same, the application is displaying the user is performing the sitting activity. Then in the second image [Figure 5.37] the phone was set up in the way that it'll fit in our pockets when we stand up. So the application predicting that the user is performing a standing activity as expected.

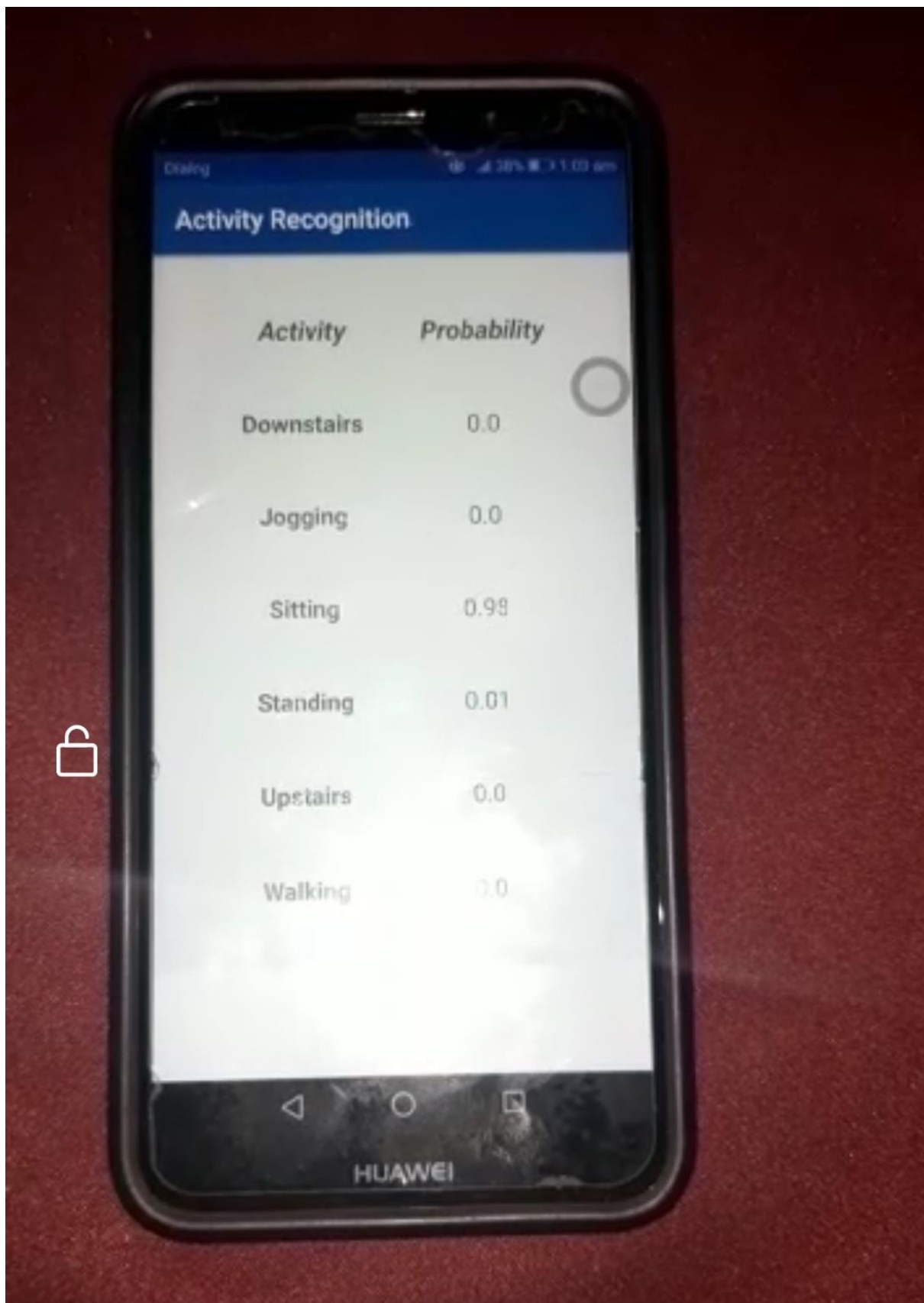
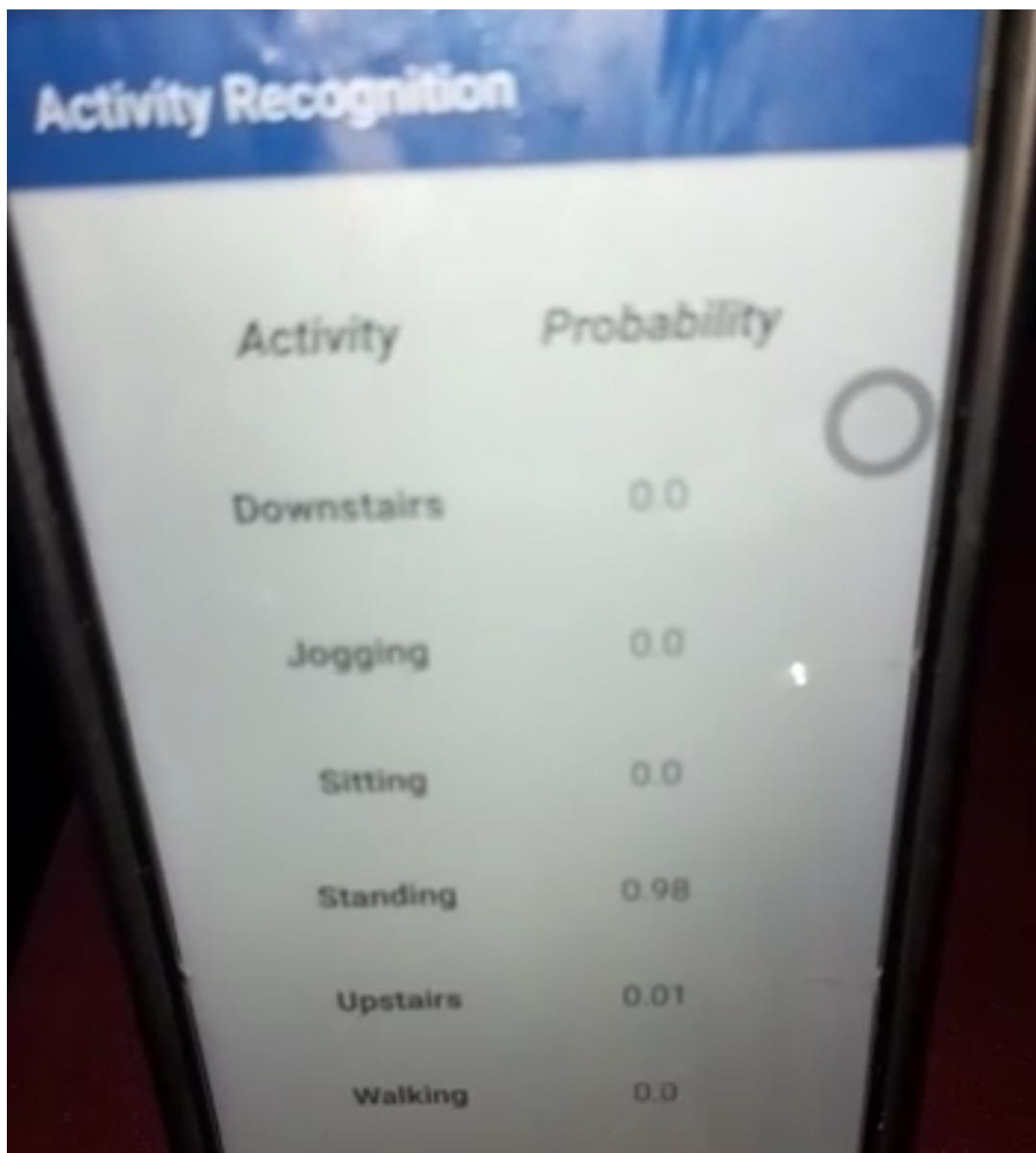


