A Project Report

on

InnoHAR: A DEEP NEURAL NETWORK FOR HUMAN ACTIVITY RECOGNITION

Submitted in the partial fulfillment of the requirements

for the award of

Bachelor of Technology

in

Electronics & Communication Engineering

By

Rohit Raj Anand (20175139)

Satya Shree Samantroy (20175105)

Amresh Yadav (20155014)

Under the guidance of

Dr. Vinay Kumar

(Assistant Professor)

Department of Electronics & Communication Engineering

Motilal Nehru National Institute of Technology Allahabad, Prayagraj–211004, INDIA

(December 2020)

UNDERTAKING

We declare that the work presented in this project titled **"*InnoHAR: A deep neural network for Human activity recognition*"**, submitted to the DEPARTMENT OF ELECTRONICS & COMMUNICATIONS ENGINEERING, MOTILAL NEHRU NATIONAL INSTITUTE OF TECHNOLOGY ALLAHABAD, PRAYAGRAJ for the award of the **Bachelor of Technology** degree in ***Electronics & Communication Engineering*** is our original work.
We have not plagiarized or submitted the same work for award of any other degree.

Date:
Prayagraj, India

Rohit Raj Anand (20175139)
Satya Shree Samantroy (20175105)
Amresh Yadav (20155014)

Department of Electronics & Communications Engineering
Motilal Nehru National Institute of Technology Allahabad
Prayagraj, India

CERTIFICATE

This is to certify that the work contained in project titled "**InnoHAR: A deep neural network for Human Activity Recognition**" submitted by **Rohit Raj Anand, Satya Shree Samantroy** and **Amresh Yadav** in the partial fulfillment of the requirement for the award of **Bachelor of Technology** in *Electronics & Communication Engineering* to the Electronics & Communication Engineering Department, Motilal Nehru National Institute of Technology, Allahabad, is a bona fide work of the students carried out under my supervision.

Date:
Prayagraj, India

Dr. Vinay Kumar
Assistant Professor
ECE Department
MNNIT Allahabad

Acknowledgements

We take this opportunity to express our genuine thanks and gratitude to our project supervisor, **Dr. Vinay Kumar,** Department of Electronics & Communication Engineering, **Motilal Nehru National Institute of Technology, Allahabad** for his regular guidance and insightful critique and comments during the course of the work. We shall forever be grateful for our association with him and for his constant encouragements and freedom of thought and actions which helped us complete this work.

We are also thankful to our colleagues and friends for their constant support. We get a great pleasure in thanking everybody who helped us complete this work directly or indirectly.

Date:
Prayagraj, India

Rohit Raj Anand (20175139)
Satya Shree Samantroy (20175105)
Amresh Yadav (20155014)

Abstract

In this report we will be implementing a detection system which can detect complex human activities from sensor data. We will be using a deep neural network “InnoHAR” for this problem of complex Human activity recognition.

Nowadays there are multiple wearable devices available to the users which contain many embedded sensors like accelerometers, gyroscopes etc. These devices are used for tracking human activities like sleep, walking, sitting idle etc. The activity tracking can be extended to more complex activities like jogging, exercises etc. The accuracy of these devices depends heavily on the classification models used.

Many models like One-Dimensional CNN, Long Short-term Memory (LSTM) etc. are being used in these fields. We will be training these models as well in order to compare their performance against the InnoHAR architecture.

Contents

List of figures

List of tables

Abbreviations

- **HAR:** Human Activity Recognition
- **CNN:** Convolutional Neural Network
- **LSTM:** Long Short-Term Memory
- **GRU:** Gated Recurrent Unit

Chapter 1

Introduction

1.1 Introduction to Human Activity Recognition using sensors

Human activity recognition is the problem of classifying a sequence of data from sensors like accelerometers, gyroscopes etc. into well-defined movements. These sequences are generally recorded using specialized harness of sensors or by embedded sensors in smartphones. Sensor-based activity recognition integrates the emerging area of sensor networks with novel data mining and machine learning techniques to model a wide range of human activities.

Existing researches often use statistical machine learning methods to manually extract and construct features of different motions from time series data based on sliding windows of fixed sizes and then training machine learning models such as ensembles of decision trees. However, in the face of extremely fast-growing waveform data with no obvious laws, the traditional feature engineering methods are becoming more and more incapable. Additionally, Feature engineering required a lot of expertise in the field.

With the development of deep learning technology, we do not need to manually extract features and can improve the performance in complex human activity recognition problems. So, someone who is not an expert is the field of activity recognition could train a very good model to as it required no feature engineering. These models provided state-of-the-art results on very challenging activity recognition tasks.

1.2 Motivation

Human Activity Recognition (HAR) is the detection, interpretation, and recognition of human behaviours, which can use smart heath care to actively assist users according to their needs. Human activity recognition has wide application prospects, such as monitoring in smart homes, sports, game controls, health care, bad habits detection, and identification.

One of the applications can be Elderly patient care in which we can monitor the behaviours of elderly people to see if they are doing fine.

In Healthcare, patient behaviour can be monitored and different states like seizures and other complications can be detected using sensor data.

In gym activities, HAR can be used to classify different workouts that are being done. These classifications can help users to exercise in a better way and recognise if the workout routines are being done in the correct manner.

As long as we are able to gather enough data, we can train the Human Activity Classifier models to differentiate between various activities according to the field of application without needing expertise in field itself.

1.3 Goal

In this report we will be training the InnoHAR model for Human activity recognition and then we will compare its performance with various other neural-nets like 1D-CNN, LSTM, CNN-LSTM & ConvLSTM.

Chapter 2

Deep Neural-Nets used in HAR

There are many deep neural networks available for the problem of HAR. Some of the most common ones are as follows:

- One Dimensional Convolutional Neural Networks (CNN)
- Long Short-term Memory (LSTM)
- CNN-LSTM model
- ConvLSTM model

2.1 1D CNN Model

The technique of Convolutional Neural Networks was originally developed for image classification problems where the CNN-model learns the internal representation of a two-dimensional input, in a process referred to as feature learning. The process of convolution involved is a mathematical process involving an image and a kernel. The kernel slides along two dimensions of the image data, extracting spatial features from the images, Hence the process is also known as 2D-Convolution.

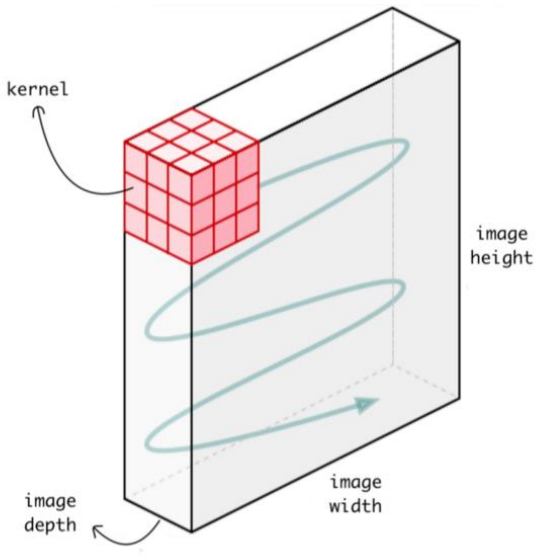

Fig 2.1: Kernel sliding in two different dimensions for image data.

The same process of convolution can be harnesses on One dimensional sequence of data such as acceleration and gyroscope data from sensors to classify human activities. The 1D-CNN model learns to extract features from sequence of observations and to map internal features to different activity types.

Here the process of convolution is same as that in 2D-Convolutions, except that the kernel slides along a single dimension instead of two dimensions. The sliding of kernel is in the direction of the time axis.

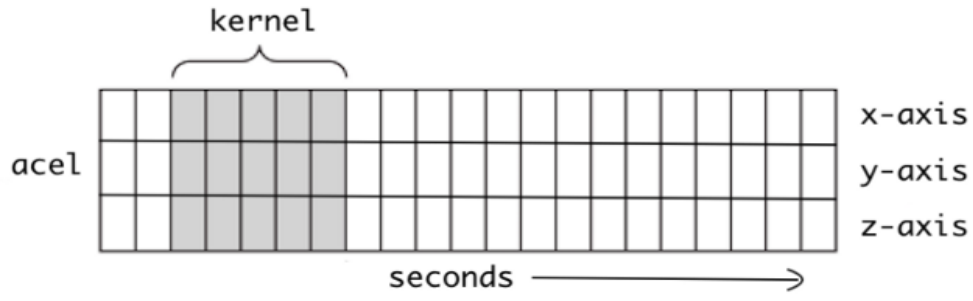

Fig 2.2: Kernel sliding along time axis of time series data (acceleration).

The 1D-CNN Model constitutes of multiple Convolutional layers for extracting features, the extracted features are then flattened and fed to fully connected layers for final classification.

2.2 LSTM Model

LSTMs were designed to counter a major problem in Recurrent Neural Networks. RNNs suffer from vanishing gradient problem. It can forget what it has seen in longer sequences, thus having a short-term memory.

LSTM was created as the solution to short-term memory. They have internal mechanisms called gates that can regulate the flow of information. These gates can learn which data in a sequence is important to keep or throw away.

By doing that, it can pass relevant information down the long chain of sequences to make predictions.

An LSTM has a similar control flow as a recurrent neural network. It processes data passing on information as it propagates forward. The differences are the operations within the LSTM's cells.

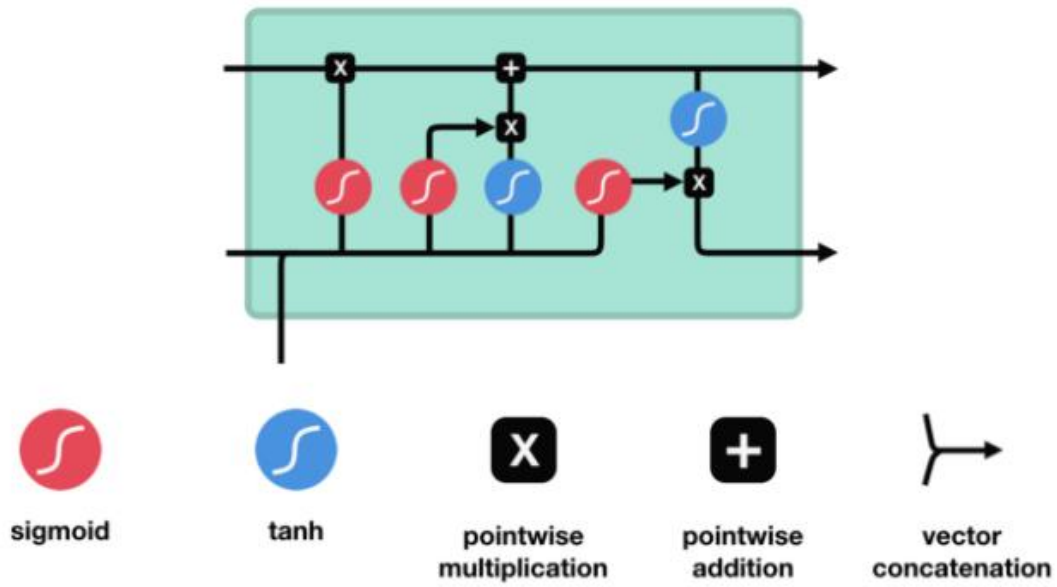

Fig 2.3: Single LSTM cell

A single LSTM cell has three gates: Forget gate, Input gate & Output gate. The Forget gate decides what is relevant to keep from prior steps. The input gate decides what information is relevant to add from the current step. The output gate determines what the next hidden state should be.

A LSTM model generally has a LSTM layer followed by fully connected layers for classification.

2.3 CNN-LSTM Model

The CNN-LSTM model architecture involves using Convolutional Neural Network (CNN) layers to extract features from input data followed by using LSTM to support sequence prediction.

CNN-LSTM were developed for visual time series prediction problems like Activity recognition, Image description, Video description etc.

The CNN-LSTM model can easily be applied to our problem of Human Activity Recognition as well. Because the problem has both spatial and temporal features. The CNN LSTM model will read subsequences of the main sequence in as blocks, extract features from each block, then allow the LSTM to interpret the features extracted from each block.

2.4 ConvLSTM Model

Convolutional LSTM or ConvLSTM is also used for spatio-temporal data. It is a further extension of the CNN-LSTM idea.

The LSTM model reads the data in directly in order to calculate internal states and state transition. The CNN-LSTM model interprets the outputs from CNN models. ConvLSTM is using convolutions directly as part of reading input into the LSTM units themselves.

Keras Library provides a ConvLSTM layer which implements the principle of this model. We will be using this layer for classification without going into much details about the internal implementations.

Chapter 3

InnoHAR: A deep neural network for complex HAR

3.1 Inspiration – GoogLeNet Model

Google's Inception Architecture based network was an important milestone in the development of CNN classifiers. Prior to its inception, most popular CNNs just stacked convolution layers of different kernel sizes deeper and deeper, hoping to get better performance.

Using the Inception Architecture (defined int next section), Google created a deep Neural Network – GoogLeNet, which had far better performance than any CNNs available for Image Classification. This model was faster to train and had better performance than any available CNN architecture. This model was also able to go deeper than any other Network and thus can better extract features and classify Images.

The InnoHAR model was inspired from the GoogLeNet Model Architecture. Like GoogLeNet, it tries to use Inception like architecture to achieve better performance in HAR problems.

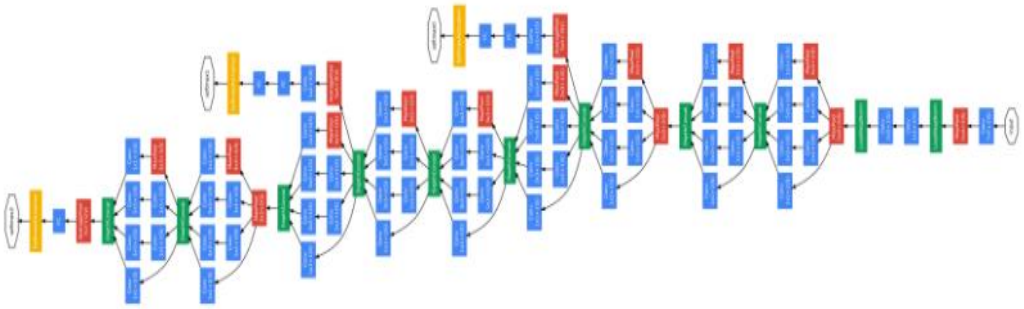

Fig 3.1: GoogLeNet model based on Inception Architectures

3.2 The Inception Architecture

Inception module takes several convolutional kernels of different sizes and stack their outputs along the depth dimension in order to capture features at different scales. The idea was to take the ensemble of different sizes of convolution layers at each stage instead of choosing a size.

Generally, 1x1, 3x3, 5x5 kernel sizes were used in CNNs. The suggested architecture is a combination of all those layers with their output filter banks concatenated into a single output vector forming the input of the next stage. Additionally, since pooling operations have been essential for the success of current convolutional networks, it was suggested that adding an alternative parallel pooling path in each such stage should have additional beneficial effect, too. These ideas lead to the formation of the Inception Architecture.

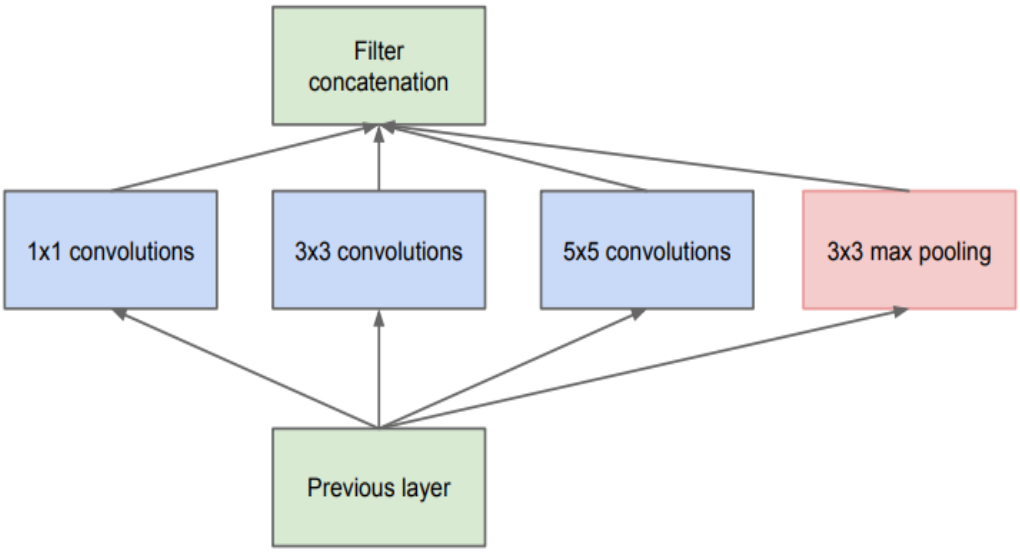

Fig 3.2: Inception Model (Naïve form)

One big problem with the above module, at least in this naive form, is that even a modest number of 5×5 convolutions can be prohibitively expensive on top of a

convolutional layer with a large number of filters. This problem becomes even more pronounced once pooling units are added to the mix: As for each pooling layer, the number of output filters must be same as the number of filters in previous stage. Because of this, the number of parameters to be trained becomes very-very large.

