CSYE 7245 Big Data Systems and Intelligence Analytics
Spring 2018

HUMAN ACTIVITY RECOGNITION

Progress Draft

Harshad Pardeshi

Krutika Deshpande

Prithvi Kamath

Under the guidance of

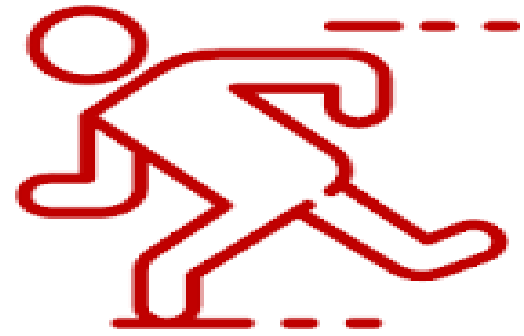
Prof. Nick Brown

1. Abstract

Human activity recognition plays a significant role in human-to-human interaction and interpersonal relations. To enhance the study in the field of human interaction we have classified the human activity performed using Deep Learning. As '**Deep Learning'** is the idea of this decade, we have implemented one of the first advances in deep learning for object detection i.e. Long Short-Term Memory network(**LSTM**) a special kind of Recurrent Neural Network(RNN), capable of learning long-term dependencies.

The research is performed on data collected from accelerometer and gyroscope sensors built from the recordings of 30 participants performing activities of daily living while carrying a waist-mounted smartphone with embedded inertial sensors. The objective is to classify activities into one of the six activities performed viz **WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING.**

Our goal for this project was to showcase that standard machine learning algorithms cannot predict sequential classification problems better whereas the same can be predicted accurately with deep neural networks. With the given data, we started with analyzing the data using exploratory data analysis, later applied machine learning algorithms like SVC, Decision Tree to classify the activities. As the data contains sequential values implemented RNN using Keras and then developed an Android App to track the activity performed for which we developed an LSTM model in TensorFlow.



2. Introduction

The human ability to recognize another person's activities is one of the main subjects of study of the scientific areas of computer vision and machine learning. Because of this research, many applications, including video surveillance systems, human-computer interaction, and robotics for human behavior characterization, require a multiple activity recognition system.

With applications in various fields, we collected dataset where 30 volunteers within an age bracket of 19-48 years were asked to perform six activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone (Samsung Galaxy S II) on the waist. Using its embedded accelerometer and gyroscope, 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz was captured. The experiments have been video-recorded to label the data manually. [1]

The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window). The sensor acceleration signal, which has gravitational and body motion components, was separated using a Butterworth low-pass filter into body acceleration and gravity. The gravitational force is assumed to have only low frequency components, therefore a filter with 0.3 Hz cutoff frequency was used. From each window, a vector of features was obtained by calculating variables from the time and frequency domain. [1]

In 2001, Paul Viola and Michael Jones invented an efficient algorithm for face detection. Their demo that showed faces being detected in real time on a webcam feed was the most stunning demonstration of computer vision and its potential at the time [6]. Soon, it was implemented in OpenCV and face detection became synonymous with Viola and Jones algorithm.

Every decade or so a new idea comes along that is so effective and powerful that you abandon everything and wholeheartedly embrace it. Deep Learning is that idea of this decade. Deep learning models have crushed other classical models on the task of image classification and they are now state of the art in object detection as well. One of the first advances in using deep learning for object detection was OverFeat from NYU published in 2013. They proposed a multi-scale sliding window algorithm LSTM Neural Network.

We want to apply our learnings about these advances in algorithms for human activity recognition as currently it is a fundamental problem in perception that is receiving increasing attention.

3. Code with Documentation

Starting with understanding the dataset, we understood the dataset contains 7352 observations for body acceleration, body gyroscope and total body acceleration in all the three axes.

To find the distribution in the dataset, we performed exploratory data analysis which gave us the frequency of activities and users on the training dataset.

