Recurrent Neural Network Models for Human Activity Recognition

IE7860: Intelligent Analytics

Final Term Project

26th April 2020

Team:

Mathai Paul (gx3081)

Theju Chikkathamme Gowda (gy1672)

ABSTRACT

The objective of this project was classification of accelerometer data recorded by smart phones into pre-defined human activities. The source of our dataset was UCI repository. Davide Anguita, et al. from the University of Genova, Italy contributed the data in their 2012 paper titled "Human Activity Recognition on Smartphones using a Multiclass Hardware-Friendly Support Vector Machine (D. Anguita, 2012). Human Activity Recognition database is built from the recordings of 30 subjects performing activities of daily living (ADL) while carrying a waist-mounted smartphone. Creating features from the time series data based on fixed-sized windows and training them on machine learning models is a classical approach to this problem. In this project, deep learning methods such as recurrent neural networks like as LSTMs and variations that make use of one-dimensional convolutional neural networks or CNNs was used.

INTRODUCTION

Human Activity Recognition (HAR) is the process of identifying what a person is doing based on sensor readings. Activities are generally divided into classes, and the goal of HAR is to identify which class of activity is performed. In this project we are using machine learning methods to analyse the data. XGBoost, LSTM, ConvoLSTM and CNN LSTM methods have been explored to classify six different activities: walking, walking upstairs, walking downstairs, standing, sitting and laying. Based on the results models are compared for their performance with their measuring metric.

XGBoost

XGBoost is an acronym for eXtreme Gradient Boosting (Guestrin, 2016). The algorithm is an implementation of a gradient boosting machine (Friedman, 2001). Gradient boosting machines are based on an ensemble of decision trees, where multiple weak learner trees are used in combination as a collective to give better predictions than individual trees can do (Johnson, 2013). In comparison with older gradient boosting algorithms, XGBoost has superior regularization and better handling of missing values, as well as much improved efficiency (Guestrin, 2016). XGBoost is a very efficient tree ensemble method, which contains an extensive set of regularization mechanisms to prevent overfitting.

Recurrent Neural Networks (RNN)

Recurrent neural networks are designed to learn from sequence data over time. Compared to a classical approach, using a Recurrent Neural Networks (RNN) with Long Short-Term Memory cells (LSTMs) does not require any feature engineering. Data can be fed directly into the neural network. Recently, LSTMs and its variations which uses one-dimensional convolutional

neural networks or CNNs have been shown to promising in solving challenging activity recognition tasks using feature learning.

LSTM

Most prominent type of RNN is the long short-term memory network, or LSTM, is widely used as its design overcomes the difficulties in training a recurring neural network on sequence data. LSTM model has layers which is comprised of units that have gates that govern input, output, and recurrent connections. Each LSTM unit acquires internal memory as an input sequence is read, this can be used as a type of local variable or memory register. The LSTM maps each window of sensor data to an activity.

CNN-LSTM

Commonly LSTM is used in conjunction with a CNN on HAR problems. With a CNN model is features are extracted from a subsequence of raw sample data. For each subsequence, the output features from the CNN are then interpreted by an LSTM in aggregate. CNNs layers are capable of extracting features from input sequences, in this case windows of input accelerometer data.

ConvLSTM

A further extension of the CNN-LSTM idea was to perform the convolutions of the CNN as part of the LSTM. This combination is called a Convolutional LSTM, or ConvLSTM, this model is also used for spatio-temporal data. Unlike the CNN LSTM, the ConvLSTM uses convolutions directly as part of reading input into the LSTM units themselves.

METHODOLOGY

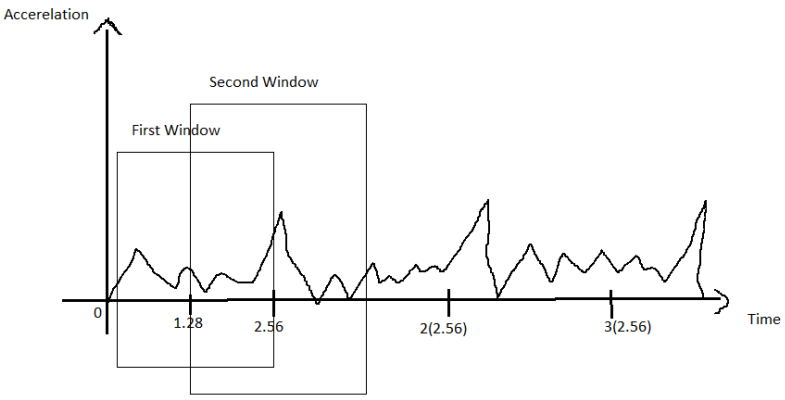
Methodology adopted in this study involves the following steps. Every model was performed parallelly due to computational limitations.