Since these Inception modules are supposed to be stacked together, the total number of parameters to train would increase to a horrific value after a few stages, leading to a computational blow-up. Thus, the above Inception architecture would work but it won't be very efficient.

This leads to the second idea of the Inception architecture: judiciously reducing dimension wherever the computational requirements would increase too much otherwise. For this purpose, 1x1 convolution layers are utilized. 1x1 convolutions are used to compute reductions before the expensive 3x3 and 5x5 convolutions. Besides being used as reductions, they also include the use of rectified linear activation making them dual-purpose. The same reduction is also applied to the max-pooling layer, resulting in the modified Inception Architecture shown below.

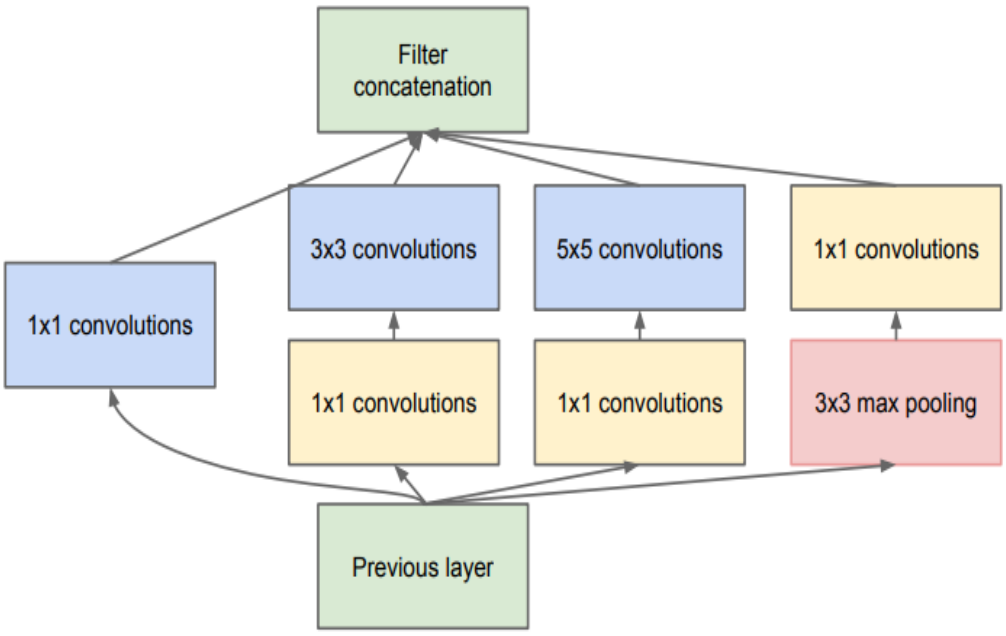

Fig 3.3: Inception module with dimensionality reduction.

This modified Inception module was used for GoogLeNet and the results were very exceptional.

3.3 Modified Inception Architecture for InnoHAR

The Human Activity recognition problem requires the model to be able to extract one - dimensional time series data features instead of 2-D feature as was the case in image classification. As seen earlier in section 2.1, a One-dimensional convolution layer can be utilized for this purpose as opposed to the 2-D convolutional layer for image data. In a two-dimensional convolution the kernel slides along two dimensions (e.g. width and height of an image) while in one-dimensional convolution the kernel slides along one dimension only (e.g. along time axis in time-series data).

This was the main idea behind the modification of Inception Architecture for time series data classification. The supposed 2-D convolutional layers from the original Inception architecture were replaced by 1-D convolutional layer. Hence, the 2-dimensional 3x3 convolutional layer was replaced by 1-dimensional 1x3 convolutional layer. The 2-D 5x5 convolutional layer was replaced by the 1-D 1x5 convolutional layer.

The same was true for all 2-D 1x1 convolution layer also. This layer in which the kernel slides along two different dimensions was replaced by a one-dimensional 1x1 convolutional layer where kernel slides along the time dimension only. The same was true for the max-pooling layer also.

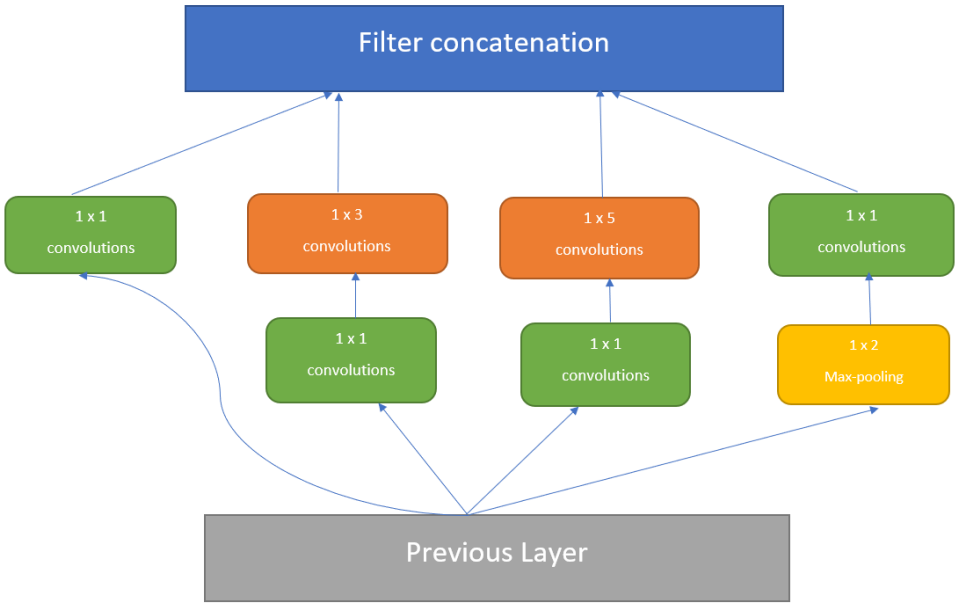

Fig 3.4: Inception-like module for InnoHAR.

The 2-D max-pooling operation happening in the original Inception module was replaced with a one-dimensional max-pooling operation. The result was an Inception Like model which can be used with time series data, making it suitable for the problem of HAR.

3.4 The InnoHAR Model

The Inception-like module from section 3.3, are the building block of the InnoHAR model. Different layers are connected as follows in order to create the aforementioned model:

- The Input layer is passed through three Inception-like modules.
- After that we connect it with max-pooling layer to help the network better eliminate misjudgment caused by noise disturbance.
- Then we pass it through another Inception module and a max pooling layer.
- Finally, the output is passed through two GRU layers, so that the model can better extract the sequential temporal dependencies.
- Then we'll pass the output of GRU Layers through the SoftMax layer for classification.

This model basically works in two different parts: Spatial feature extraction and temporal feature extraction.

For Spatial feature extraction, the Inception-like module from previous section is used. In each Inception-like module, a 1x1 convolution kernel is used to directly activate the combination of multi-channel information and pass it to the next layer. Two convolution kernels of 1x3 and 1x5 are cascaded respectively by a 1x1 convolution kernel, and the feature information of different scales is extracted for the whole model. The output splicing with only 1x1 convolution, also produces a ResNet residual connection effect. At the same time, there is a 1x3 pooling layer followed by a 1x1 convolution kernel to provide feature enhancement and filtering.

For temporal feature extraction, either LSTM (Long Short-term Memory) or GRU (Gated Recurrent Unit) could have been used. Since GRU provides better time efficiency, GRU layers are used in this model. Two GRU layers are used to extract temporal data. Finally, a Dense layer with soft-max activation was added at last in order to classify the extracted features.

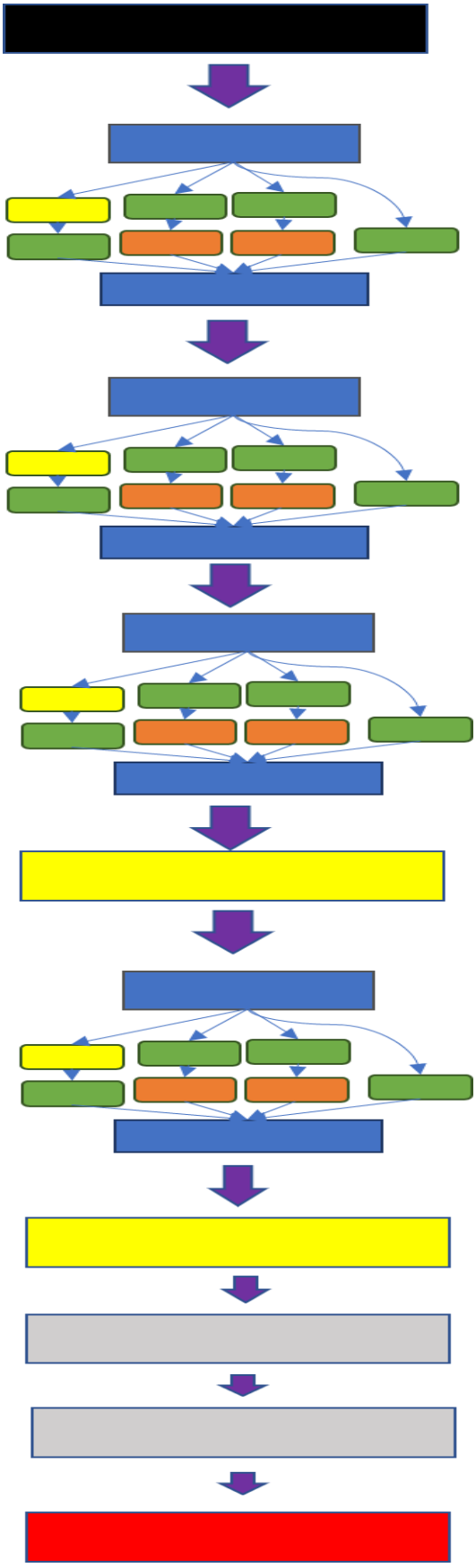

Fig 3.5: InnoHAR model Layer by Layer.

The above diagram depicts the InnoHAR model as proposed. The Black box at the top implies the Input layer. Then we can see the Inception-Like modules stacked together. The Yellow block is supposed to be a Max-Pooling layer to perform one-dimensional max pooling for eliminating the misjudgements caused by noise disturbances. The grey coloured blocks are GRU layers used. Finally, the red block is a dense layer with softmax activation function for classification.

Within the Inception blocks: green signifies 1x1 convolutions, orange is for 1x3 and 1x5 convolutions and yellow blocks are max-pooling layers.

Chapter 4

Experimental setup & Result Analysis

4.1. Experimental Setup

4.1.1. Dataset: Understanding and visualization

We will be using UCI's Human Activity Recognition Using Smartphones Dataset. For creating this dataset an experiment was carried out by Smartlab - Non-Linear Complex Systems Laboratory and CETpD - Technical Research Centre for Dependency Care and Autonomous Living.

The experiments have been carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed 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 were captured at a constant rate of 50Hz. The experiments were video recorded to label the data manually. The obtained dataset was then randomly partitioned into two sets, where 70% of the volunteers was selected for generating the training data and 30% the test data.

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.

After the separation of body acceleration from total acceleration, the sensor data now had 9 different features:

- Body Acceleration (x, y and z axis)
- Angular Velocity (x, y and z axis)
- Total Acceleration (x, y and z axis)

These 9 channel features will be directly input into our model.

This data is available as multiple files from which it was loaded into python-NumPy arrays. After that features were plotted for each of the six activity classes in order to visualize the data in a better way.