Figure 1: Frequency of Training examples by Activity Type

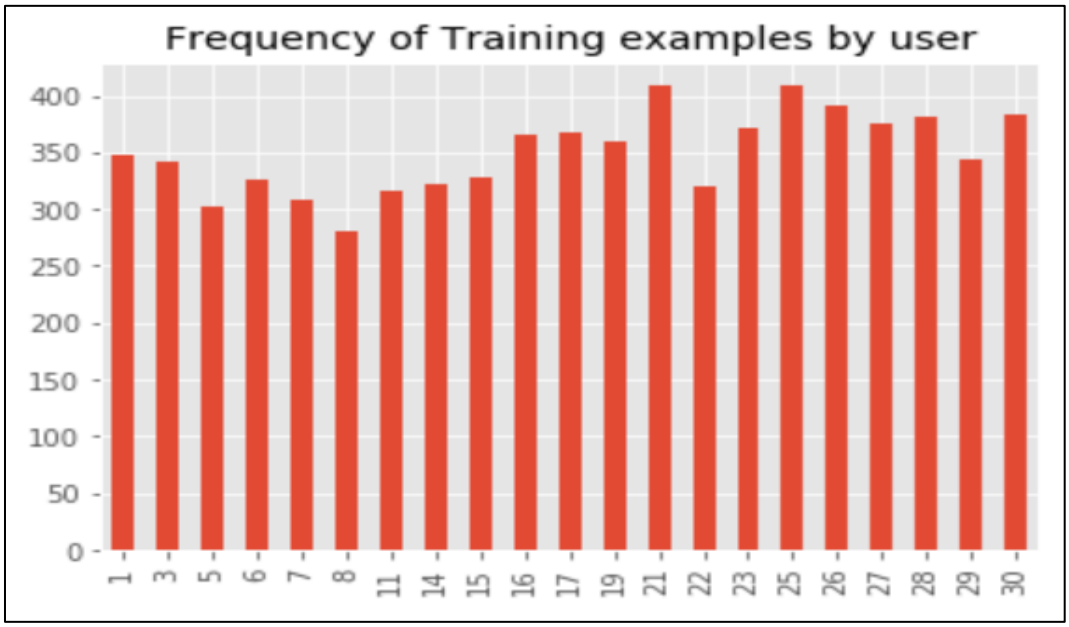


Figure 2: Frequency of Training examples by user

With exploratory data analysis done, we moved on to implementing machine learning models on the dataset. As the dataset is a 3-D matrix, we converted it to a (7352 X 1152) dataframe to implement these models. Following is the accuracy of algorithms implemented:

Algorithm	Accuracy
Decision Tree	71.97 %
K Nearest Neighbors	61.89 %
SVC	76.95 %
Gaussian Naïve Bayes	72.48 %
Quadratic Discriminant Analysis	48.45 %