- Data Loading
- Data Splitting
- Exploratory Data Analysis
- Hyper-Parameter Tuning
- Explainable AI- SHAP
- Model fit
- Model Evaluation
- Model Repeat
- Result Summary

Data Loading

This dataset is collected from 30 persons performing different activities with a smartphone tied to their waists. Sensors in that smartphone are used to record the data. This experiment was video recorded to label the data manually. Using Accelerometer, we can measure Acceleration, using Gyroscope we can measure Angular Velocity. Sensors have captured '3-axial linear acceleration'(*tAcc-XYZ*) from the accelerometer and '3-axial angular velocity' (*tGyro-XYZ*) from Gyroscope with several variations. (D. Anguita, 2012) Our input is 6-Time Series data and output is the predicted class of activity, thus making it a Multi-class Classification problem.

The dataset is pre-processed by applying noise filters and then sampled in fixed-width windows of 2.56 seconds each with a 50% overlap. ie., each window has 128 readings. A sampling rate of 50Hz was used. A feature vector is obtained from each window by calculating variables from the time and frequency domain.



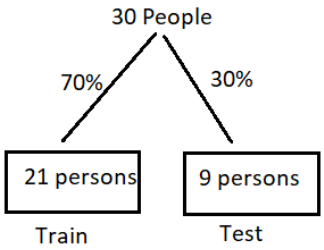
Dividing data into windows of size 2.56 sec

For every signal, now we have 128-dimension vectors. The activities were classified as horizontal walking, walking upstairs, walking downstairs, sitting, standing and laying. Finally, we have a vector of features with a total of 561 predictors.

Data Splitting

The data set is split into training and testing datasets by 30% of the 30 participants for the test and 70% of the participants randomly selected for training dataset. This exercise of random selection ensures that the test data is independent of the training data. The training set contains 7352 samples while the test set contains 2947 samples. As it is a time series data,

multiple samples from the same participant performing the same activity is included in the data set. Any form of random selection of samples such as cross validation or bootstrapping will not produce an independent data set.

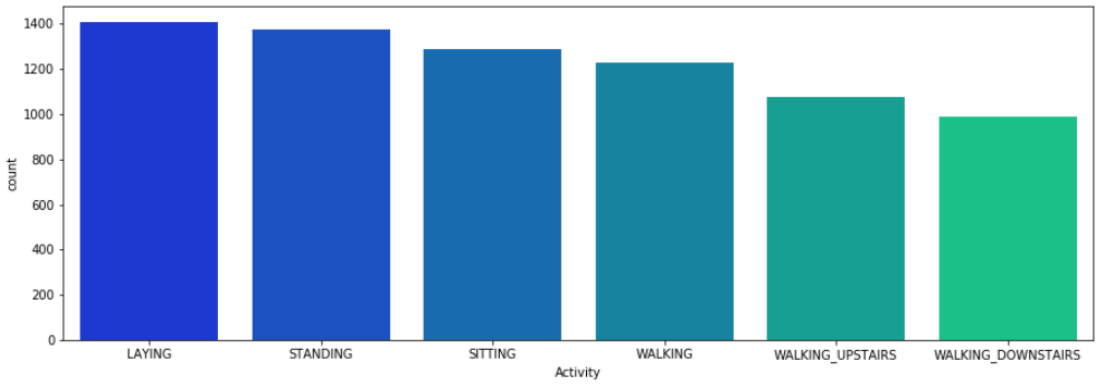


Exploratory Data Analysis

Dataset is checked for its shape; its datatypes and the dataset are checked for any null values among all its features. Fortunately, there were no missing or null values in the dataset. Further dataset was explored for analysis before building the models

Exploratory Data Analysis (EDA) is the most important part before training any kind of model, using EDA we can understand a lot about our data, and it tells how to train a model. We can ask and answer numerous questions to understand the data. One such question can be:

Q1 Identify class distribution of dataset: We can realize that the dataset is balanced.



Q2 How active are the participants?

We can count the number of sensor measurements for each activity of every participant.

subject	LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
1	50	47	53	95	49	53
3	62	52	61	58	49	59
5	52	44	56	56	47	47
6	57	55	57	57	48	51
7	52	48	53	57	47	51
8	54	46	54	48	38	41
11	57	53	47	59	46	54
14	51	54	60	59	45	54
15	72	59	53	54	42	48
16	70	69	78	51	47	51
17	71	64	78	61	46	48
19	83	73	73	52	39	40
21	90	85	89	52	45	47
22	72	62	63	46	36	42
23	72	68	68	59	54	51
25	73	65	74	74	58	65
26	76	78	74	59	50	55
27	74	70	80	57	44	51
28	80	72	79	54	46	51
29	69	60	65	53	48	49
30	70	62	59	65	62	65

Q3 For how many hours each activity is done by participants.

