IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Human Activity Recognition Using Smartphone

[1]Saniya, [2]Falguni Rathiya, [3]Vaishnavi Wakde

Department of Computer Engineering, Dhole Patil College of Engineering, Wagholi, Pune, India

[4]Prof. Chaitanya Mankar, Assistant Professor

Department of Computer Engineering, Dhole Patil College of Engineering, Wagholi, Pune, India

Abstract: Nowadays Activity detection is becoming an important part of our daily lives, as more people are becoming health conscious. With the help of activity detection, some diseases can be found at an early stage and treated on a prior basis. Therefore, the software of these applications should be very light so that it can run on maximum smartphones. In this paper, we will be building a machine learning model that will be capable of streaming motion data and recognizing the activities. We are applying the LSTM algorithm to a dataset containing accelerometer and gyroscope sensor values from a smartphone, thereafter training a classifier to recognize activities that can also be deployed in a smartphone. In this paper, we develop a lightweight algorithm for activity detection based on Long Short-Term Memory networks, which is being used for time series classification for handling continuous data with an accuracy of 98%.

Index Terms **– Accelerometer, Gyroscope, Human Activity Recognition, LSTM, Machine Learning, Smartphone.**

I. INTRODUCTION

Human activity recognition has a broad area for research and development. It can be achieved by smartphones, wearables, or other devices. Currently, smartphones come with many embedded sensors such as accelerometers, gyroscopes, light sensors, proximity sensors, etc. The values from the dataset for the accelerometer and gyroscope can be used to detect the activity being performed by the user. We are currently detecting movements done by the user such as walking, walking upstairs, walking downstairs, sitting, standing, and jogging, etc. with an accuracy of 98%. The accelerometer and gyroscope values for different movements display a specific pattern. We have trained an LSTM Time-series classification algorithm to recognize these patterns. The primary objective of this project is to develop an efficient human action recognition system using multiple views whereas the secondary objective is to understand the human behavioral model using probabilistic action graphs. It can be very beneficial to keep track of the basic activities as it improves well-being.

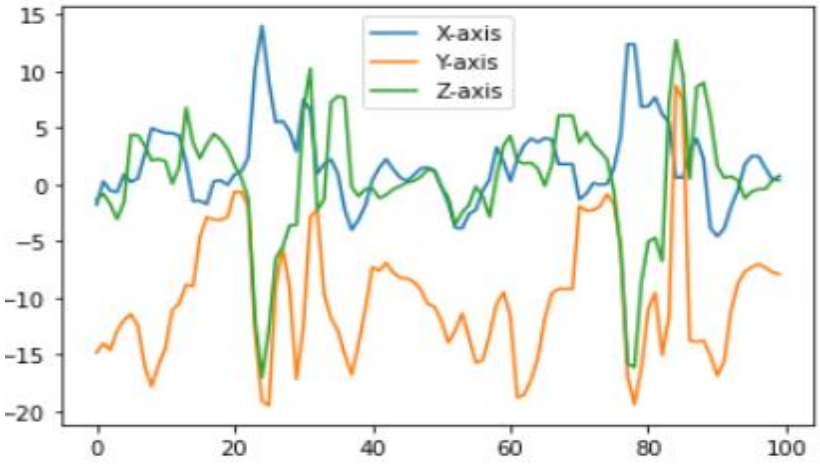
II. MODEL TRAINING

A. Raw Data Analysis

This study considers six activities, namely jogging, sitting, standing, walking, upstairs, and downstairs. These activities are available in raw data, provided by the University of Twente, and are ideal to use as a base for our research. We are using only the left pocket and right pocket data to analyze human actions. Magnetometer readings are being discarded as most smartphones don't have magnetometers. Only the accelerometer and gyroscope readings are being considered. Most of these activities also involve repetitive motion making it easier to train and recognize. The activities have been recorded using a tri-axial accelerometer. The x-axis represents the horizontal motion, which is towards the left and right sides of the phone. This is used to capture the horizontal motion of the leg. The y-axis detects the vertical motion which is the direction to the top and bottom of the phone, while the z-axis represents the motion into and out of the screen, which can be used to capture the leg's backward and forward motion.