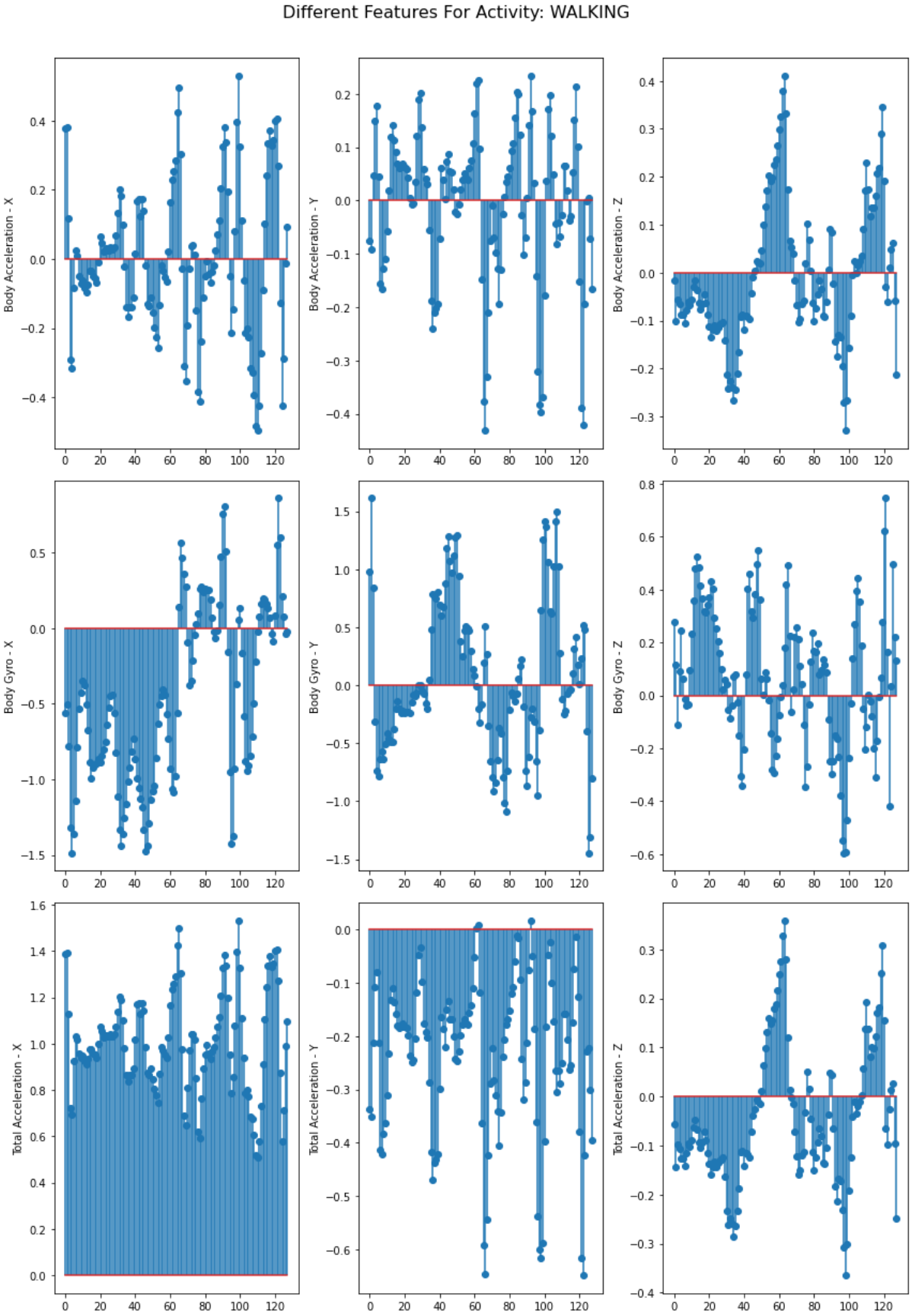

Fig 4.1: Feature plot for Activity: WALKING

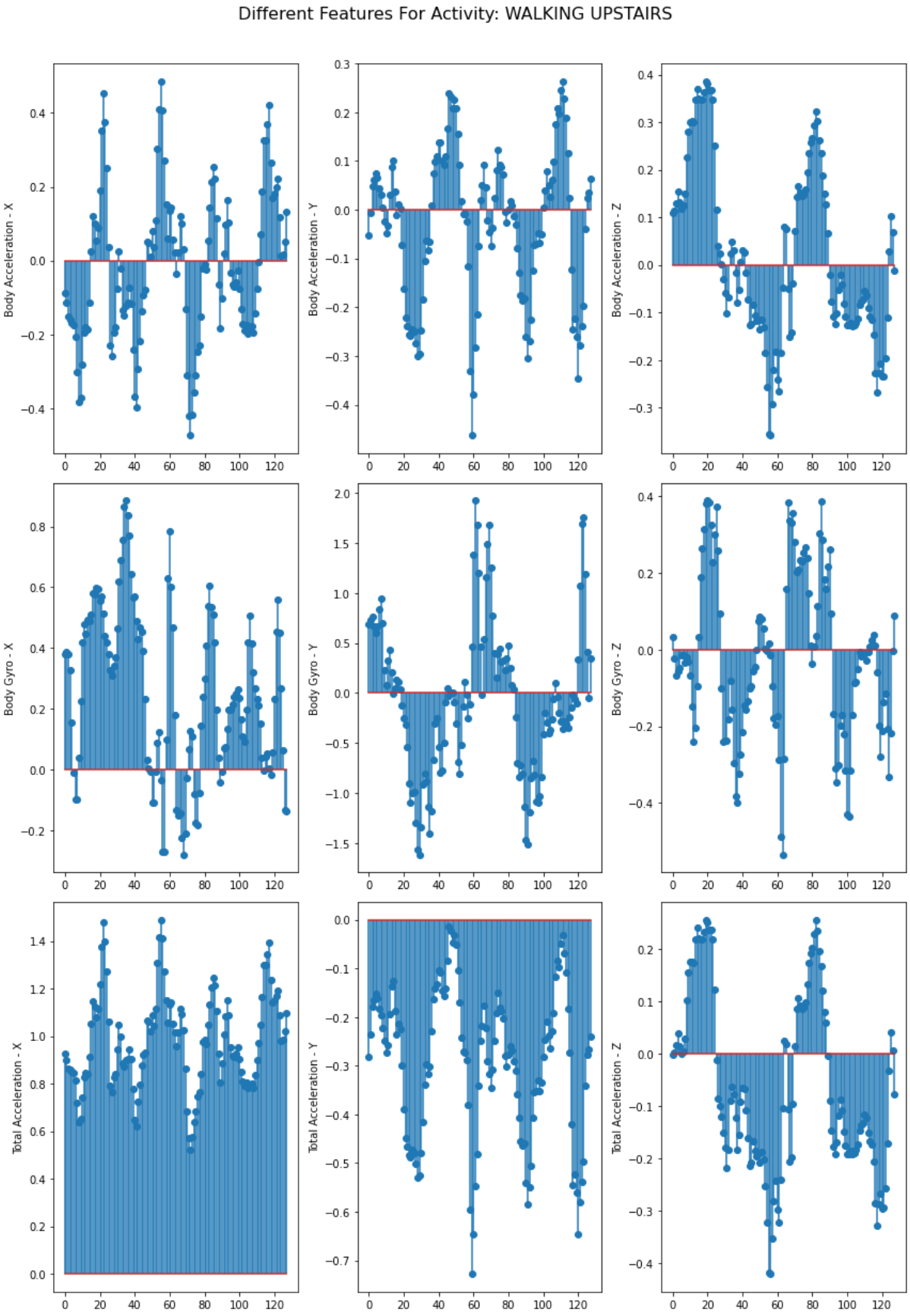

Fig 4.2: Feature plot for Activity: WALKING UPSTAIRS

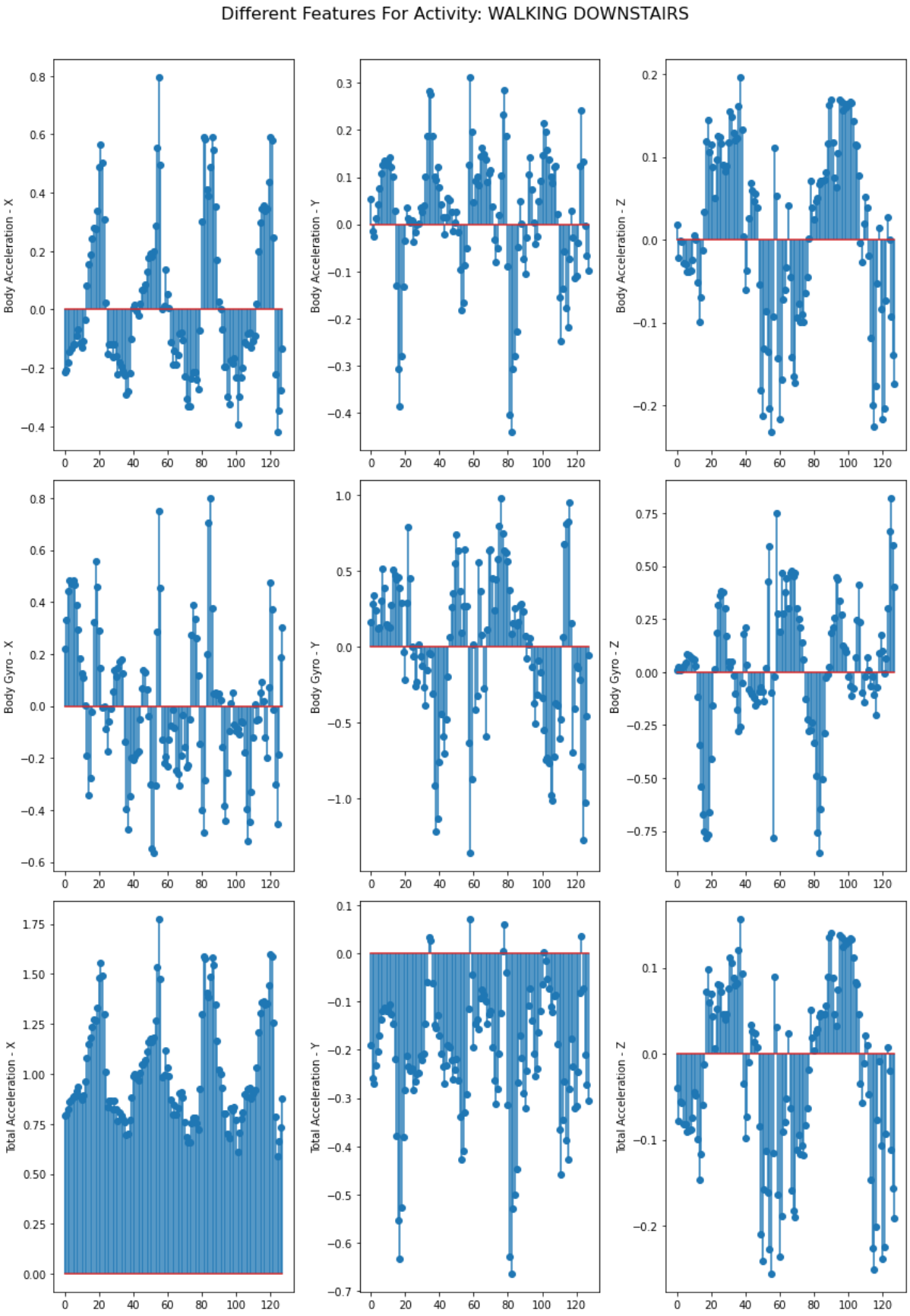

Fig 4.3: Feature plot for Activity: WALKING DOWNSTAIRS

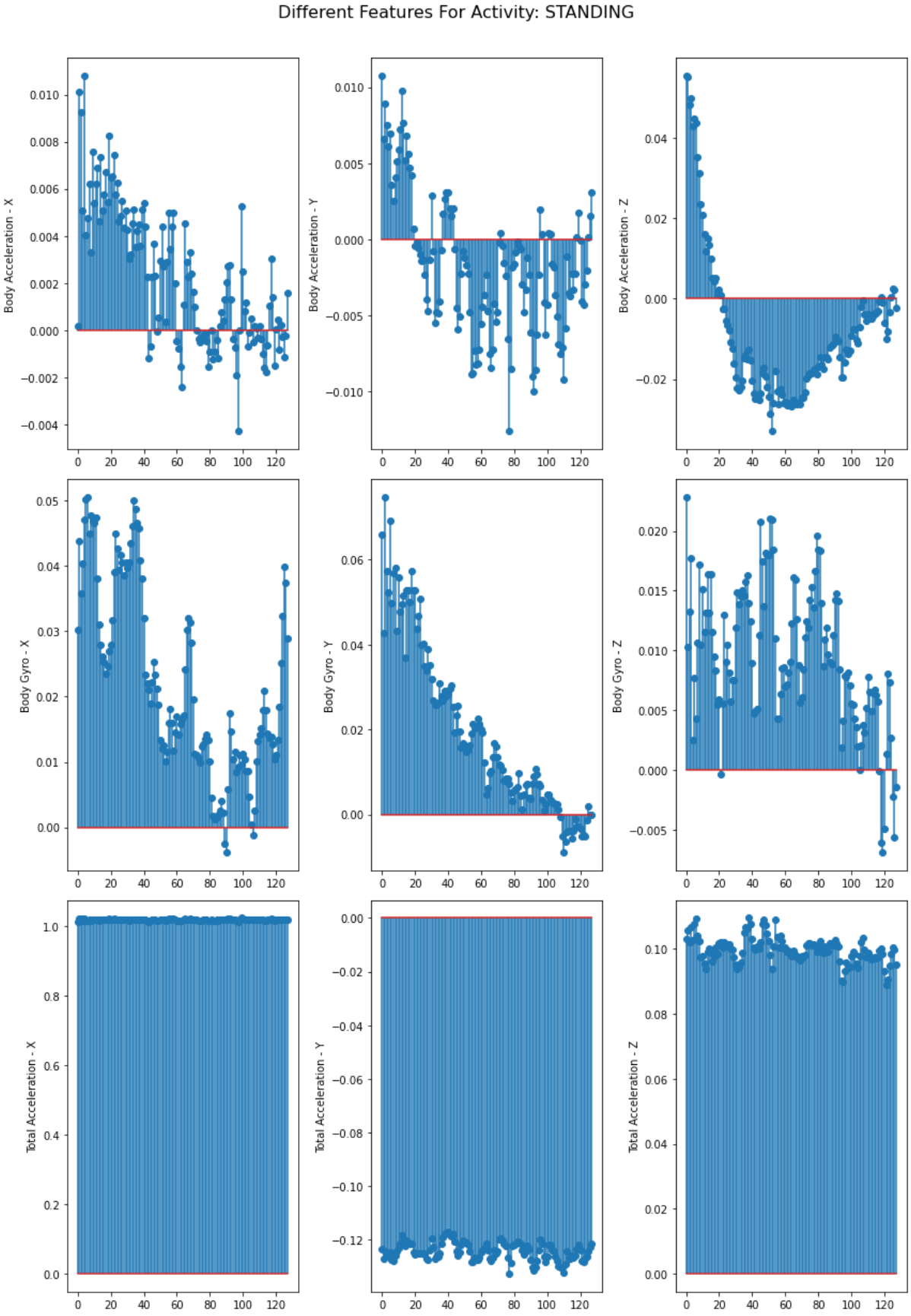

Fig 4.4: Feature plot for Activity: STANDING

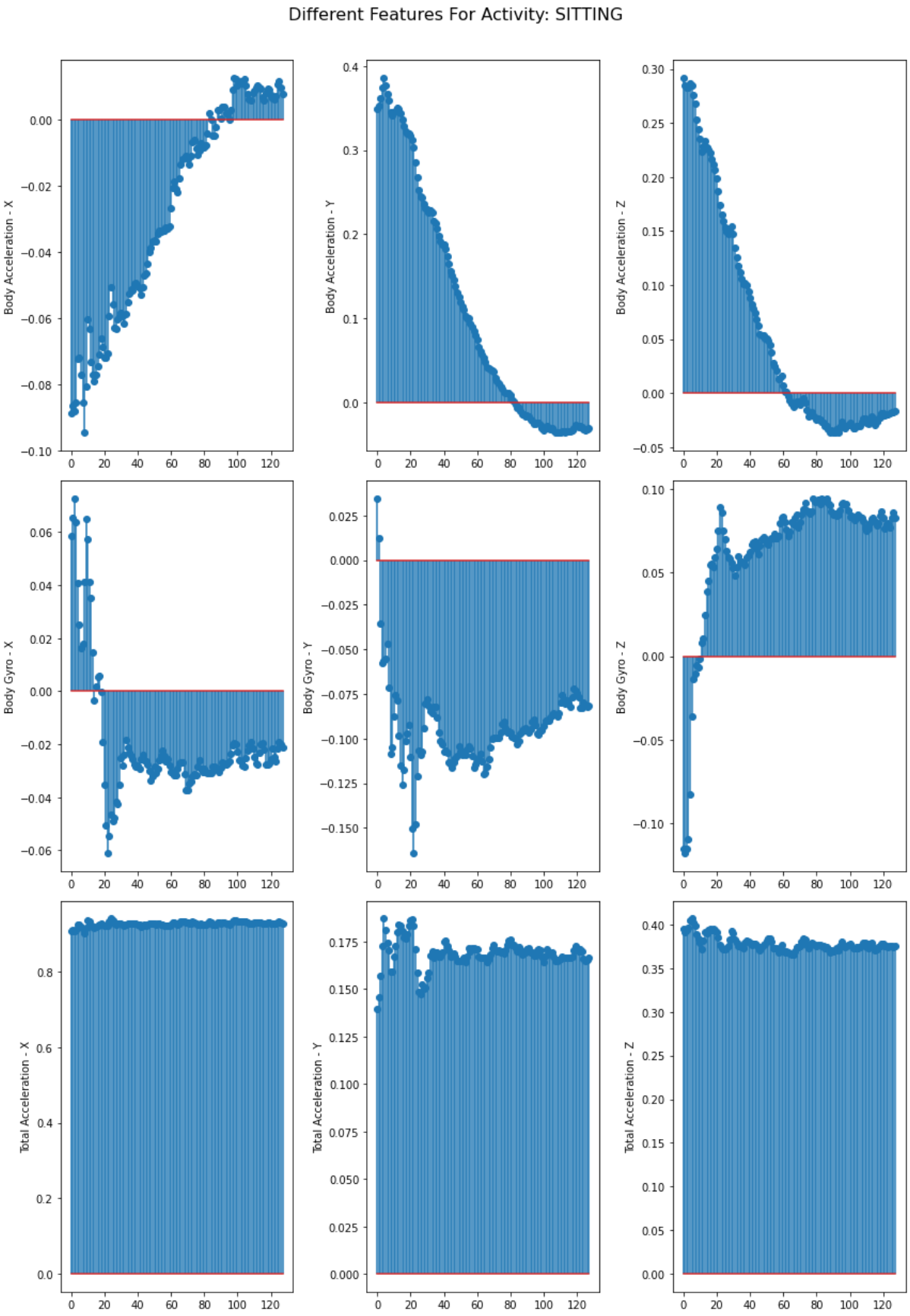

Fig 4.5: Feature plot for Activity: SITTING

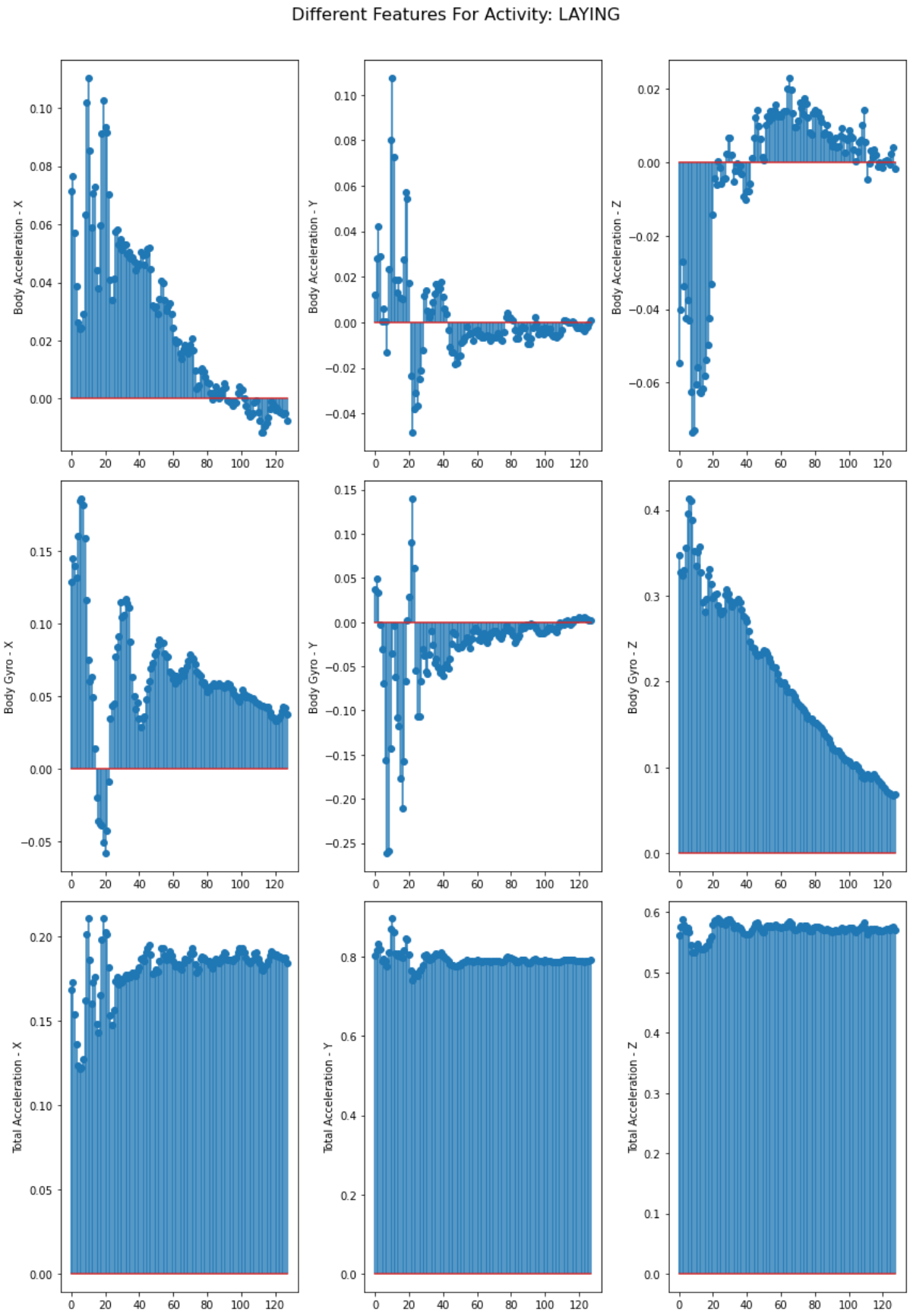

Fig 4.6: Feature plot for Activity: LAYING

4.1.2. Language and Environment Used

Python is a general-purpose programming language started by Guido van Rossum, which became very popular in short time mainly because of its simplicity and code readability. It enables the programmer to express his ideas in fewer lines of code without reducing any readability. Compared to other languages like C/C++, Python is slower. But another important feature of Python is that it can be easily extended with C/C++. This feature helps us to write computationally intensive codes in C/C++ and create a Python wrapper for it so that we can use these wrappers as Python modules. This gives us two advantages: first, our code is as fast as original C/C++ code (since it is the actual C++ code working in background) and second, it is very easy to code in Python.

As Model training requires a lot of computational power and often a GPU (Graphics Processing Units) which was not available to us, we decide to use Google's Colaboratory Notebooks as our development environment. Colab is a Python development environment that runs in the browser using Google Cloud. It allows us to connect to powerful hardware in order to run our codes. This really makes the computational heavy job of training a Deep Learning model quite painless. It also allows us to use TPU (Tensor Processing Unit) to train our models. Cloud TPU is the custom-designed machine learning ASIC for processing tensors. It further speeds up the process of model training and testing. Additionally, Colab notebooks allows different members of a team to collaborate on a single project. Hence, this decision to use Colab Notebooks helped us a lot during this work.

4.1.3. Libraries Used

The following libraries were used throughout different stages of this project:

- **TensorFlow:** TensorFlow is a library developed by the Google Brain Team to accelerate machine learning and deep neural network research. It was built to run on multiple CPUs or GPUs and even mobile operating systems, and it has several wrappers in several languages like Python, C++ or Java. It is called TensorFlow because it takes input as a multi-dimensional array, also known as tensors. You can construct a sort

of **flowchart** of operations (called a Graph) that you want to perform on that input. The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as output.

- **Keras:** Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

- **NumPy:** NumPy is a highly optimized open-source python library to perform calculations on array like objects. It allows us to exploit vectorization to perform very fast numerical calculations. NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries. The core of NumPy is well-optimized C code. It allows us to enjoy the flexibility of Python with the speed of compiled code. Several other python libraries like SciPy, matplotlib, pandas etc. are built directly on top of NumPy.