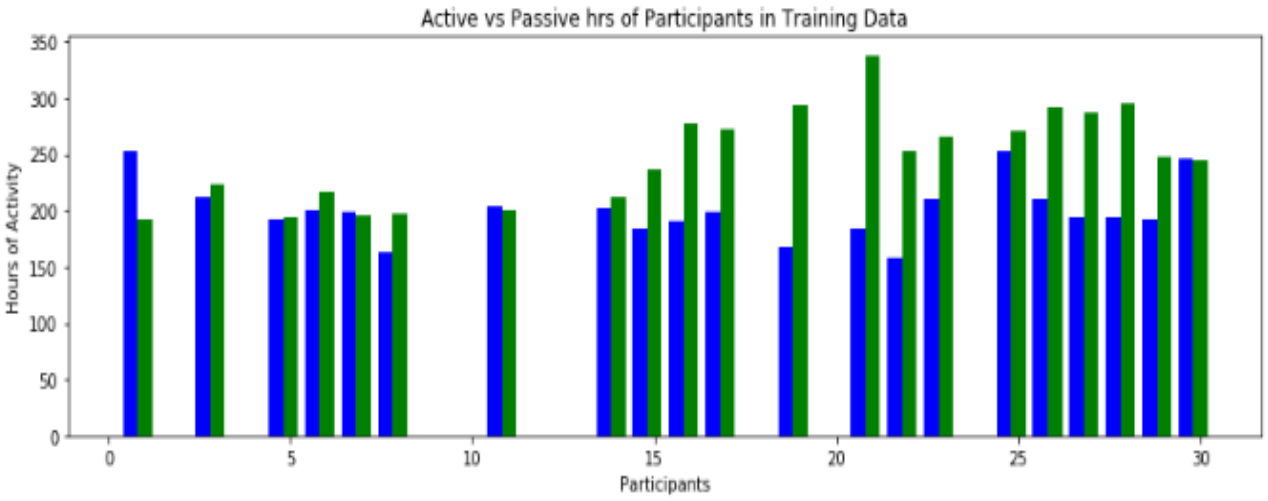
As we know that the sampling is done for 1.28 secs, we can multiply this by the count to obtain the time duration.

subject	LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
1	64.00	60.16	67.84	121.60	62.72	67.84
3	79.36	66.56	78.08	74.24	62.72	75.52
5	66.56	56.32	71.68	71.68	60.16	60.16
6	72.96	70.40	72.96	72.96	61.44	65.28
7	66.56	61.44	67.84	72.96	60.16	65.28
8	69.12	58.88	69.12	61.44	48.64	52.48
11	72.96	67.84	60.16	75.52	58.88	69.12
14	65.28	69.12	76.80	75.52	57.60	69.12
15	92.16	75.52	67.84	69.12	53.76	61.44
16	89.60	88.32	99.84	65.28	60.16	65.28
17	90.88	81.92	99.84	78.08	58.88	61.44
19	106.24	93.44	93.44	66.56	49.92	51.20
21	115.20	108.80	113.92	66.56	57.60	60.16
22	92.16	79.36	80.64	58.88	46.08	53.76
23	92.16	87.04	87.04	75.52	69.12	65.28
25	93.44	83.20	94.72	94.72	74.24	83.20
26	97.28	99.84	94.72	75.52	64.00	70.40
27	94.72	89.60	102.40	72.96	56.32	65.28
28	102.40	92.16	101.12	69.12	58.88	65.28
29	88.32	76.80	83.20	67.84	61.44	62.72
30	89.60	79.36	75.52	83.20	79.36	83.20