Figure 5.36 : Live Application Part 1

A photograph of a smartphone screen displaying an 'Activity Recognition' application. The screen shows a list of activities and their corresponding probabilities. The activities listed are Downstairs, Jogging, Sitting, Standing, Upstairs, and Walking. The probabilities are 0.0 for Downstairs, Jogging, Sitting, and Walking; 0.98 for Standing; and 0.01 for Upstairs. A circular icon is visible on the right side of the screen.

Activity	Probability
Downstairs	0.0
Jogging	0.0
Sitting	0.0
Standing	0.98
Upstairs	0.01
Walking	0.0

Figure 5.37 : Live Application Part 2



## 6 Used Technologies

PyCharm (Community edition 2020.1)

Android Studio

Python 3.6

Tensorflow 1.15.2

## 7 Possible Areas for Future Improvements

This project was developed only to identify what a human is performing at a particular time. This can be further developed to identify more activities. Also if the user had an accident (fall) such as falling down the stairs , enhanced versions of these projects might be able to recognize those situations as well. So that the required help can be sent by the relevant parties to help that person.

## 8 References

- 1) <http://www.cis.fordham.edu/wisdm/includes/files/sensorKDD-2010.pdf> - Project based research paper
- 2) <http://www.cis.fordham.edu/wisdm/dataset.php> - Dataset
- 3) <https://github.com/aqibsaheed/Human-Activity-Recognition-using-CNN/tree/master/ActivityRecognition> - Android application feature extraction
- 4) <https://stackoverflow.com/questions/34343259/is-there-an-example-on-how-to-generate-protobuf-files-holding-trained-tensorflow> - Tensorflow trained model freezing
- 5) <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>