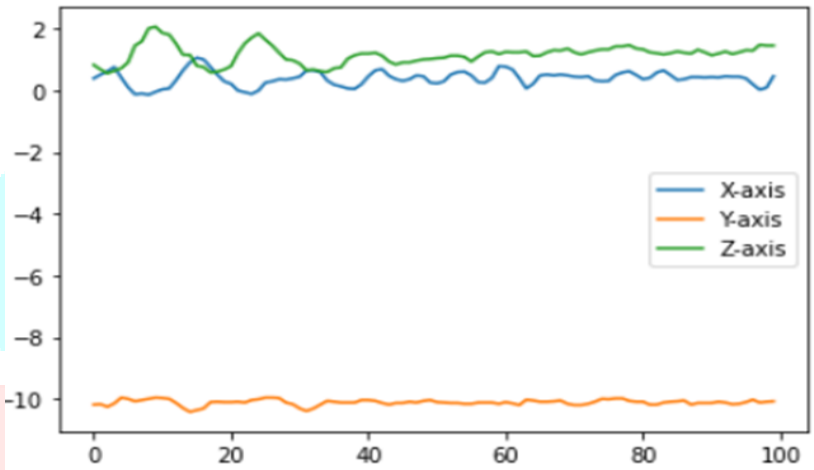
Various activities and observations are being visualized in the graph for the x, y, and z-axis.

1. Walking



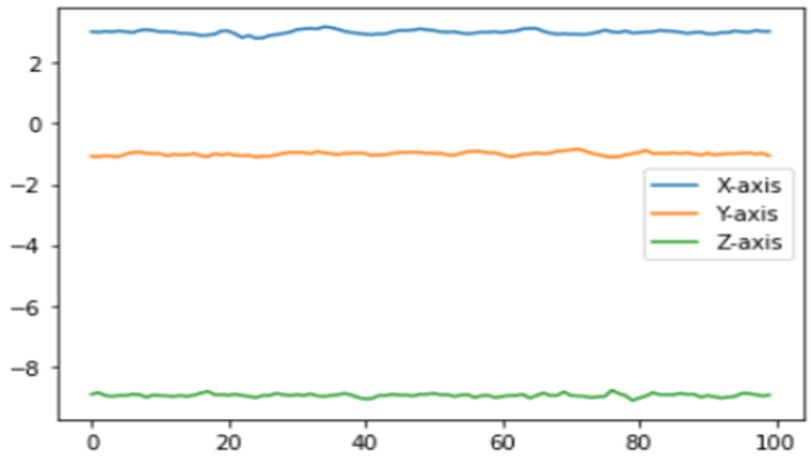
We can see from the graph that the user is moving with regular rhythms for walking activity.
We can see the patterns along the axes and there are a lot of fluctuations along all three axes and the y-axis is touching around 20.

2. Standing



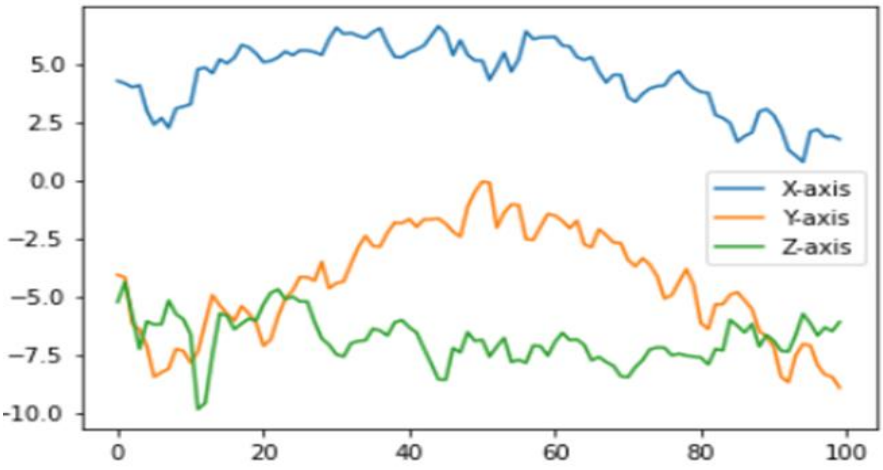
The actions depict that the motion is almost constant having low values i.e. the user is standing still which is being shown with a straight line.