Q4 Active vs Passive hours for Each participant of training dataset

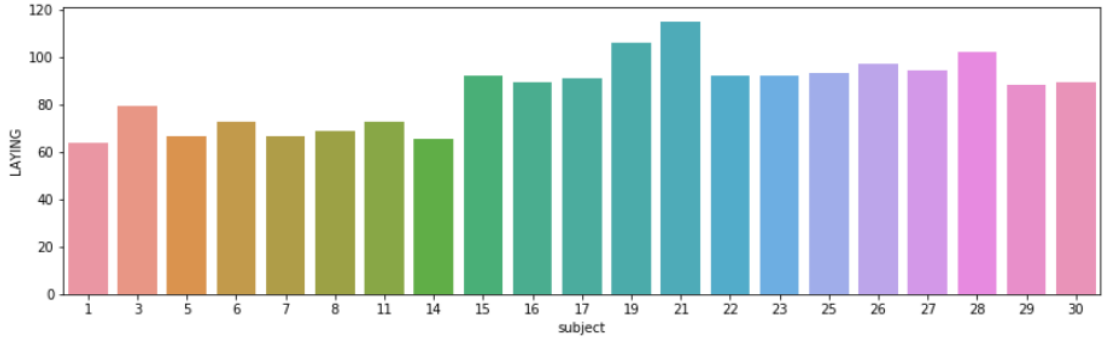
The activities can be divided into those that involve movement, called "*dynamic*," and those where the subject is stationary, called "*static*," then understand how the participants perform on these two activities.

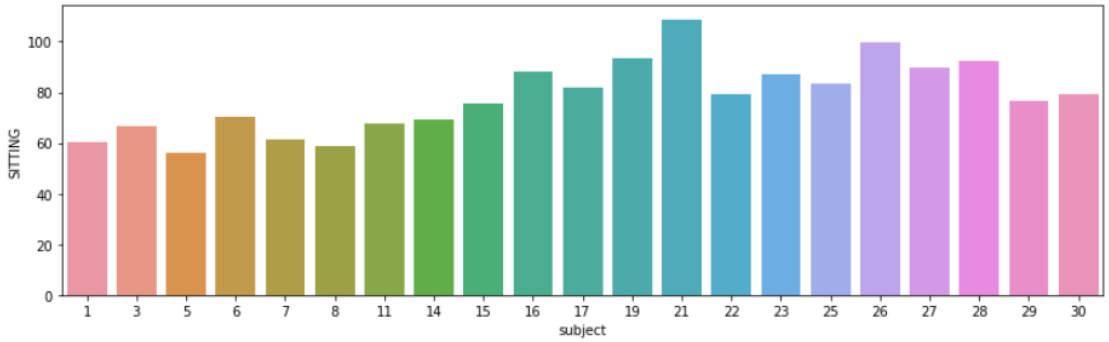
subject	LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS	active	passive
1	64.00	60.16	67.84	121.60	62.72	67.84	252.16	192.00
3	79.36	66.56	78.08	74.24	62.72	75.52	212.48	224.00
5	66.56	56.32	71.68	71.68	60.16	60.16	192.00	194.56
6	72.96	70.40	72.96	72.96	61.44	65.28	199.68	216.32
7	66.56	61.44	67.84	72.96	60.16	65.28	198.40	195.84
8	69.12	58.88	69.12	61.44	48.64	52.48	162.56	197.12
11	72.96	67.84	60.16	75.52	58.88	69.12	203.52	200.96
14	65.28	69.12	76.80	75.52	57.60	69.12	202.24	211.20
15	92.16	75.52	67.84	69.12	53.76	61.44	184.32	235.52
16	89.60	88.32	99.84	65.28	60.16	65.28	190.72	277.76
17	90.88	81.92	99.84	78.08	58.88	61.44	198.40	272.64
19	106.24	93.44	93.44	66.56	49.92	51.20	167.68	293.12
21	115.20	108.80	113.92	66.56	57.60	60.16	184.32	337.92
22	92.16	79.36	80.64	58.88	46.08	53.76	158.72	252.16
23	92.16	87.04	87.04	75.52	69.12	65.28	209.92	266.24
25	93.44	83.20	94.72	94.72	74.24	83.20	252.16	271.36
26	97.28	99.84	94.72	75.52	64.00	70.40	209.92	291.84
27	94.72	89.60	102.40	72.96	56.32	65.28	194.56	286.72
28	102.40	92.16	101.12	69.12	58.88	65.28	193.28	295.68
29	88.32	76.80	83.20	67.84	61.44	62.72	192.00	248.32
30	89.60	79.36	75.52	83.20	79.36	83.20	245.76	244.48