- **Scikit-learn:** Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means etc. and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

- **Matplotlib:** Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. Pyplot is a Matplotlib module which provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python, and the advantage of being free and open source.

4.2. Result Analysis

4.2.1. Model Training

We used Keras to build our models. We built the models layer by layer using appropriate Keras Layers and then trained these models on the training set. After that the predictions were made on the test set accuracy, classification reports, confusion metrices etc. were obtained. Here we are presenting each model along with various obtained plots and metrices so that we can compare them.

4.2.1.1. One-Dimensional CNN model

A. Model plot and Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 126, 64)           1792
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 126, 64)           4160
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 122, 64)           20544
_________________________________________________________________
dropout (Dropout)            (None, 122, 64)           0
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 61, 64)            0
_________________________________________________________________
flatten (Flatten)            (None, 3904)              0
_________________________________________________________________
dense (Dense)                (None, 100)               390500
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 606
=================================================================
Total params: 417,602
Trainable params: 417,602
Non-trainable params: 0
_________________________________________________________________
```

Fig 4.7: 1D CNN Model: Summary

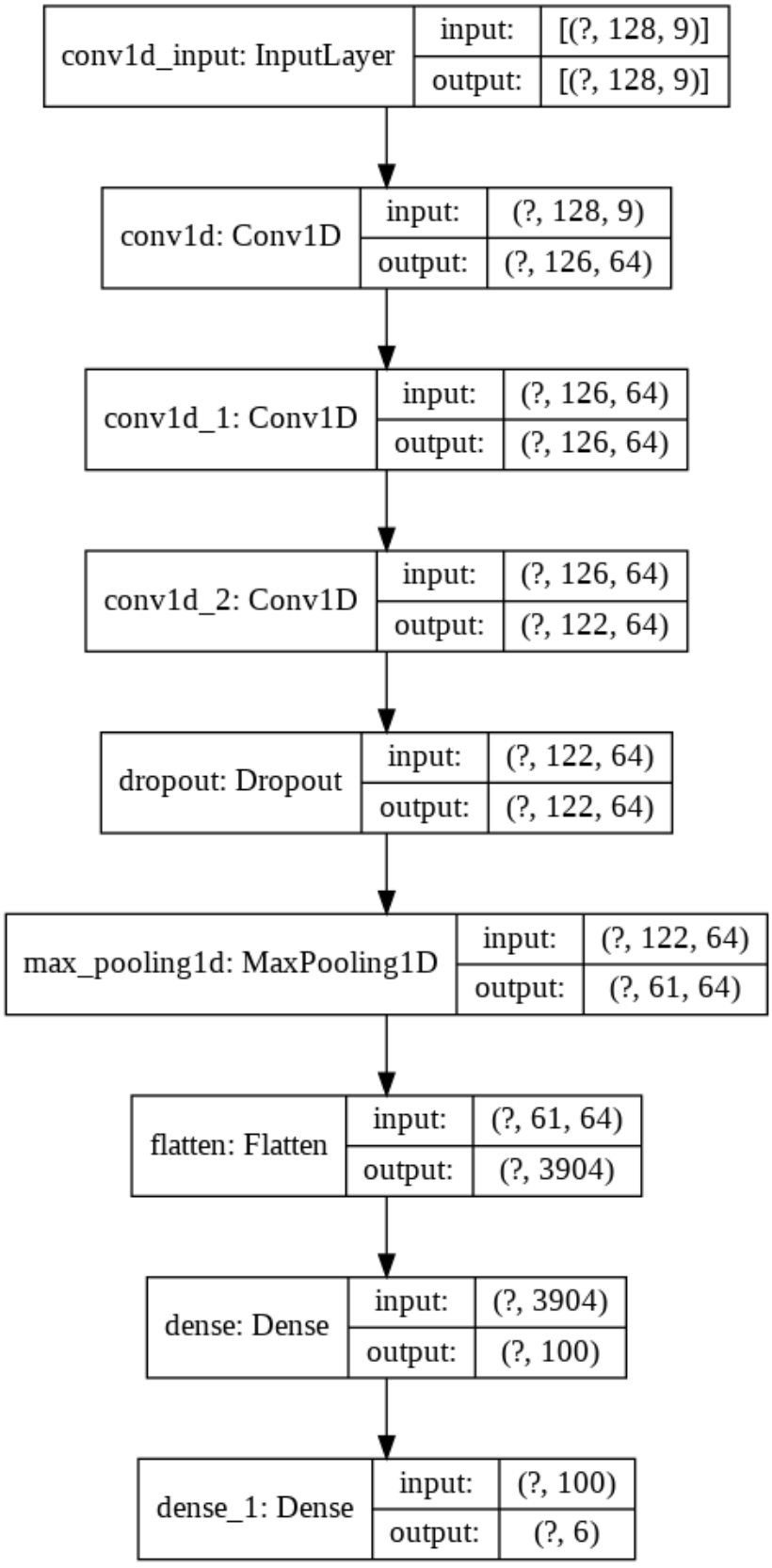

Fig 4.8: 1D CNN Model: Model Plot.

B. Accuracy and Loss plots

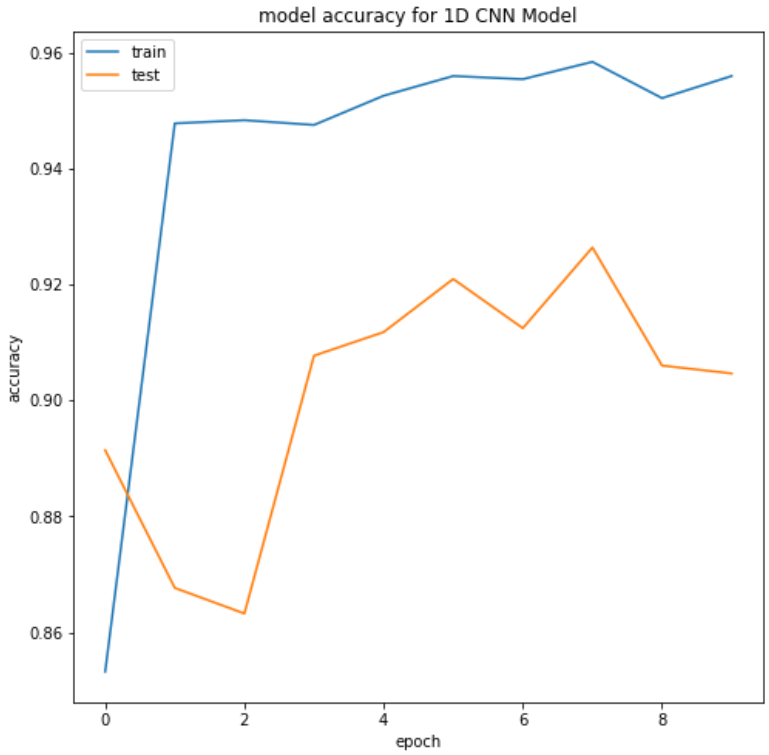

Fig 4.9: 1D CNN Model: Accuracy vs. Epochs Plot.

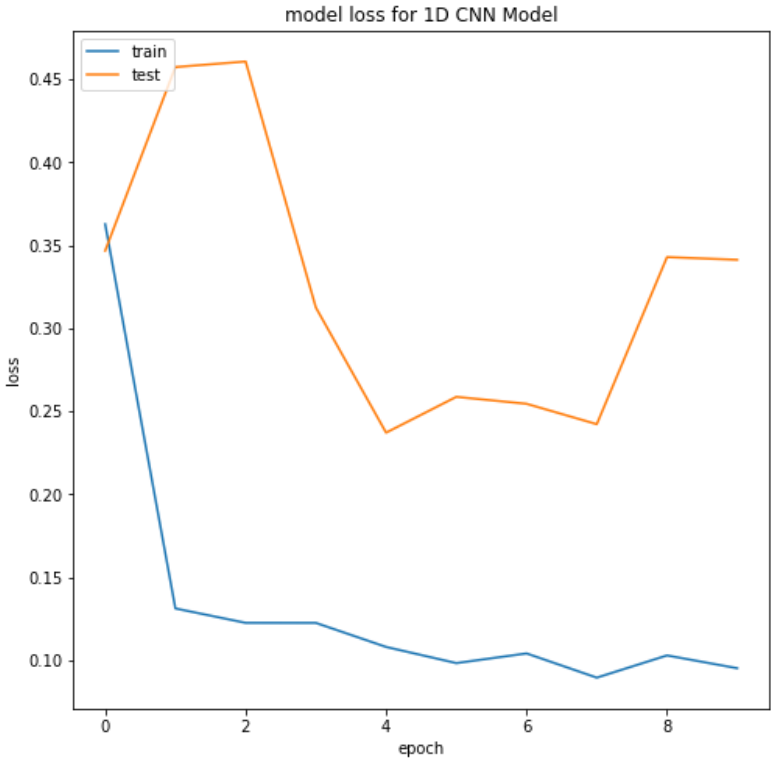

Fig 4.10: 1D CNN Model: Loss vs. Epochs Plot.

C. Confusion Matrix

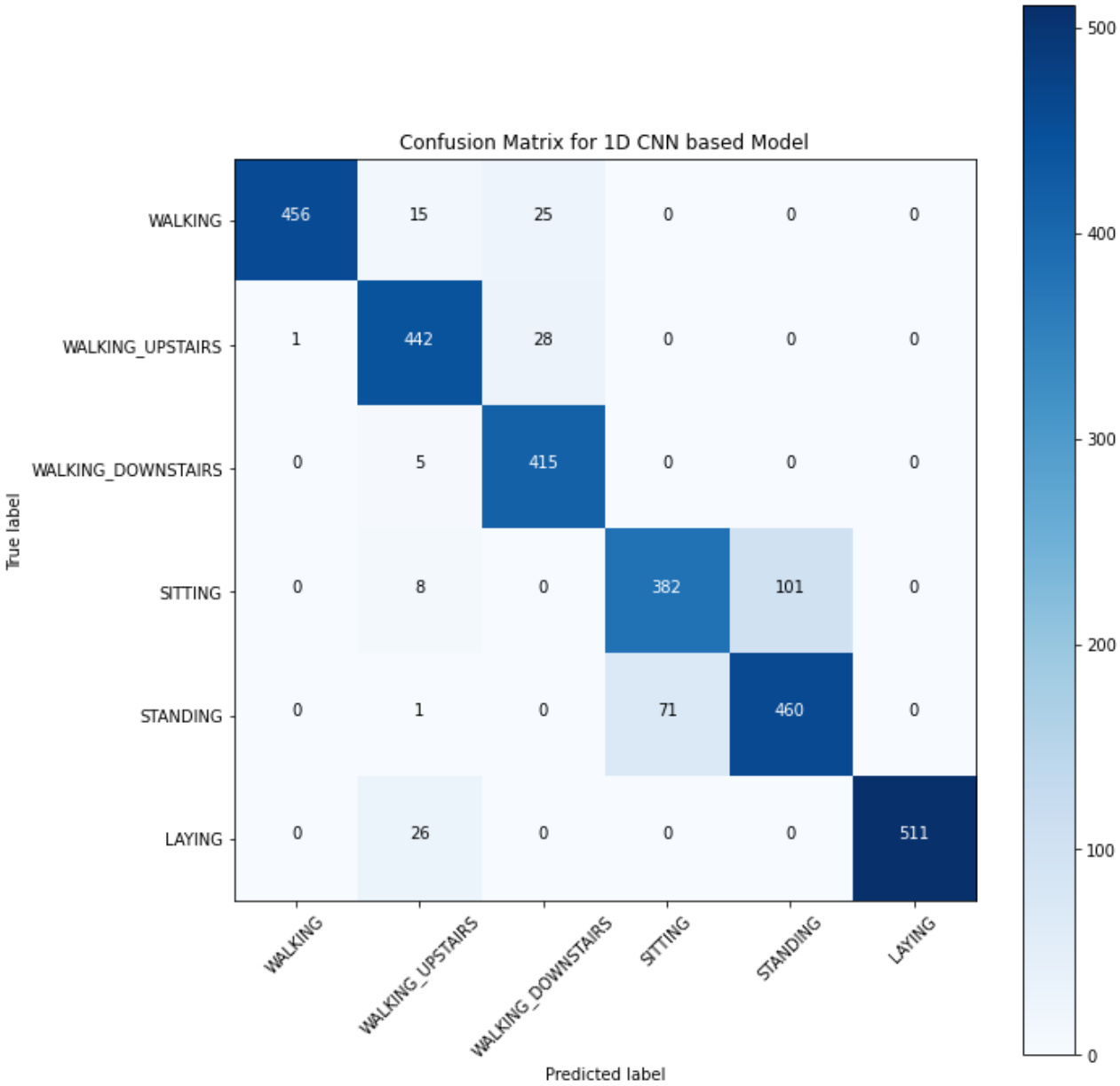

Fig 4.11: 1D CNN Model: Confusion Matrix.

D. Classification Report

Activity Classes	Precision	Recall	F1-Score	Support
WALKING	0.99781	0.91935	0.95698	496
WALKING UPSTAIRS	0.88934	0.93843	0.91322	471
WALKING DOWNSTAIRS	0.88675	0.98810	0.93468	420
SITTING	0.84327	0.77800	0.80932	491
STANDING	0.81996	0.86466	0.84172	532
LAYING	1.00000	0.95158	0.97519	537

Table 4.1: 1D CNN Model: Classification Report

	Accuracy	Precision	Recall	F1-Score
Macro	0.90465	0.90619	0.90669	0.90519
Weighted	0.90465	0.90719	0.90465	0.90472

Table 4.2: 1D CNN Model: Performance Metrices

4.2.1.2. LSTM model

A. Model plot and Summary

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 100)               44000
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense (Dense)                (None, 100)               10100
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 606
=================================================================
Total params: 54,706
Trainable params: 54,706
Non-trainable params: 0
_________________________________________________________________
```