3. Sitting



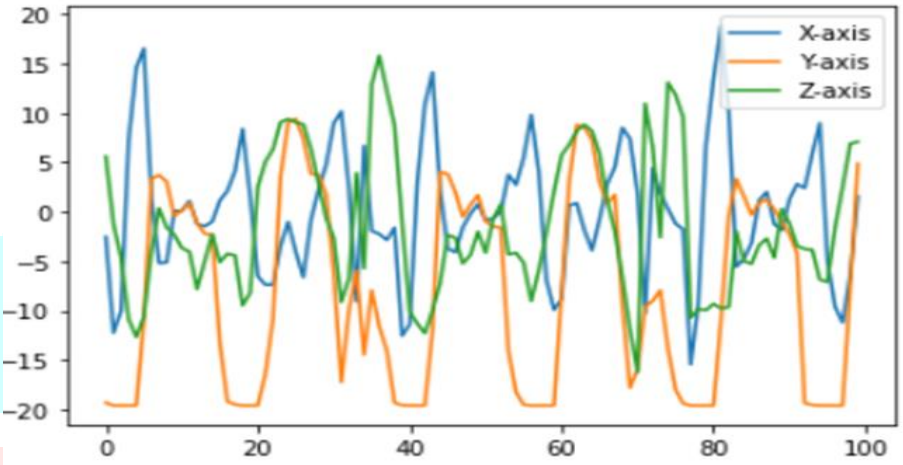
The values of X and Z did not touch 3, and y is between -2 and 0 because of gravity. And these low values are the reason for the stationary position.

4. Biking



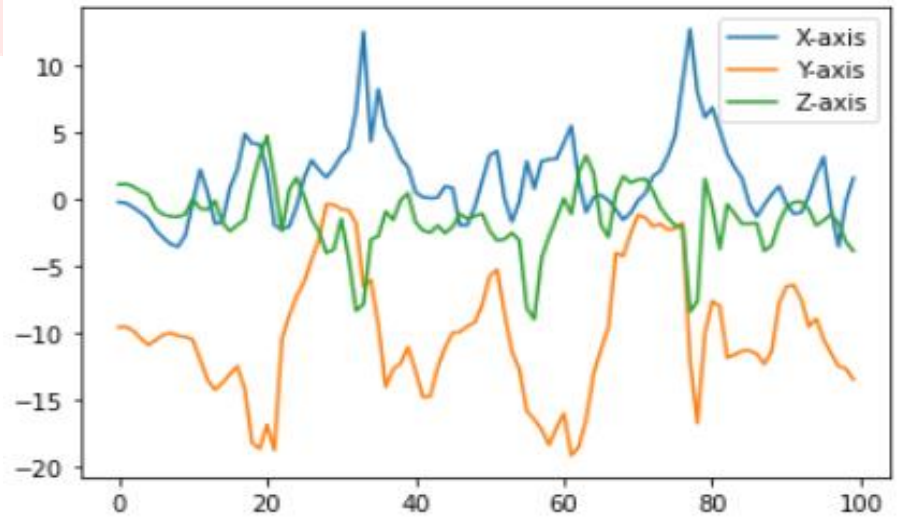
Continuous irregular fluctuations are being shown along all three axes in biking activity.