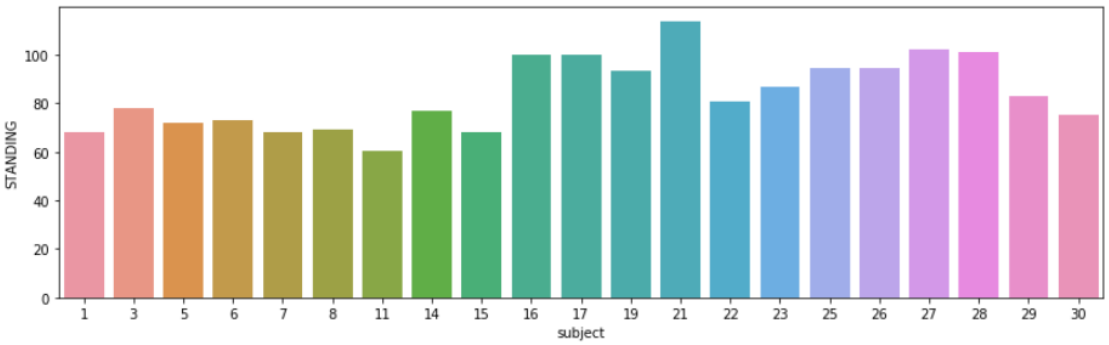


It can be observed that participant 19 and 21 are highly active.

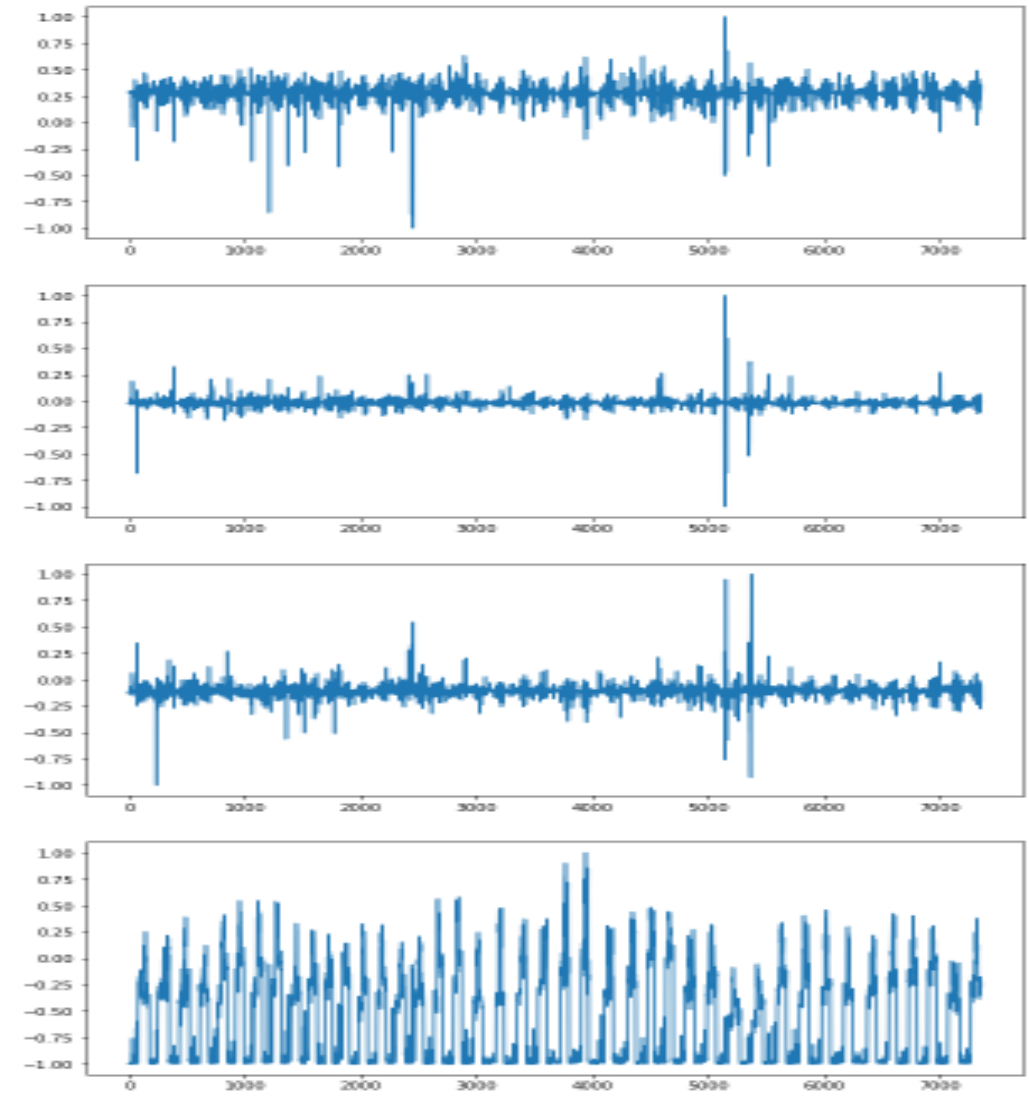
Q5 Number of hours spent by each participant on Laying/Sitting/Standing