Table 1: Algorithm and Accuracy

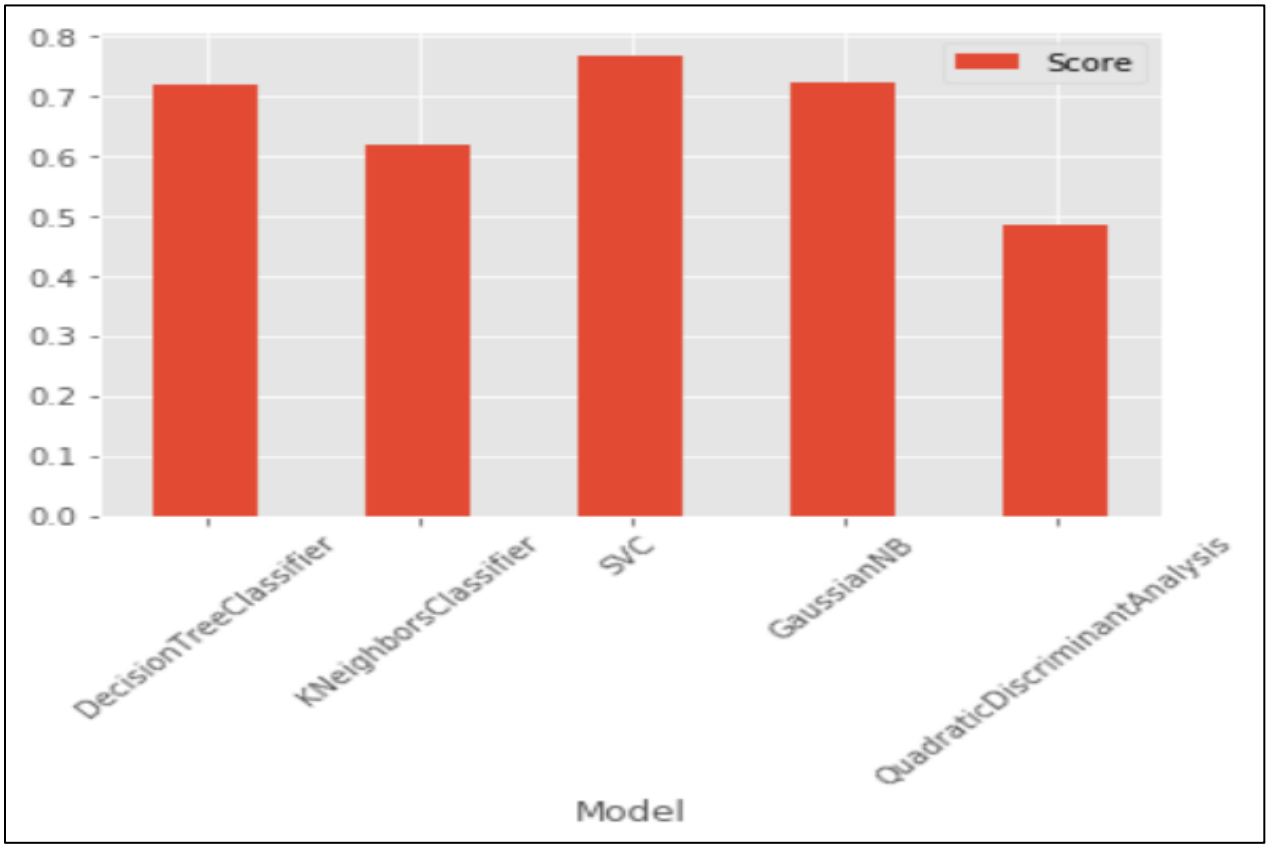


Figure 3: Plot Algorithm vs Algorithm Accuracy

As expected there were a lot of prediction errors while implementing these algorithms, hence we decided to implement recurrent neural networks using Keras to improve the accuracy of the model.

To add a Recurrent layer, we make use of a Gated Recurrent Unit (GRU) layer with 32 units. It has reset gate applied to hidden state before matrix multiplication. [2]

```
In [16]: from keras.models import Sequential
         from keras import layers
         from keras.optimizers import RMSprop

         model = Sequential()
         model.add(layers.GRU(32, input_shape=(None, X_train.shape[-1])))
         model.add(layers.Dense(1))

         model.compile(optimizer='adam', loss='mae', metrics=['accuracy'])
```

We tried using different optimizers such as SDG, RMSprop and Adam. Adam turned out be give best results for our data. After this we train our model for 50 epochs. [2]

```
In [17]: history = model.fit(x=X_train,
                             y=y_train,
                             batch_size=200,
                             epochs=20,
                             verbose=1,
                             callbacks=None,
                             #validation_split=0.2,
                             validation_data = (X_test, y_test),
                             #validation_data=None,
                             shuffle=True,
                             #class_weight=None, sample_weight=None,initial_epoch=0, steps_per_epoch=None, validation_steps=None
                            )
```

We tried splitting training data into training and validation sets. This decreased our accuracy as we had less data to train. Rather we use test data to validate our model. This gives us training accuracy of 91.68% and a validation accuracy of 81.81%

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer [3].