Fig 4.12: LSTM Model: Summary

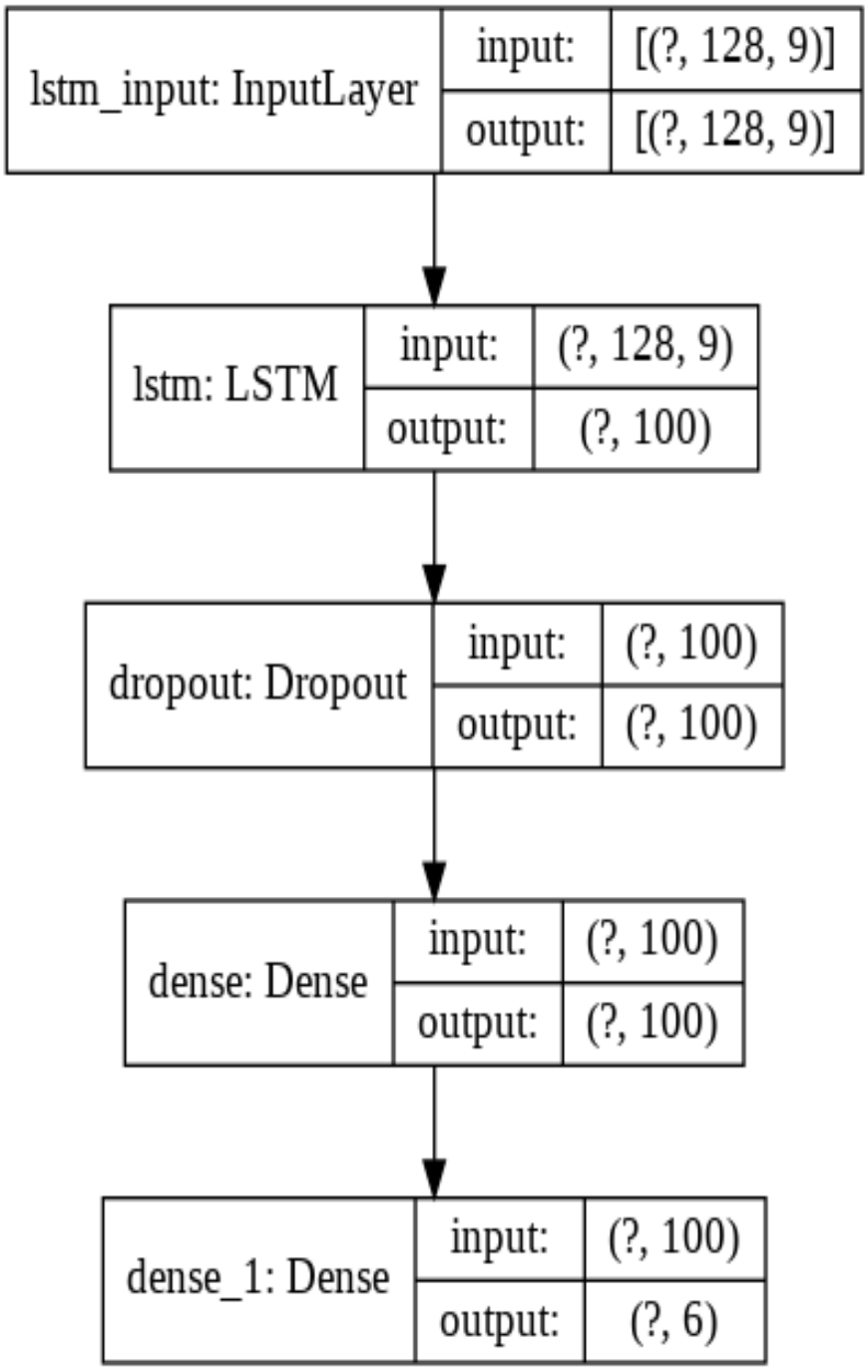

Fig 4.13: LSTM Model: Model Plot.

B. Accuracy and Loss plots

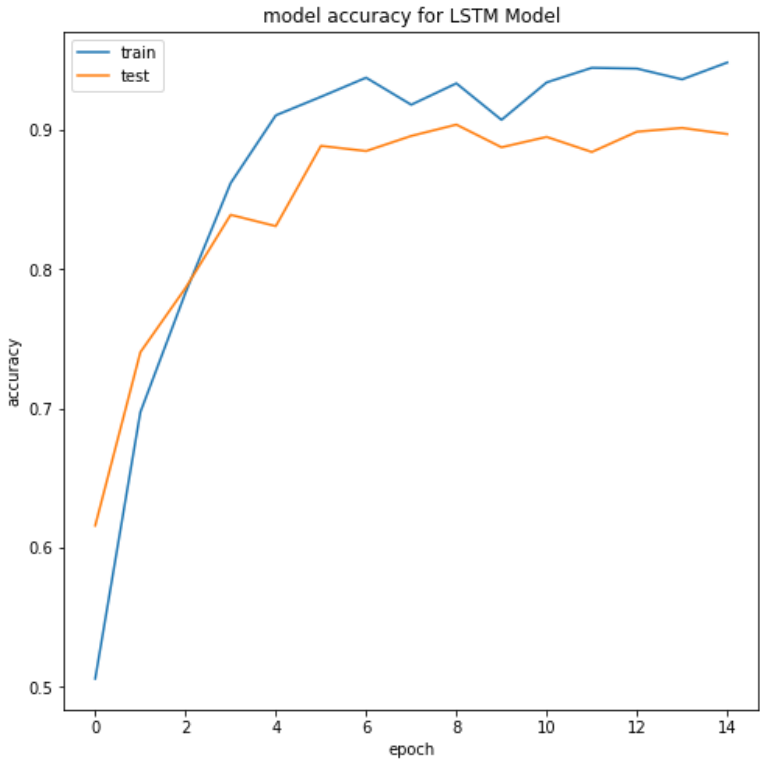

Fig 4.14: LSTM Model: Accuracy vs. Epochs Plot.

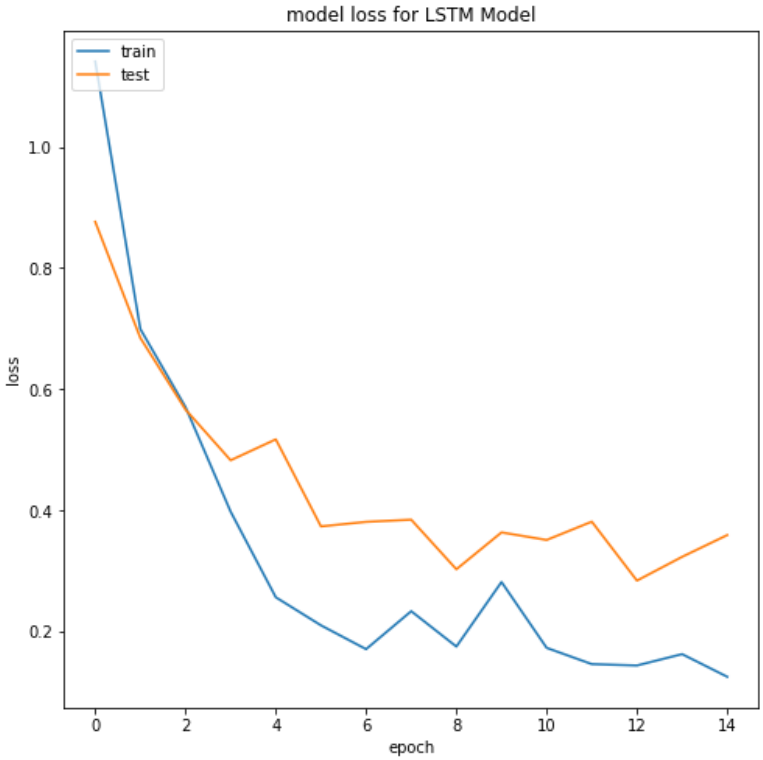

Fig 4.15: LSTM Model: Loss vs. Epochs Plot.

C. Confusion Matrix

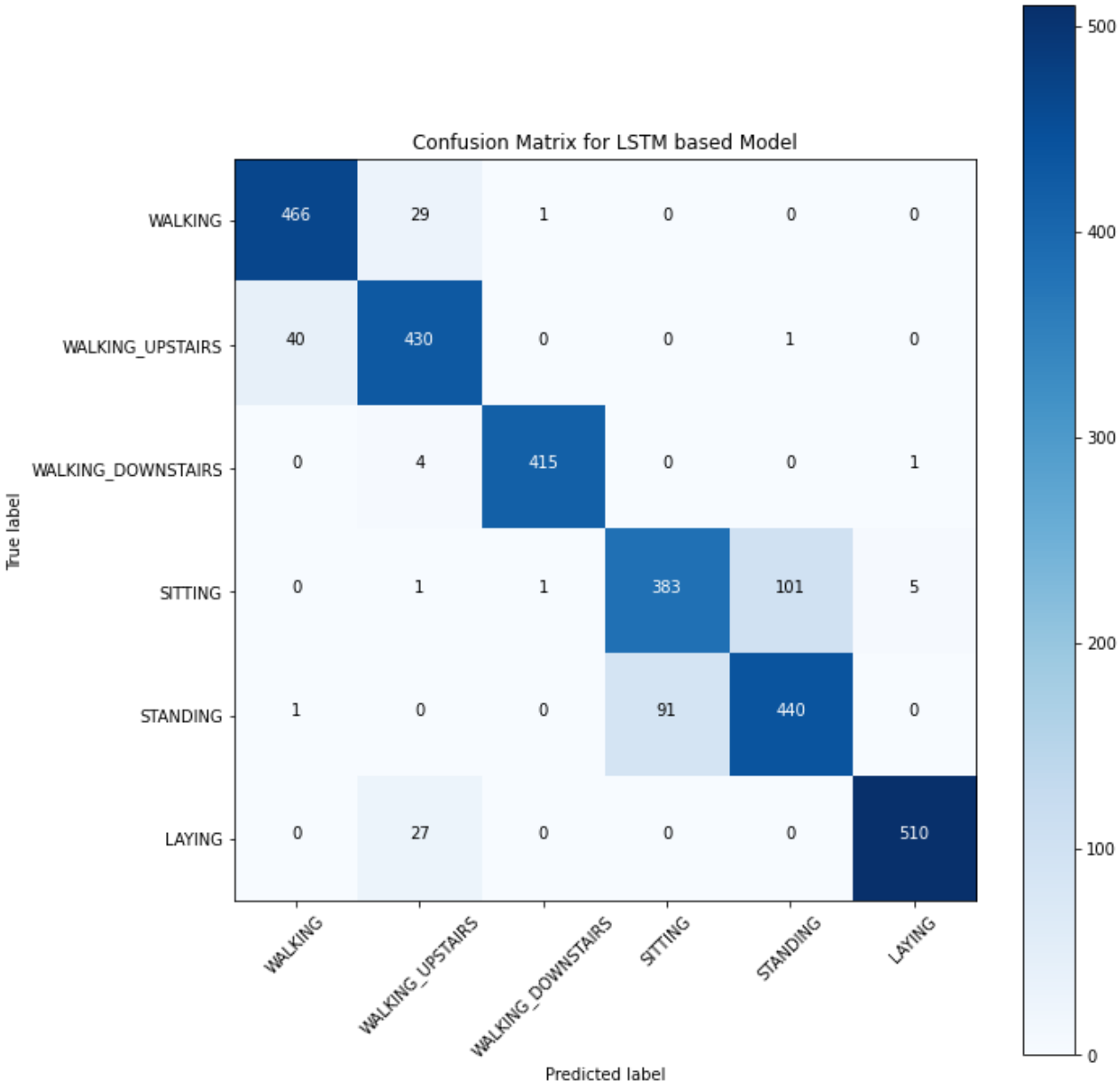

Fig 4.16: LSTM Model: Confusion Matrix.

D. Classification Report

Activity Classes	Precision	Recall	F1-Score	Support
WALKING	0.91913	0.93952	0.92921	496
WALKING UPSTAIRS	0.87576	0.91295	0.89397	471
WALKING DOWNSTAIRS	0.99520	0.98810	0.99164	420
SITTING	0.80802	0.78004	0.79378	491
STANDING	0.81181	0.82707	0.81937	532
LAYING	0.98837	0.94972	0.96866	537

Table 4.3: LSTM Model: Classification Report

	Accuracy	Precision	Recall	F1-Score
Macro	0.89718	0.89972	0.89957	0.89944
Weighted	0.89718	0.89777	0.89718	0.89727