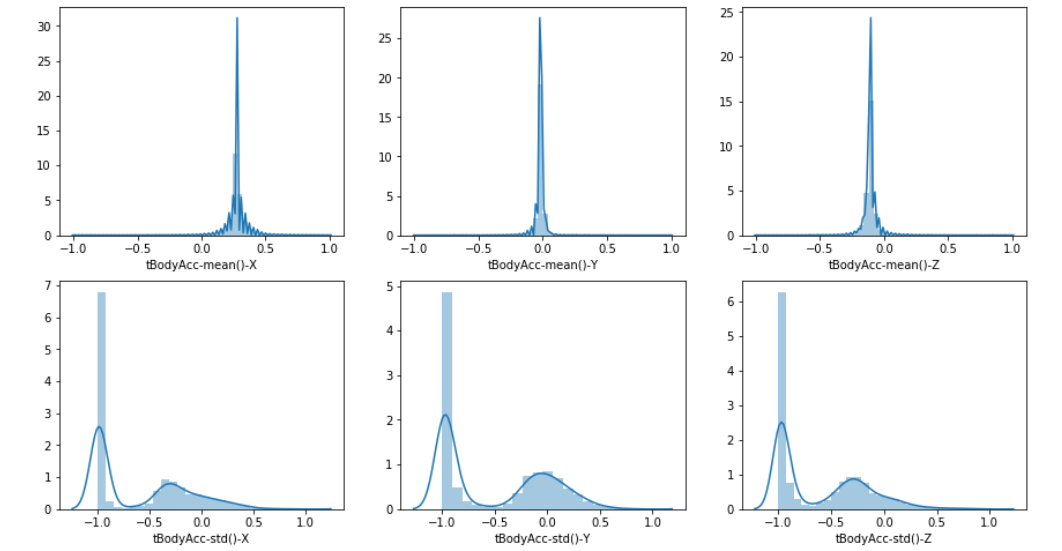


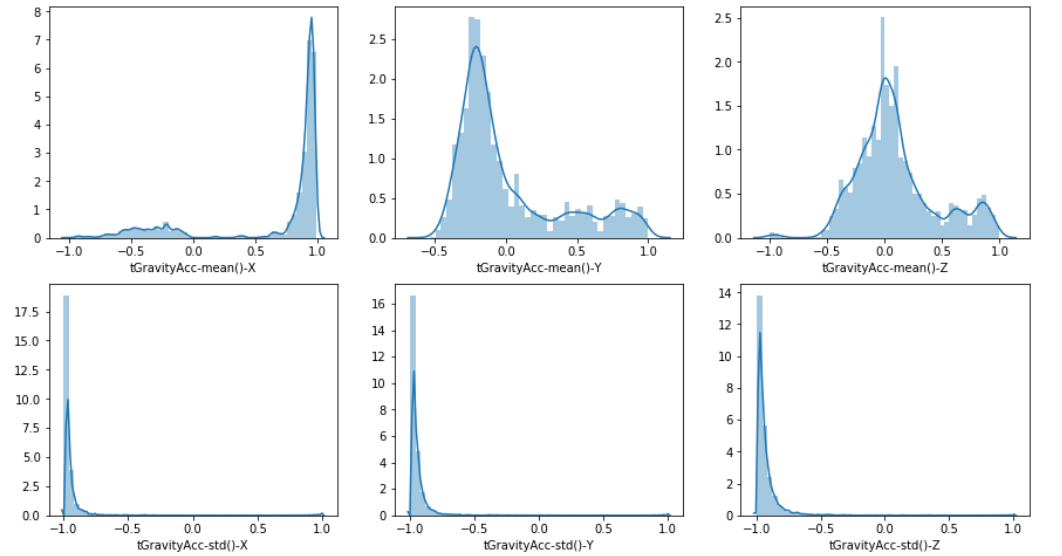
Q6 Visualize time series data of the training set containing 7352 rows