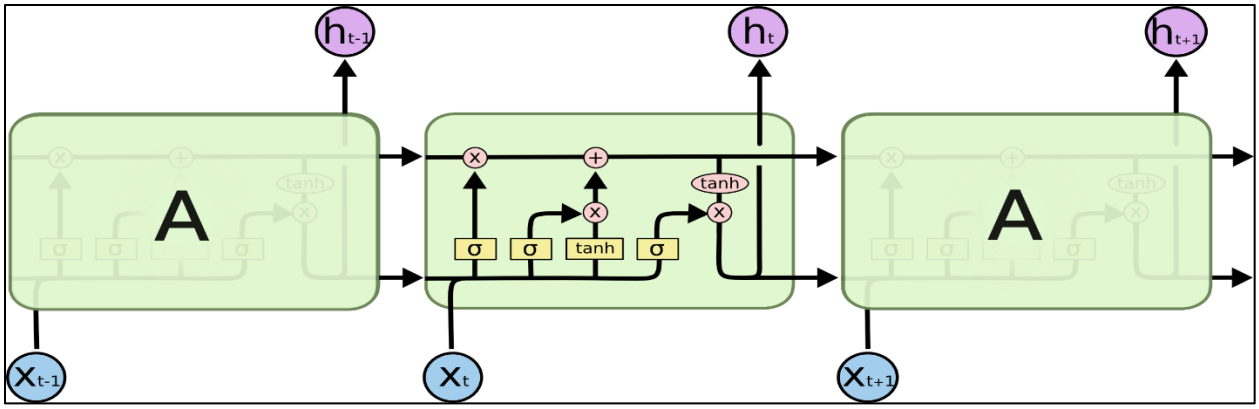


Figure 4: LSTM Network

As our objective was to predict the activities using LSTM, we implemented LSTM using tensorflow with the following parameters [8]

- **number of LSTM Cells**: 2
- **learning_rate**: 0.0025
- **lambda_loss_amount**: 0.0015
- **epochs**: 300
- **activation function**: TanH
- **optimizer**: AdamOptimizer
- **cost function**: Softmax Loss

In [23]:
```
def LSTM_RNN(_X, _weights, _biases):
    #input shape: (batch_size, n_steps, n_input)
    _X = tf.transpose(_X, [1, 0, 2])  # permute n_steps and batch_size
    #Reshape to prepare input to hidden activation
    _X = tf.reshape(_X, [-1, n_input])
    #new shape: (n_steps*batch_size, n_input)

    #Linear activation
    _X = tf.nn.tanh(tf.matmul(_X, _weights['hidden']) + _biases['hidden'])
    #Split data because rnn cell needs a list of inputs for the RNN inner loop
    _X = tf.split(_X, n_steps, 0)
    #new shape: n_steps * (batch_size, n_hidden)

    #Define two stacked LSTM cells (two recurrent layers deep) with tensorflow
    lstm_cell_1 = tf.contrib.rnn.BasicLSTMCell(n_hidden, forget_bias=1.0, state_is_tuple=True)
    lstm_cell_2 = tf.contrib.rnn.BasicLSTMCell(n_hidden, forget_bias=1.0, state_is_tuple=True)
    lstm_cells = tf.contrib.rnn.MultiRNNCell([lstm_cell_1, lstm_cell_2], state_is_tuple=True)
    #Get LSTM cell output
    outputs, states = tf.contrib.rnn.static_rnn(lstm_cells, _X, dtype=tf.float32)

    #Get last time step's output feature for a "many to one" style classifier,
    #as in the image describing RNNs at the top of this page
    lstm_last_output = outputs[-1]

    #Linear activation
    return tf.matmul(lstm_last_output, _weights['out']) + _biases['out']
```

With 32 hidden layers and 6 output classes, the model was trained using LSTM for random normal network initialization.

4. Discussion

With the extracted data, we performed exploratory data analysis analyzing the distribution of the 6 activities amongst 30 users. Initially implemented Decision Tree, K Nearest Neighbors, SVC, Gaussian Naïve Bayes, Quadratic Discriminant Analysis on the dataset. As the dataset contains sequential values we implemented RNN using Keras resulting in 89.51% test accuracy. Furthermore, to study and predict the activity performed more accurately we decided on implementing LSTM using TensorFlow which gave the test accuracy of 90.77%.