5. Jogging

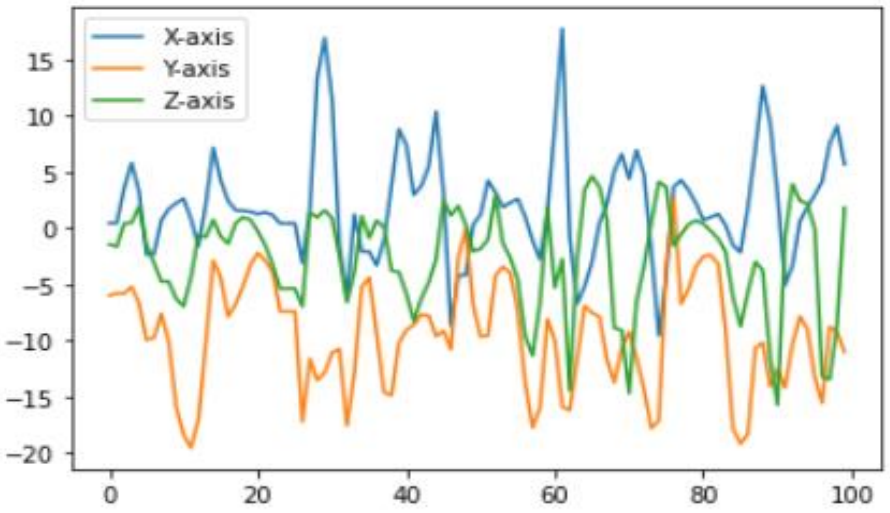


Jogging and walking are looking a bit similar in patterns but the fluctuations or peaks are denser than in walking.

6. Upstairs



7. Downstairs



In upstairs and downstairs figures, you can find the sudden and wide fluctuations or peaks mostly found along the Z and Y axes.

B. Feature generation and data transformation

In order to have many training batches at hand, it is necessary to split each class into a set number of data points and to store this into segments. We have considered the first 100 data points for training and a batch size of 1024. We define a window size of 100 and a number of features are 9 respectively. The segments can then be fed into our network for training, where the network will learn itself, what features belong to what class, based on the labels provided with the training data. When the real-time data is fed into the trained network, it will then be able to make predictions.

The following method uses a windowing method to extract 100 data points from the dataset to create segments and labels for our model training.

Convert dataset into time series sequence

```
In [27]: from keras.preprocessing.sequence import TimeseriesGenerator

         n_time_steps = 100
         n_features = 9

         train_gen = TimeseriesGenerator(X_train.to_numpy(), y_train, length=n_time_steps, batch_size=1024)
         test_gen = TimeseriesGenerator(X_test.to_numpy(), y_test, length=n_time_steps, batch_size=1024)
```

C. Data Processing

From our dataset, it is necessary to split the data into training, testing, and validation sets for modeling purposes. A competing concern is that, with a smaller training set, the parameter estimates have more variance. However, with a smaller test set, the performance statistics will have greater variance. It is important that neither of the variances is too high. The reason for having validation data is to help evaluate the quality of the model, to avoid over-and under-fitting, and to assist in selecting the model that would perform the best on unseen data.

The steps required for data separation are, therefore:

- split data into 80:20, training: testing, for instance
- split the remaining training data into 80:20, training: validation
- Sub-sample a couple of random selections of the training data and train the model with this and record the performance on the validation set.
- Perform the sequence with different percentages of training and testing data to get the optimal value

Split training data into training and validation data

In [26]:
```
def train_test_split(X, y, split_size=0.8):
    split= int(len(X) * split_size)
    train_x = X[:split]
    train_y = y[:split]
    test_x = X[split:]
    test_y = y[split: ]
    return train_x, test_x, train_y, test_y

X_train,X_test,y_train, y_test =train_test_split(X_train, y_train)

print("X_train shape ", X_train.shape)
print("Y train shape ", y_train.shape)
print("X_test shape ", X_test.shape)
print("y_test shape ", y_test.shape)
```

```
X_train shape  (1008000, 9)
Y train shape  (1008000,)
X_test shape  (252000, 9)
y_test shape  (252000,)
```

D. Model Building and Training

We make a sequential model having the first LSTM layer containing 32 LSTM units and we will return sequences as it is having output by taking two-dimensional input which is a group of 100 values from every input column. By using the regularization function we prevent the layer from overfitting and assign a name to the layer. Flatten layer will convert the two-dimensional output from the LSTM layer to one-dimensional input to the next layer. The next layer is a feed-forward layer which contains rectified linear units with regularization function and the name assigned to it. The dense layer is the next layer that will act as a classifier for the model. The last layer contains servants of the max unit which uses will throw out probability depending upon the input it gets from the previous layer.