Q8 Distribution of Attribute

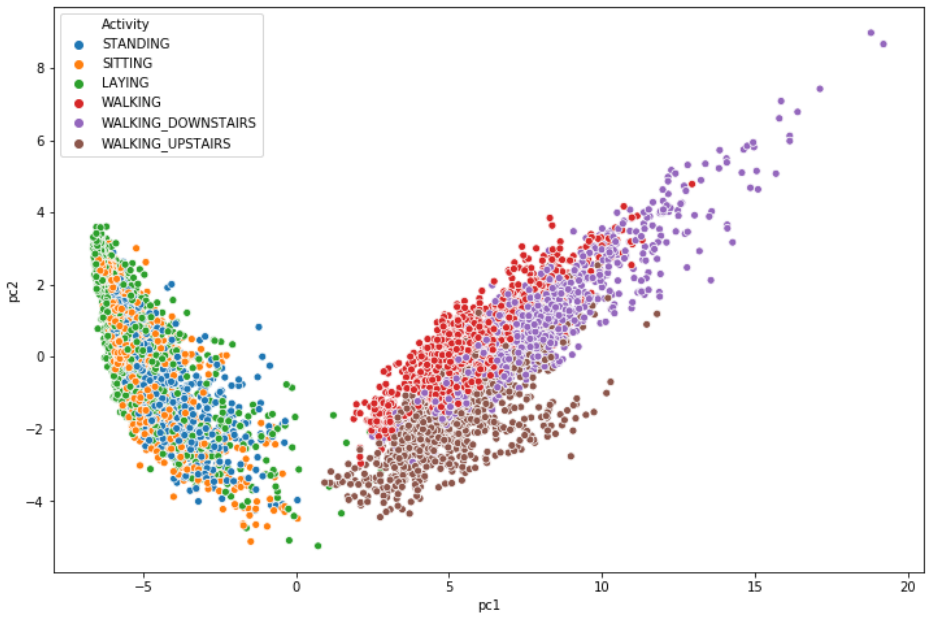
It is important to understand the distribution of the values in all the columns, as there are too many attributes, we cannot examine all of them. So, we could just get a glimpse of the distribution by looking at the first 6 columns of body acceleration data and 6 columns of gravity acceleration data





Principal Component Analysis (PCA)

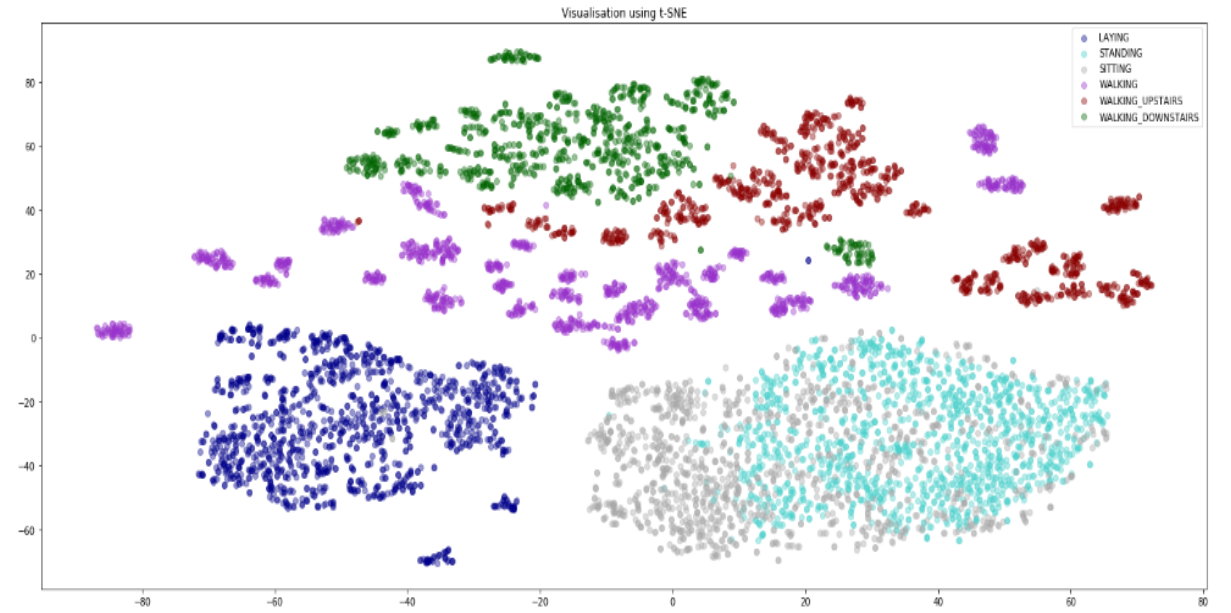
A natural next step is to use PCA to inspect the data further (Esbensen, 1998.). Important latent structures in the data can be interpreted through PCA. It consists of a direction in the predictor space which maximizes the variance of the samples. Each PC is normal to all the other PC's, such that there is no correlation between them.