Table 4.4: LSTM Model: Performance Metrices

4.2.1.3. CNN-LSTM Model

A. Model plot and Summary

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed_5 (TimeDist (None, None, 30, 64)      1792
_________________________________________________________________
time_distributed_6 (TimeDist (None, None, 28, 64)      12352
_________________________________________________________________
time_distributed_7 (TimeDist (None, None, 28, 64)      0
_________________________________________________________________
time_distributed_8 (TimeDist (None, None, 14, 64)      0
_________________________________________________________________
time_distributed_9 (TimeDist (None, None, 896)         0
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               398800
_________________________________________________________________
dropout_4 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_4 (Dense)              (None, 100)               10100
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 606
=================================================================
Total params: 423,650
Trainable params: 423,650
Non-trainable params: 0
_________________________________________________________________
```

Fig 4.17: CNN LSTM model: Summary

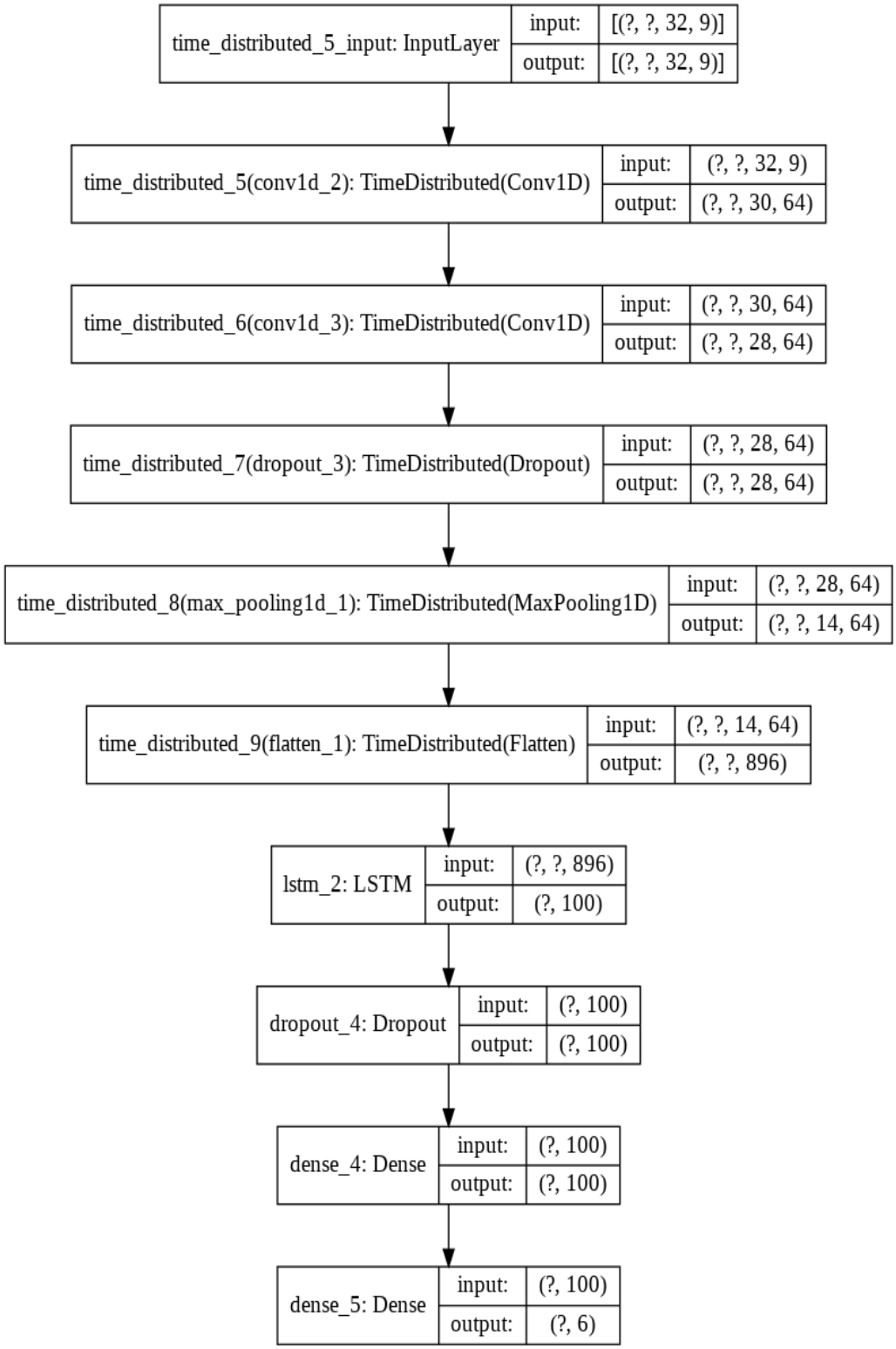

Fig 4.18: CNN-LSTM Model: Model Plot.

B. Accuracy and Loss plots

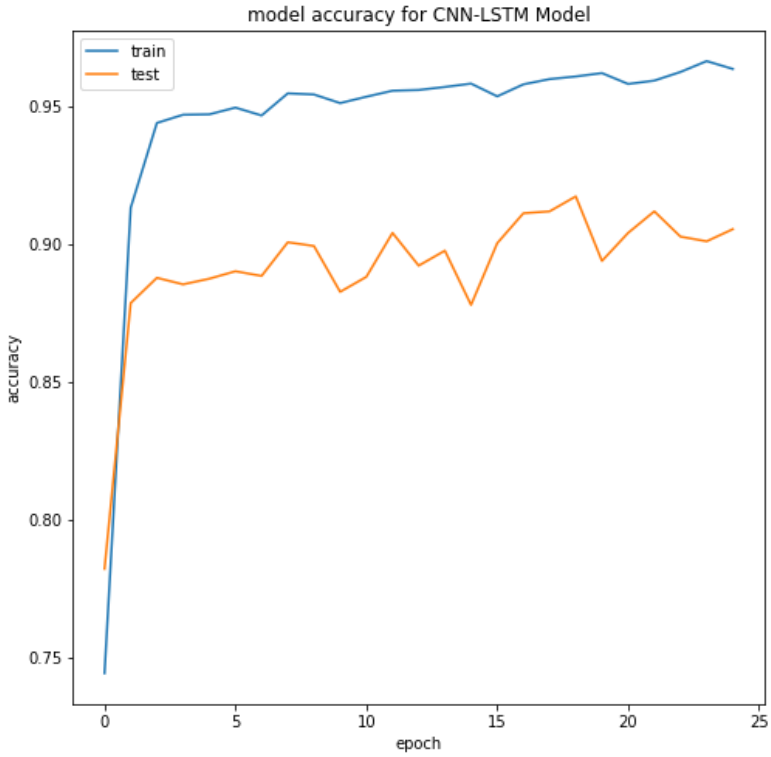

Fig 4.19: CNN-LSTM Model: Accuracy vs. Epochs Plot.

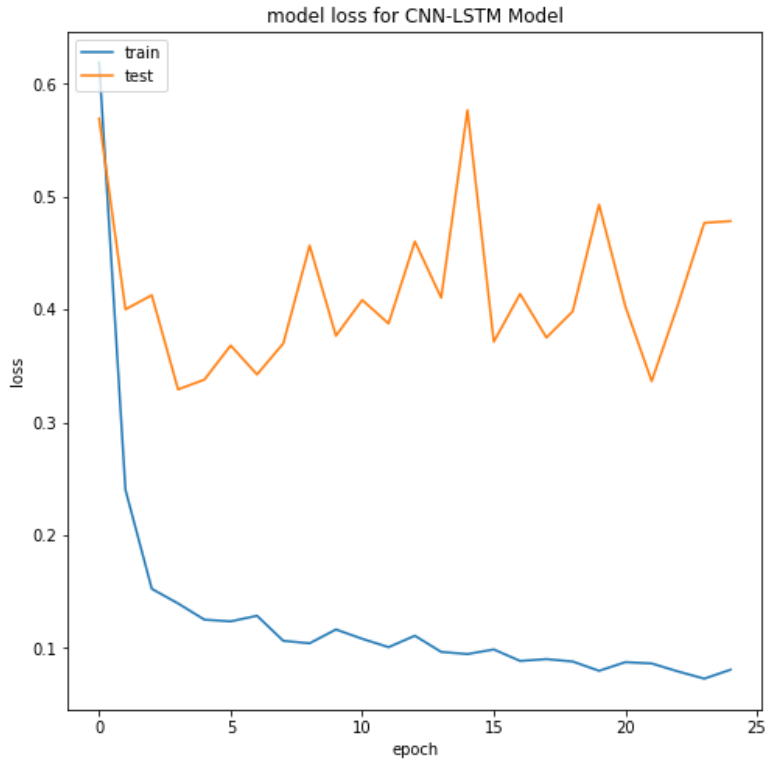

Fig 4.20: CNN-LSTM Model: Loss vs. Epochs Plot.

C. Confusion Matrix

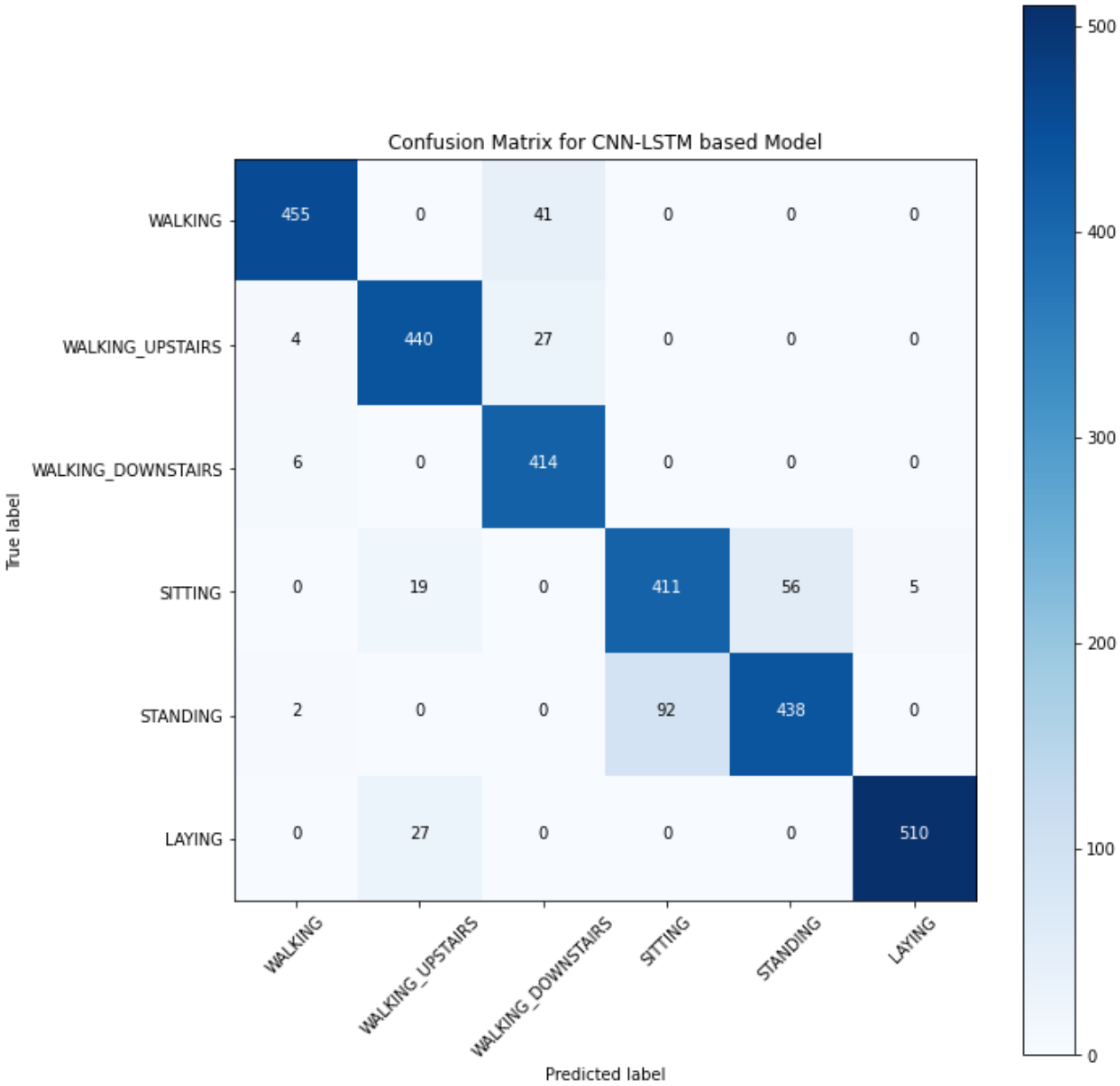

Fig 4.21: CNN-LSTM Model: Confusion Matrix.

D. Classification Report

Activity Classes	Precision	Recall	F1-Score	Support
WALKING	0.97430	0.91734	0.94496	496
WALKING UPSTAIRS	0.90535	0.93418	0.91954	471
WALKING DOWNSTAIRS	0.85892	0.98571	0.91796	420
SITTING	0.81710	0.83707	0.82696	491
STANDING	0.88664	0.82331	0.85380	532
LAYING	0.99029	0.94972	0.96958	537

Table 4.3: CNN-LSTM Model: Classification Report

	Accuracy	Precision	Recall	F1-Score
Macro	0.90533	0.90543	0.90789	0.90547
Weighted	0.90533	0.90773	0.90533	0.90542

Table 4.4: CNN-LSTM Model: Performance Metrices

4.2.1.4. ConvLSTM Model

A. Model plot and Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_lst_m2d (ConvLSTM2D)    (None, 1, 30, 64)         56320
_________________________________________________________________
dropout (Dropout)            (None, 1, 30, 64)         0
_________________________________________________________________
flatten (Flatten)            (None, 1920)              0
_________________________________________________________________
dense (Dense)                (None, 100)               192100
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 606
=================================================================
Total params: 249,026
Trainable params: 249,026
Non-trainable params: 0
_________________________________________________________________
```

Fig 4.22: ConvLSTM Model: Summary

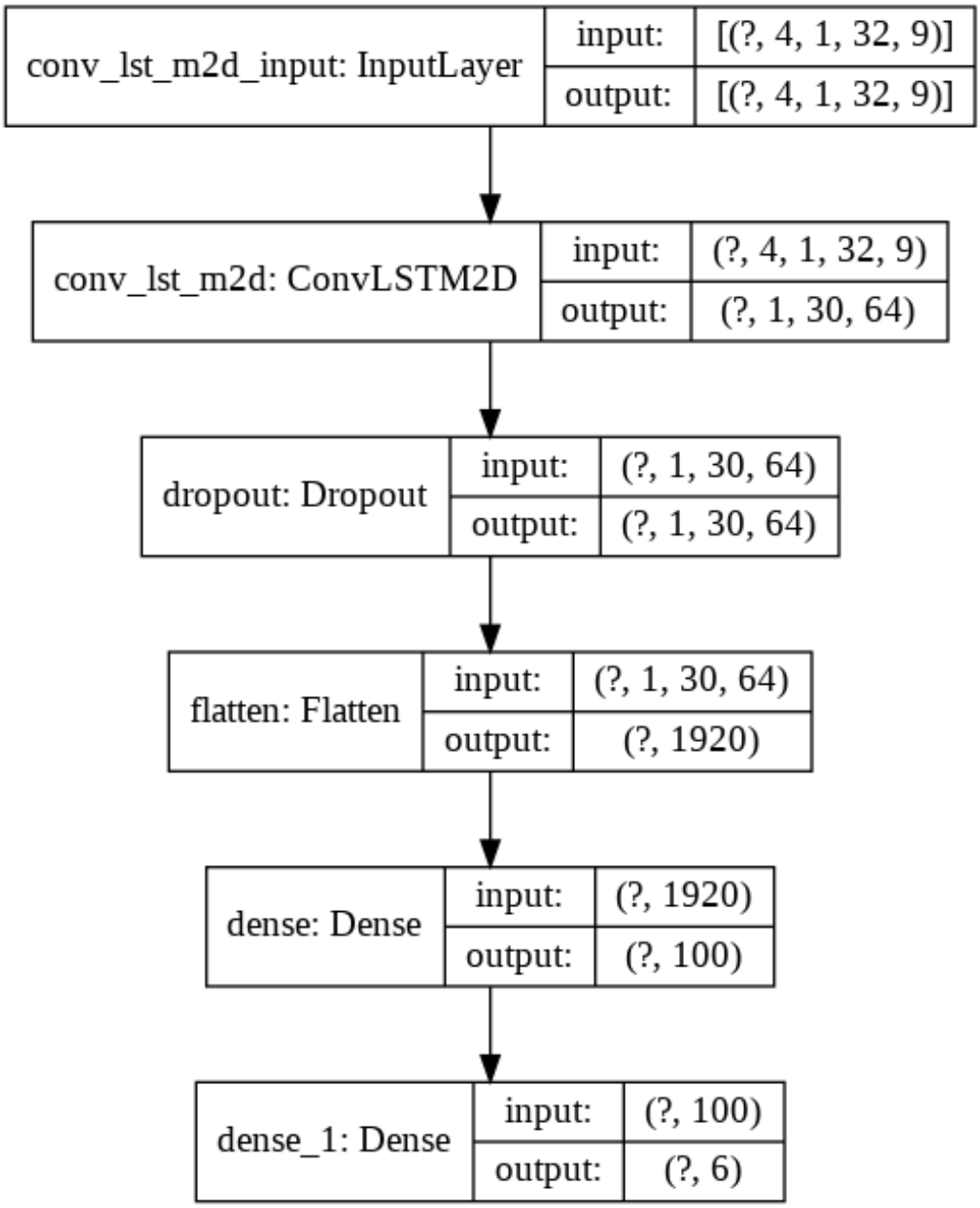

Fig 4.23: ConvLSTM Model: Model Plot.

B. Accuracy and Loss plots

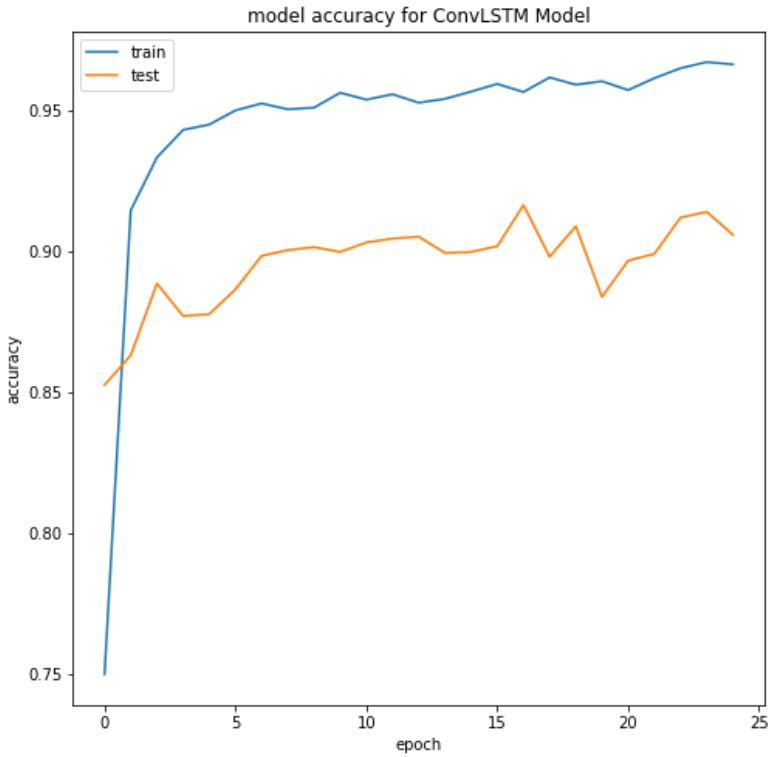

Fig 4.24: ConvLSTM Model: Accuracy vs. Epochs Plot.