After the successful prediction, we thought of developing an Android App. The dataset available contains 9 variables with body acceleration, body gyroscope and total acceleration in all the three axes. The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window). The sensor acceleration signal, which has gravitational and body motion components, was separated using a Butterworth low-pass filter into body acceleration and gravity. The gravitational force is assumed to have only low frequency components, therefore a filter with 0.3 Hz cutoff frequency was used. From each window, a vector of features was obtained by calculating variables from the time and frequency domain.

For the android app we get 6 variables i.e. body gyroscope and total acceleration tri-axially. To deduce the body acceleration from total acceleration, the gravity factor should be deducted from the total acceleration. To get rid off the gravity factor, we tried two approaches viz implementing Butterworth low-pass filter and applying Machine Learning Algorithm.

1. Machine Learning Algorithm

We tried getting the third variable by implementing Machine Learning algorithms (Linear Regression and Random Forest) as well as implementing the Butterworth low pass filter. Linear Regression gave an R square value of 0.22 while Random forest model was overfitting to the training data.

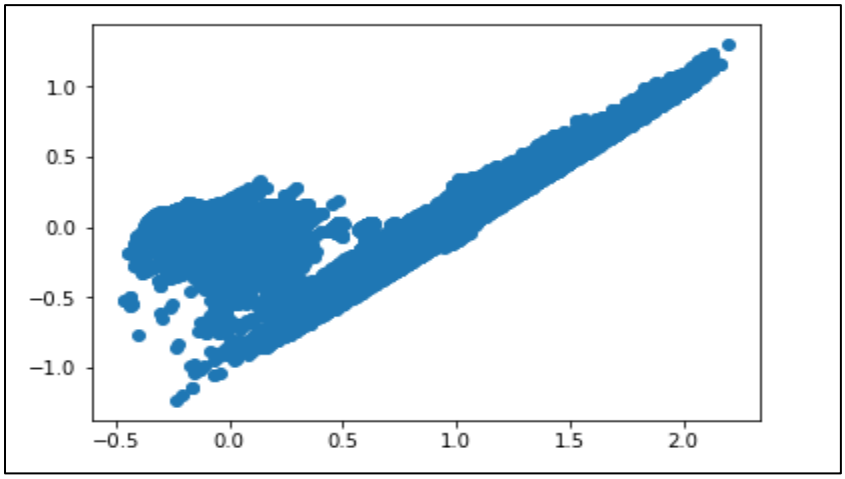


Figure 5: Body Acceleration vs Total Body Acceleration

2. Implementation of Butterworth Low pass filter

Implementation of butterworth low pass filter didn't give values as expected in the training data and hence we finally decided to re-train the model ignoring the 3rd feature and using this newly built model to take run-time values from the smartphone