Create and compile LSTM model

In [28]:
```
from keras.models import Sequential
from keras.layers import Dense,Flatten, LSTM
from keras.regularizers import l2
from keras.optimizers import Adam
```

In [29]:
```
model=Sequential()
model.add(LSTM(32, return_sequences=True, input_shape = (n_time_steps, n_features),
               kernel_regularizer = l2(0.000001), bias_regularizer = l2(0.000001), name='lstm_1'))
model.add(Flatten(name='flatten'))
model.add(Dense(64, activation='relu',kernel_regularizer = l2(0.000001), bias_regularizer = l2(0.000001), name='dense_1' ))
model.add(Dense(len(np.unique(y_train)), activation='softmax',
                kernel_regularizer = l2(0.000001), bias_regularizer = l2(0.000001), name='output'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 100, 32)           5376
_________________________________________________________________
flatten (Flatten)            (None, 3200)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                204864
_________________________________________________________________
output (Dense)               (None, 7)                 455
=================================================================
Total params: 210,695
Trainable params: 210,695
Non-trainable params: 0
_________________________________________________________________
```

We will train the model for 5 epochs. The model gave us an accuracy of 98%.

Start training

In [32]:
```
history = model.fit_generator(train_gen, epochs=5, validation_data=test_gen, callbacks=callbacks)
```

```
Epoch 2/5
985/985 [==============================] - ETA: 0s - loss: 0.0794 - accuracy: 0.9827
Epoch 00002: val_loss improved from 0.07809 to 0.06497, saving model to model.h5
985/985 [==============================] - 219s 223ms/step - loss: 0.0794 - accuracy: 0.9827 - val_loss: 0.0650 - val_accurac
y: 0.9828
Epoch 3/5
985/985 [==============================] - ETA: 0s - loss: 0.0603 - accuracy: 0.9834
Epoch 00003: val_loss improved from 0.06497 to 0.05151, saving model to model.h5
985/985 [==============================] - 177s 180ms/step - loss: 0.0603 - accuracy: 0.9834 - val_loss: 0.0515 - val_accurac
y: 0.9885
Epoch 4/5
985/985 [==============================] - ETA: 0s - loss: 0.0506 - accuracy: 0.9875
Epoch 00004: val_loss did not improve from 0.05151
985/985 [==============================] - 170s 172ms/step - loss: 0.0506 - accuracy: 0.9875 - val_loss: 0.0848 - val_accurac
y: 0.9777
Epoch 5/5
985/985 [==============================] - ETA: 0s - loss: 0.0724 - accuracy: 0.9873
Epoch 00005: val_loss improved from 0.05151 to 0.03549, saving model to model.h5
985/985 [==============================] - 168s 171ms/step - loss: 0.0724 - accuracy: 0.9873 - val_loss: 0.0355 - val_accurac
y: 0.9905
```

E. Saving the model

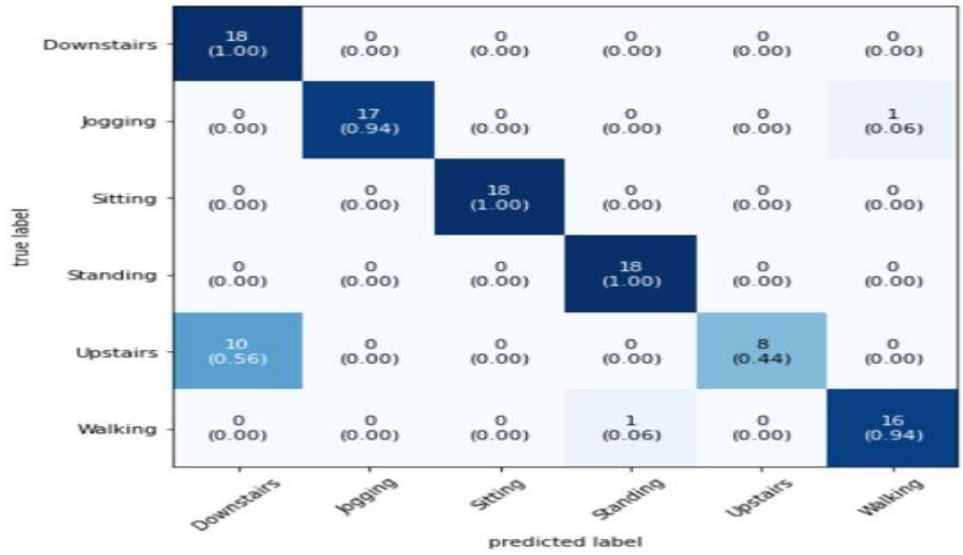
We define a model checkpoint in Keras which is a call back to save the best performance model during training. We save the model in the model.h5 file in the same directory.

```
In [31]: # prepare callbacks
         from keras.callbacks import ModelCheckpoint

         callbacks= [ModelCheckpoint('model.h5', save_weights_only=False, save_best_only=True, verbose=1)]
```