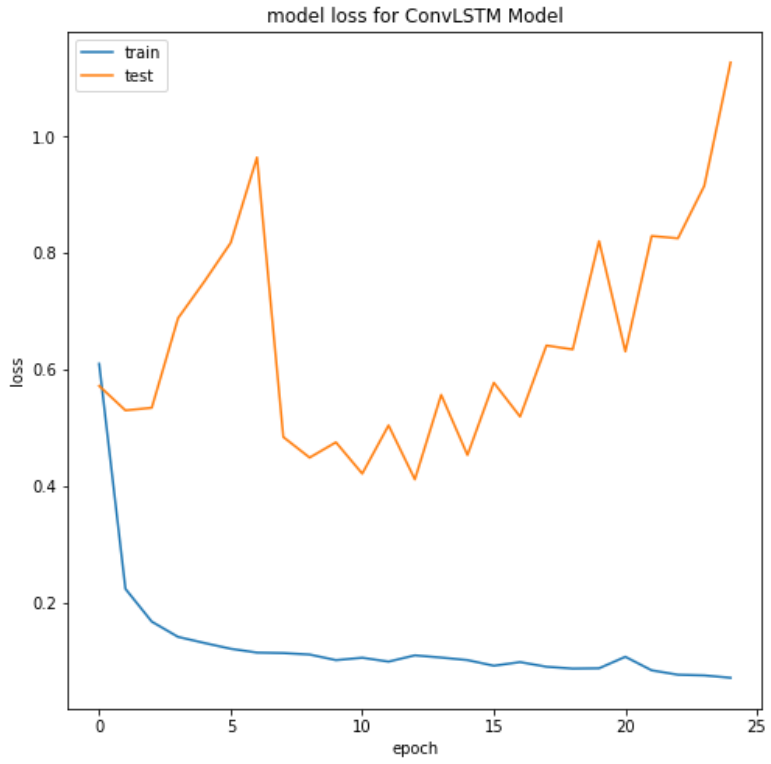

Fig 4.25: ConvLSTM Model: Loss vs. Epochs Plot.

C. Confusion Matrix

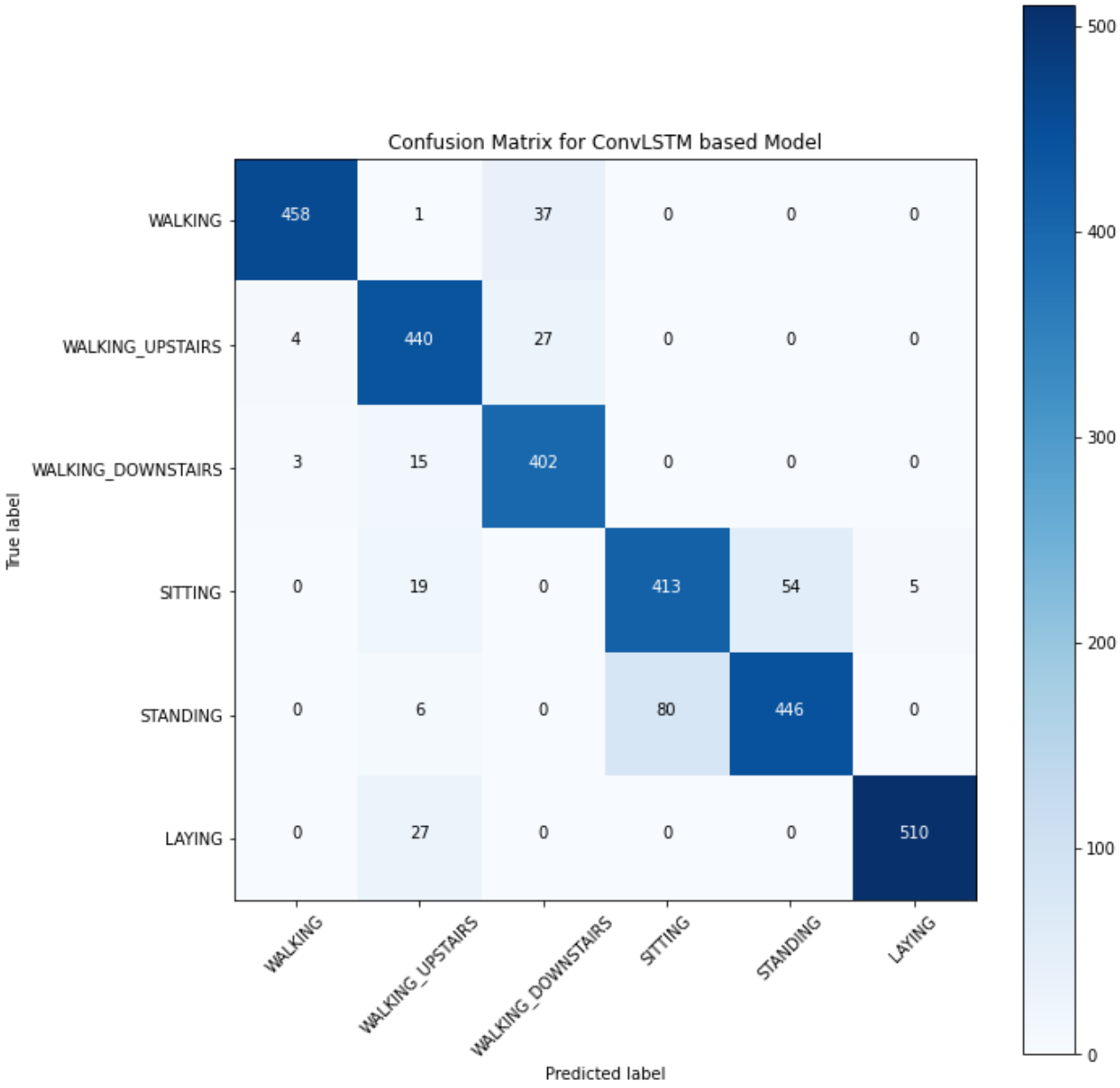

Fig 4.26: ConvLSTM Model: Confusion Matrix.

D. Classification Report

Activity Classes	Precision	Recall	F1-Score	Support
WALKING	0.98495	0.92339	0.95317	496
WALKING UPSTAIRS	0.86614	0.93418	0.89888	471
WALKING DOWNSTAIRS	0.86266	0.95714	0.90745	420
SITTING	0.83773	0.84114	0.83943	491
STANDING	0.89200	0.83835	0.86434	532
LAYING	0.99029	0.94972	0.96958	537

Table 4.7: ConvLSTM Model: Classification Report

	Accuracy	Precision	Recall	F1-Score
Macro	0.90567	0.90563	0.90732	0.90548
Weighted	0.90567	0.90820	0.90567	0.90598