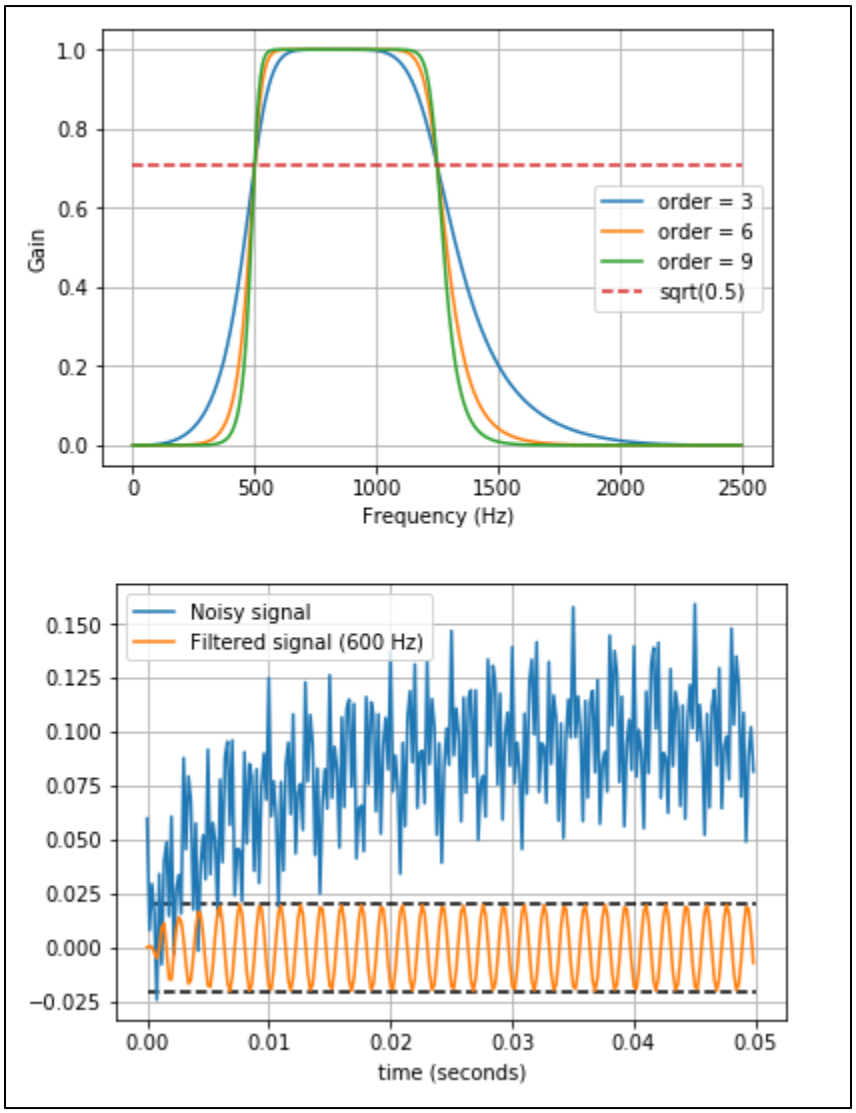


Figure 6: Butterworth Filter

5. Results

Parameters

- **number of LSTM Cells**: 2
- **learning_rate**: 0.0025
- **lambda_loss_amount**: 0.0015
- **epochs**: 300
- **activation function**: tanH
- **optimizer**: AdamOptimizer
- **cost function**: Softmax Loss

resulted in Training accuracy **98.46** % and Test Accuracy **89.97**%

Variation of Loss and Accuracy for the test dataset is shown below. With achieving a stable accuracy, the loss decreases and remains constant.