III. RESULTS

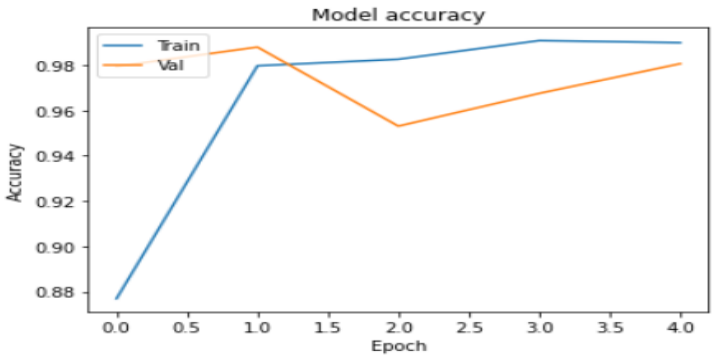
The model gave an accuracy of 98%, however, the model got confused between upstairs and downstairs. Biking activity was discarded as it is not relatable to the other activities like sitting, standing, etc. Still, the model surprisingly gave an accuracy of 98% and all the other activities are being predicted with a probability of approximately 1.

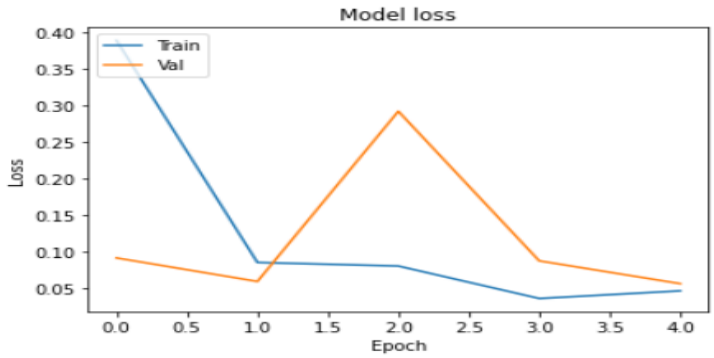


A learning curve will tell how the training set has performed against the test set.

We can see from the graph that our model is somewhat overfitted and can be corrected by feeding more data to the model.

Apart from that model accuracy is high and it remains to be at its peak.





IV. CONCLUSION

In this paper, a human activity recognition design has been proposed using an LSTM-RNN deep neural architecture. The network is able to learn all six classes successfully and efficiently with only a few hundred epochs, reaching high accuracies. The work can be further extended with different data sources as well as different classes. The model has been successfully exported and loaded to an Android app together with the TensorFlow library, giving the ability to perform real-time predictions. L2 regularization has already been implemented to prevent overfitting, but another solution would be to add dropout, which is one of the most effective and commonly used regularization techniques for neural networks to prevent overfitting models.

REFERENCES

[1] Human Activity Recognition Using LSTM-RNN Deep Neural Network Architecture Schalk Wilhelm Pienaar1, Reza Malekian1,2, Senior Member, IEEE, 1Department of Electrical, Electronic and Computer Engineering, University of Pretoria, Pretoria,0002, South Africa

[2] https://www.utwente.nl/en/eemcs/ps/dataset-folder/sensors-activity-recognition-dataset-shoaib.rar