Table 4.8: ConvLSTM Model: Performance Metrices

4.2.1.5. InnoHAR model

A. Model plot and Summary

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
incept1d_4 (Incept1D)        (None, 128, 256)          14336
_________________________________________________________________
incept1d_5 (Incept1D)        (None, 128, 248)          59704
_________________________________________________________________
incept1d_6 (Incept1D)        (None, 128, 176)          42736
_________________________________________________________________
max_pooling1d_9 (MaxPooling1 (None, 64, 176)           0
_________________________________________________________________
incept1d_7 (Incept1D)        (None, 64, 120)           18712
_________________________________________________________________
max_pooling1d_11 (MaxPooling (None, 32, 120)           0
_________________________________________________________________
gru_2 (GRU)                  (None, 32, 120)           87120
_________________________________________________________________
gru_3 (GRU)                  (None, 40)                19440
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 246
=================================================================
Total params: 242,294
Trainable params: 242,294
Non-trainable params: 0
_________________________________________________________________
```

Fig 4.27: InnoHAR Model: Summary.

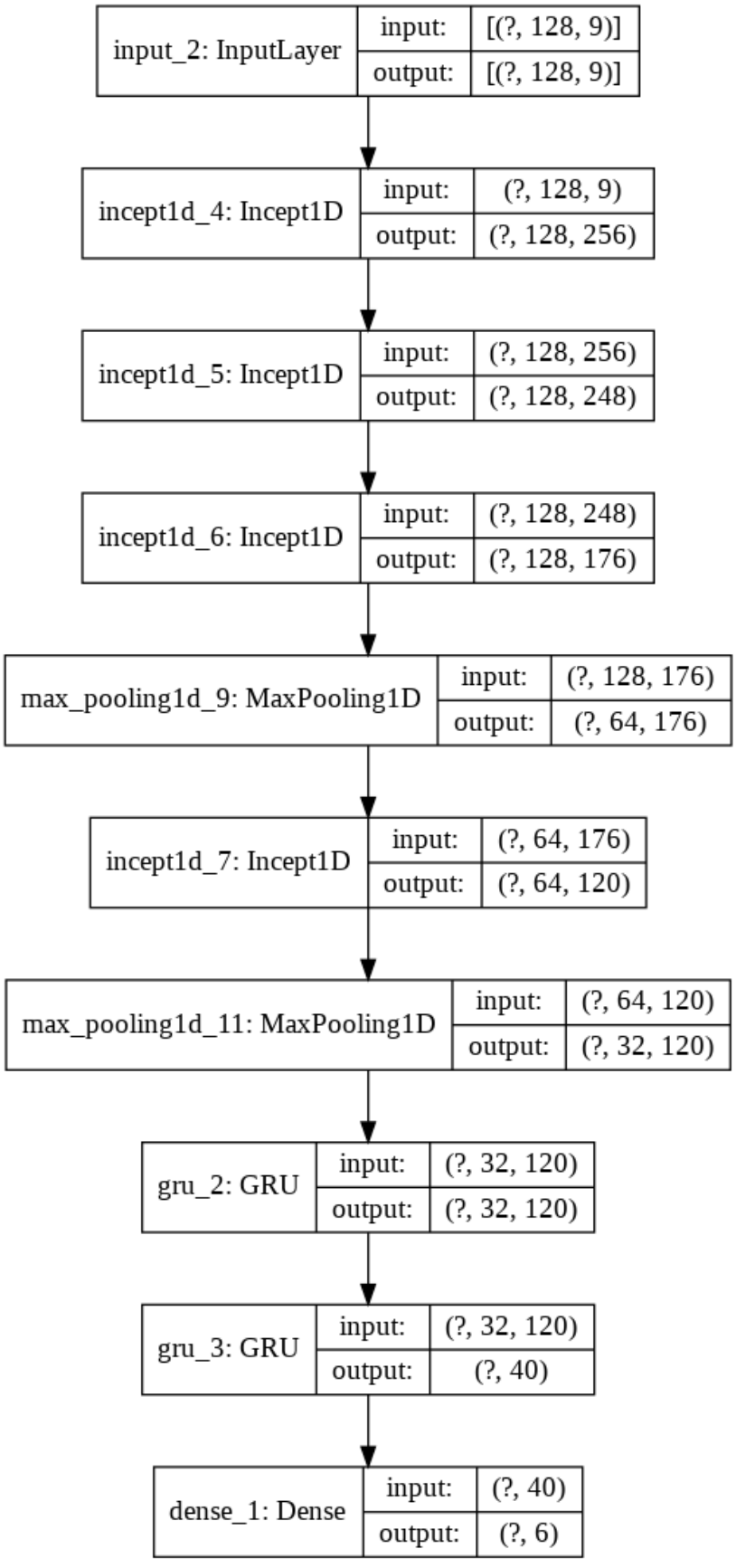

Fig 4.28: InnoHAR Model: Model Plot.

B. Accuracy and Loss plots

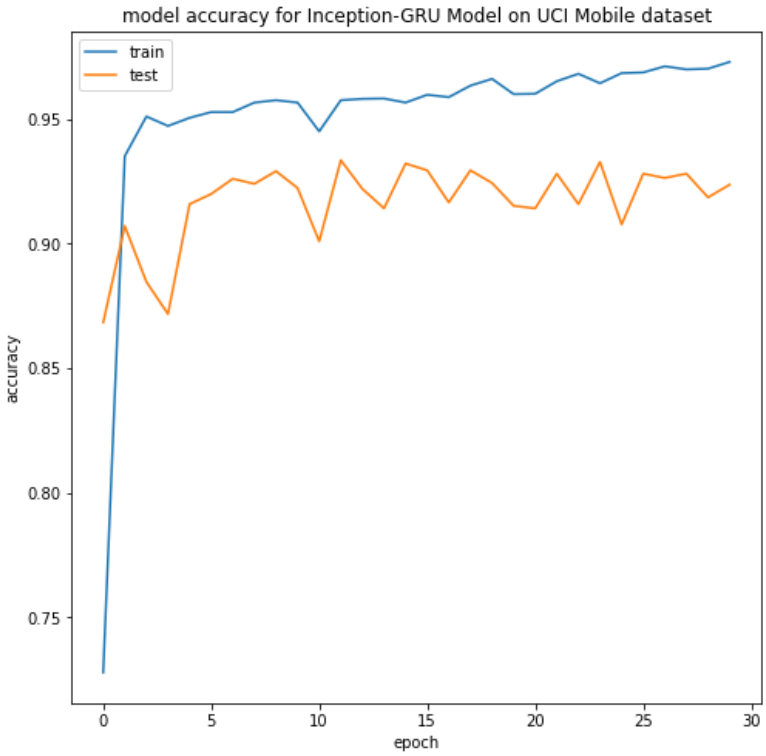

Fig 4.29: InnoHAR Model: Accuracy vs. Epochs Plot.

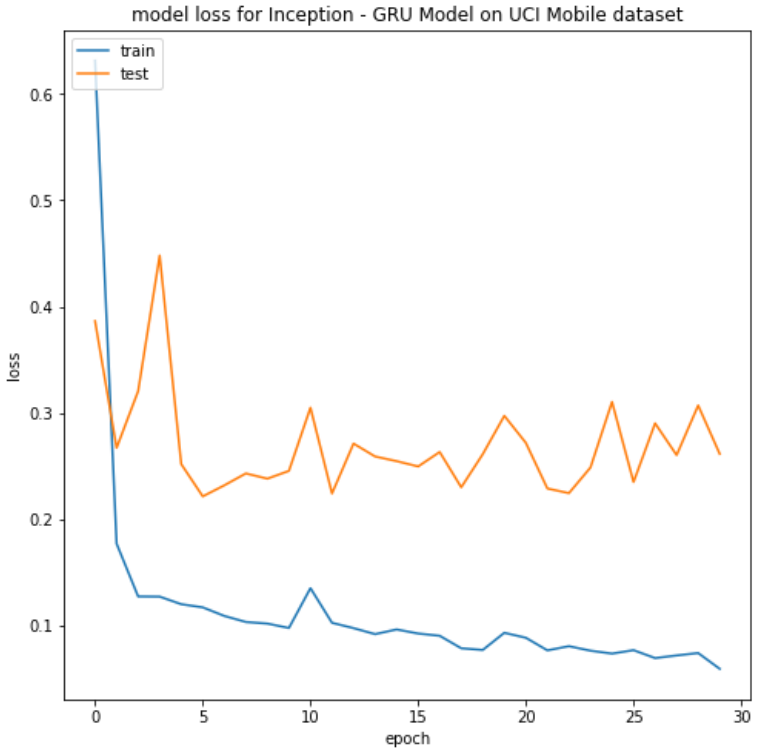

Fig 4.30: InnoHAR Model: Loss vs. Epochs Plot.

C. Confusion Matrix

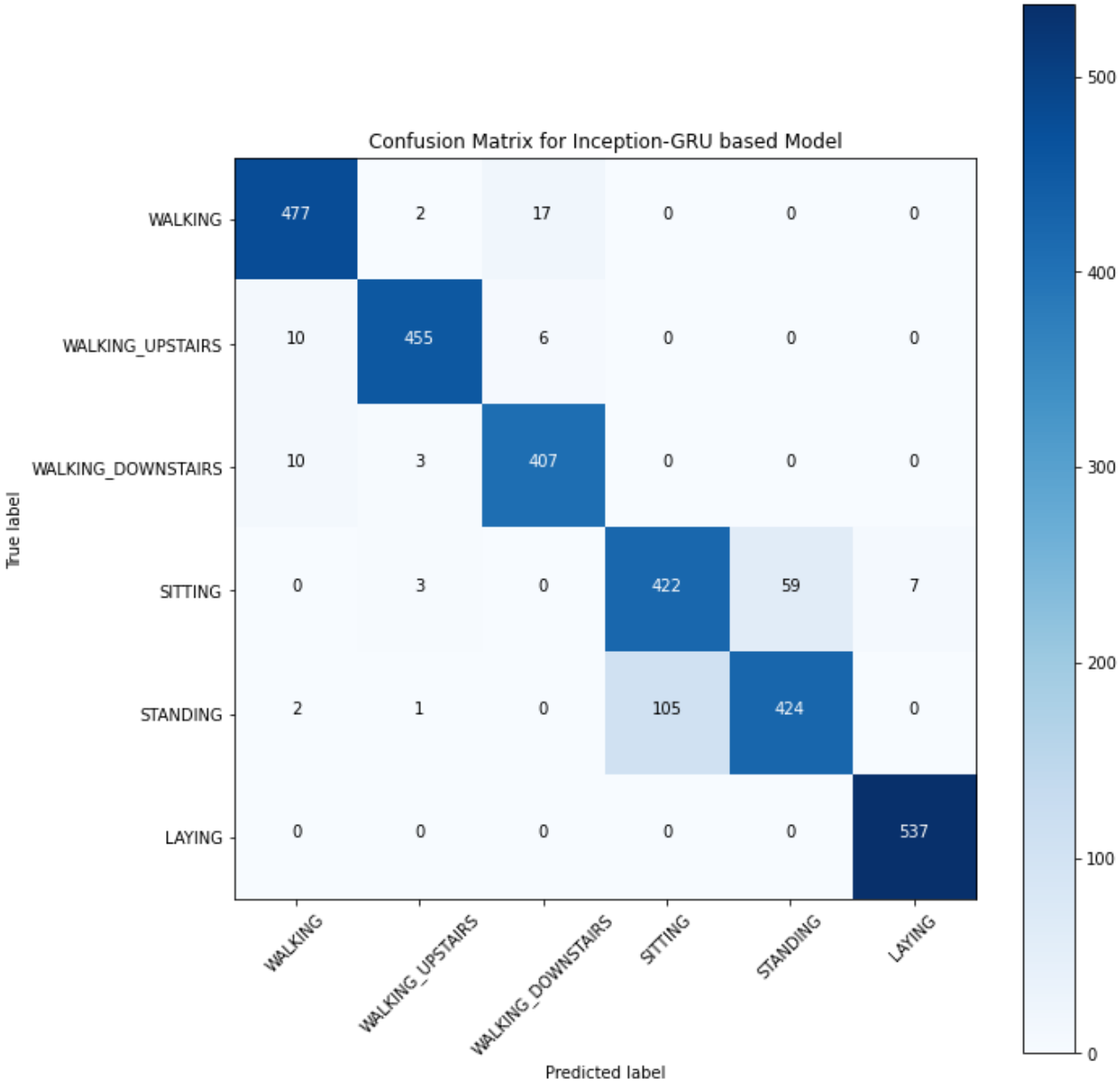

Fig 4.31: InnoHAR Model: Confusion Matrix.

D. Classification Report

Activity Classes	Precision	Recall	F1-Score	Support
WALKING	0.95591	0.96169	0.95879	496
WALKING UPSTAIRS	0.98060	0.96603	0.97326	471
WALKING DOWNSTAIRS	0.94651	0.96905	0.95765	420
SITTING	0.80076	0.85947	0.82908	491
STANDING	0.87785	0.79699	0.83547	532
LAYING	0.98713	1.00000	0.99352	537

Table 4.9: InnoHAR Model: Classification Report

	Accuracy	Precision	Recall	F1-Score
Macro	0.92365	0.92479	0.92554	0.92463
Weighted	0.92365	0.92426	0.92365	0.92340

Table 4.10: InnoHAR Model: Performance Metrices

4.2.2. Analysis of Results

Before the analysis of results, we must see if the dataset was balanced or not only then we will be able to get a clear picture of the performance of all the trained models. The below pie charts show the distributions of each activity classes in the training-set and test set.

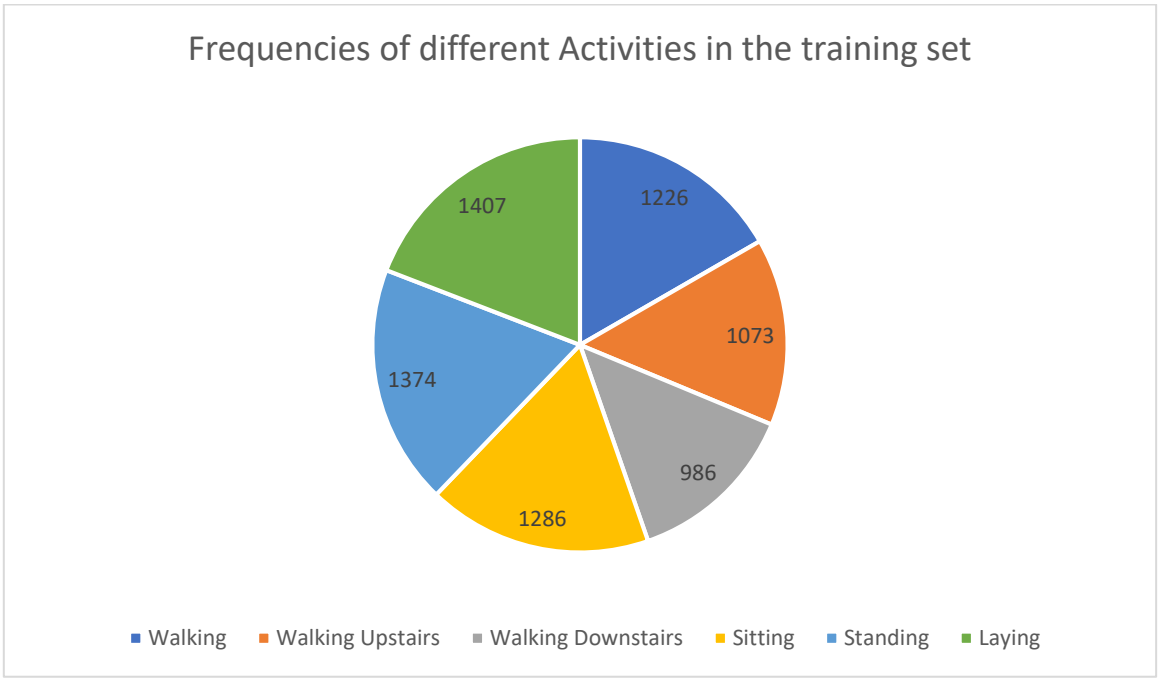

Fig 4.32: Distribution of different classes in the training set.

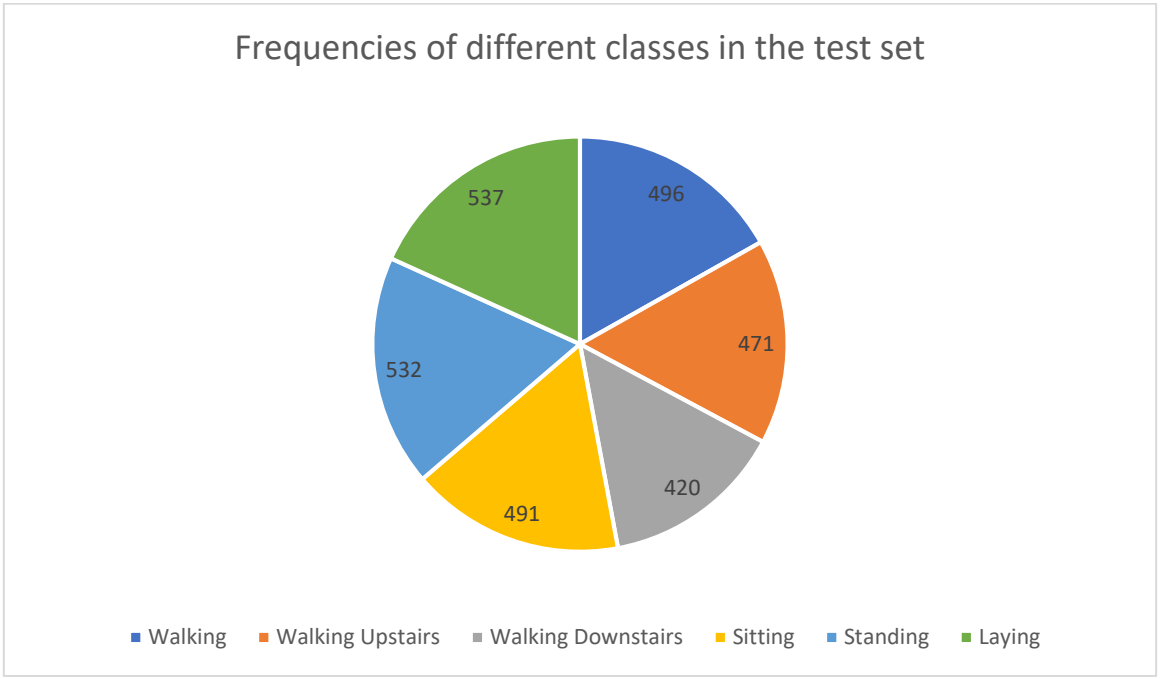

Fig 4.33: Distribution of different classes in the test set.

Looking at the pie-chart we can see that both training set and test set are fairly balanced as each of the activity classes contributes almost equal angles in both the pie charts. Since, we have a good balanced dataset, only accuracy is enough to compare all the models. We don't even need to calculate Precision, Recall and F-1 scores.

The average Accuracies of the five trained models are summarized in the table below.

Model	Accuracy
1D – CNN Model	90.465 %
LSTM Model	89.718 %
CNN-LSTM model	90.533 %
ConvLSTM model	90.567 %
InnoHAR model	92.365 %

Table 4.11: Accuracy Summary of the trained models.

From the table we can clearly see that the InnoHAR model outperforms all the other models with an approximate 2% increase in accuracy from other models which is huge.

In the below table we will compare the other metrices for all the five models in both macro mode which takes the direct average of calculated metrices for all the different classes in order to calculate the metric value for the model, as well as weighted mode which takes a weighted mean of all the metrices.

	Precision		Recall		F-1 Score	
	macro	weighted	macro	weighted	macro	weighted
1D-CNN model	0.90619	0.90719	0.90669	0.90465	0.90519	0.90472
LSTM model	0.89972	0.89777	0.89957	0.89718	0.89944	0.89727
CNN-LSTM model	0.90543	0.90773	0.90789	0.90533	0.90547	0.90542
ConvLSTM model	0.90563	0.90820	0.90732	0.90567	0.90548	0.90598
InnoHAR model	0.92479	0.92426	0.92554	0.92365	0.92463	0.92463

Table 4.12: Performance metrices Summary of the trained models.

From both the table one can easily conclude that the InnoHAR model outperforms every other model on which the experiment was performed. So, It can be assumed that the InnoHAR model can be used for Human Activity Recognition as it gave the best results on this dataset.

Below is a histogram depicting the performance metrices of various models used.

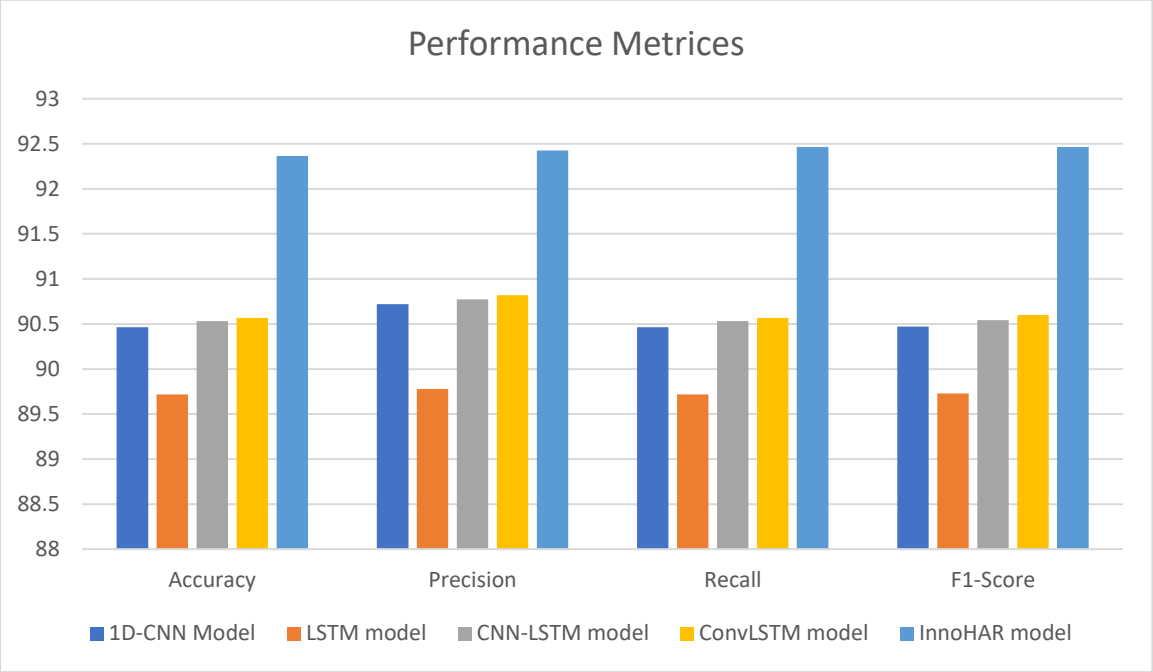

Fig 4.34: Comparing various models.

We can clearly see the superiority of InnoHAR model against the other four architectures.

Chapter 5

Conclusion

5.1 Conclusions

This project presents a very efficient deep-neural network architecture for complex human activity recognition problem. This architecture can be used in HAR research as well as different applications like elderly care, fitness tracking using smart wearables, monitoring patients at hospitals etc. We built the InnoHAR model in this project and compared it to different very efficient architecture commonly used in HAR problem to find that the InnoHAR model performs far better than any other model with a whooping 2% more accuracy than other efficient models currently being used for research and HAR applications. We successfully verified that a well performing model for Image classification like GoogLeNet can be modified to suit the needs of time-series data-based problems like HAR. We also saw that due to the sparse nature of the Inception Architecture, the InnoHAR model was easier and faster to train. So, we can conclude that the InnoHAR model can be successfully used for various Human Activity Recognition tasks.

5.2 Future scope of work

Although we did test the InnoHAR architecture on UCI's Human Activity Recognition Using Smartphones Dataset, the dataset itself is very ideal. It is well-balanced and it uses only smartphone integrated sensors. Because of this reason more verification of the architecture is needed on some other unbalanced datasets which are not ideal.

Other than just the smartphone sensors, the HAR systems often use multiple different types of sensors wearable by the user to better classify their activities. We must find out if test this architecture on such imbalanced datasets which use many sensors to gather user data. A few other datasets like the Opportunity dataset, PAMAP2 dataset can be used for this purpose to perform extensive testing and to closely observe the behavior of this architecture against the other models.

References

1. Dataset : from UCI Machine Learning Repository, https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones

2. "Going Deeper with Convolutions" published by Christian Szegedy Google Inc., Wei Liu University of North Carolina, Chapel Hill Yangqins Jia Google Inc. and team.

   https://arxiv.org/pdf/1409.4842v1.pdf

3. "InnoHAR: A Deep Neural Network for Complex Human Activity Recognition" – IEEE (2018), Cheng Xu (student member, IEEE), Duo Chai, Jie He, Xiaotong Zhang (senior member, IEEE), and Shihong Duan.

   https://www.researchgate.net/publication/330077488_InnoHAR_A_Deep_Neural_Network_for_Complex_Human_Activity_Recognition

4. Various articles from different sources, e.g.-

   https://towardsdatascience.com/

   https://medium.com/skyshidigital/getting-started-with-keras-624dbf106c87

   https://medium.com/topic/data-science

5. Keras official documentation

   https://keras.io/api/

6. NumPy and Matplotlib official documentations

   https://numpy.org/doc/1.19/

   https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.html