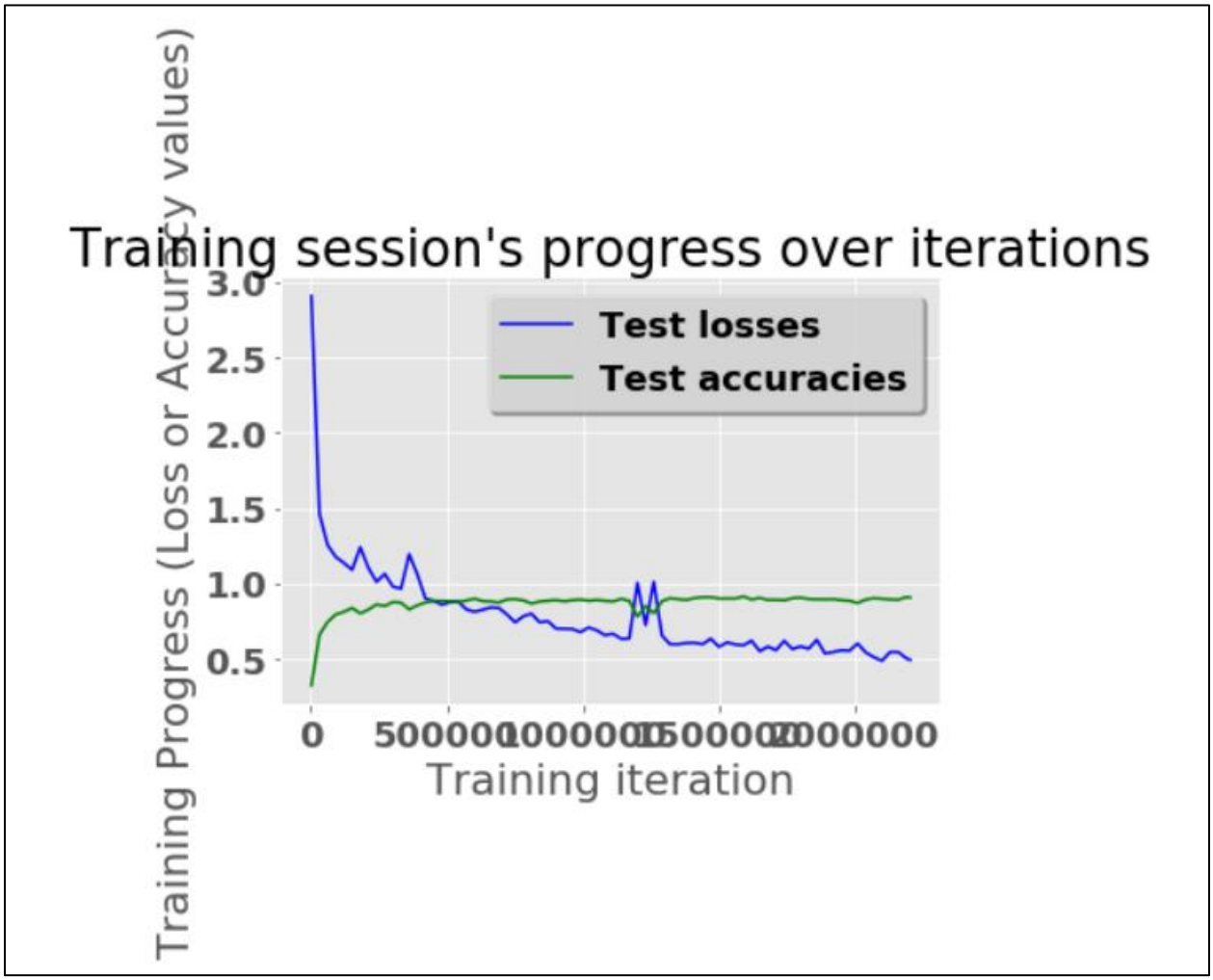


Figure 7:Test Loss vs Accuracy

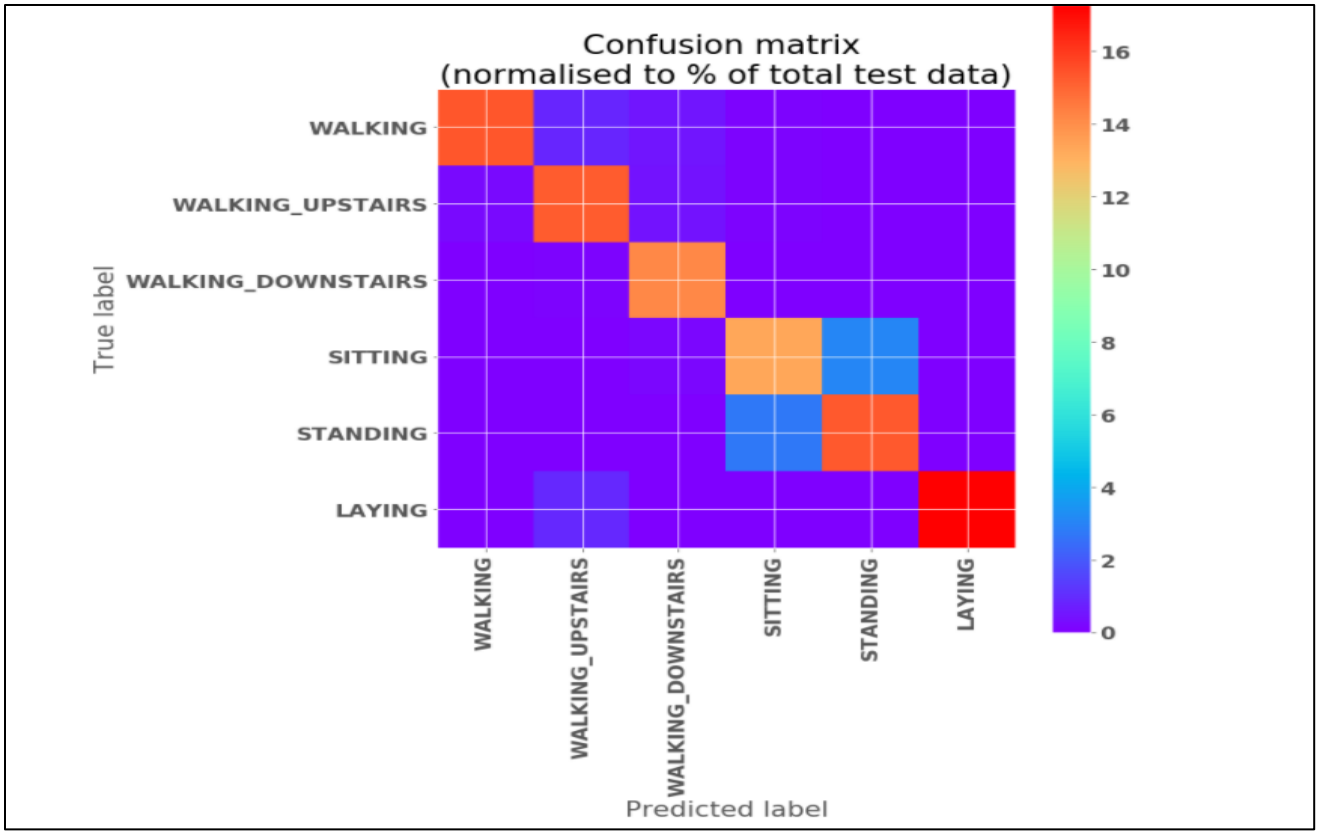


Figure 8: Confusion Matrix

We tweaked the parameters of our LSTM model with different activation functions like Sigmoid, relu, increased the number of epochs, used cost functions like hinge, cosine distance softmax_cross_entropy_with_logits and sigmoid_cross_entropy_with_logits, but so far with the above-mentioned parameters we have got the best accuracy for our test dataset.

Following table shows the performance of the two models implemented based on accuracy and loss:

Deep Learning Model	Training		Validation	
	Accuracy	Loss	Accuracy	Loss
RNN (Keras)	91.68%	0.14	81.81%	0.25
LSTM(TensorFlow)	98.46%	0.24	88.97%	0.63

6. Conclusion

Implementation of standard machine learning algorithms cannot classify predict sequential classification problems better whereas the same can be predicted accurately with deep neural networks. As evident from the above results, LSTM provided better prediction compared to RNN. proved that ANN cannot classify sequential data. With the requirement of 6 variables for the development of Android App using the designed LSTM model, we reduced the number of input variables to the model which affected the accuracy.

7. References

1. https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones
2. https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html
3. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
4. https://towardsdatascience.com/lstm-by-example-using-tensorflow-feb0c1968537
5. https://stackoverflow.com/questions/12093594/how-to-implement-band-pass-butterworth-filter-with-scipy-signal-butter
6. https://link.springer.com/article/10.1023/B:VISI.0000013087.49260.fb
7. https://medium.com/@curiousily/human-activity-recognition-using-lstms-on-android-tensorflow-for-hackers-part-vi-492da5adef64
8. https://github.com/guillaume-chevalier/LSTM-Human-Activity-Recognition
9. https://www.google.com/search?safe=active&biw=1707&bih=815&tbm=isch&sa=1&ei=QEveWvb9K-KiggfFw4vgCg&q=AI+human+activity+wallpaper&oq=AI+human+activity+wallpaper&gs_l=psy-ab.3...123063.125507.0.126493.10.10.0.0.0.0.179.722.9j1.10.0....0...1c.1.64.psy-ab..0.0.0....0.APmQXKU1shg#imgrc=GhFEq8xqeazqfM:
10. https://github.com/aymericdamien/TensorFlow-Examples