As shown in the above plot, each of the six response classes is highlighted by different coloured dots. Clearly dynamic activities and static activities can be easily separable hence identifiable. The horizontal axis shows PC-1, the most dominant PC, which accounts for 51% of the total variation in the data set. Furthermore, in the PC-2 direction, containing 7% of the total variation, the three walking activities are further separated but with significant overlap. What majorly can be inferred is that, static activities i.e. Sitting, Standing and Laying are difficult to classify.

t-SNE

Just like PCA, we tried t-SNE, here we are mapping 561 dimensions to 2dimensions, from t-sne plot we can observe that most of the activities are well separated except the light-blue and grey points are overlapping, which means standing and sitting features are overlapping.



This analysis shows that it is reasonable to continue with classification, since there is clearly information in the predictors that correlate to the activity class, i.e. there is some structure in the data that can be used by algorithms to build classification

Model Development

XGBoost

As a first model, we tried XGBoost. Machine learning (ML) methods using decision trees include multiple trees in the form of an ensemble or forest (Johnson, 2013). Different methods train and combine trees in different ways. XGBoost has superior regularization and better handling of missing values, as well as much improved efficiency (Johnson, 2013). With XGBClassifier model is built and fitted on training data.

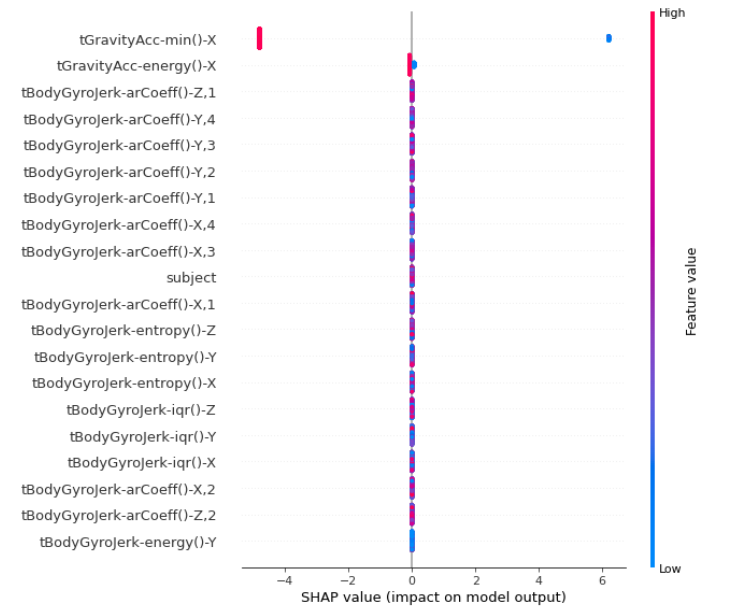
The testset must be independent of the training data in order to ensure realistic predictions of model performance. Below is the confusion matrix result of the initial model. For the model's better performance, explainable AI is further employed with hyper-parameter tuning.

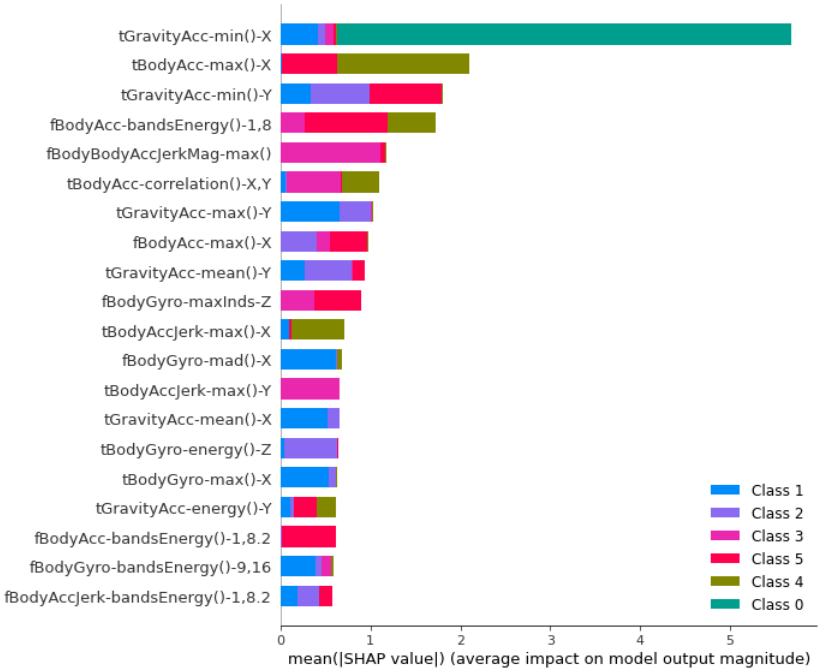
	STANDING	SITTING	LAYING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
STANDING	537	0	0	0	0	0
SITTING	0	411	77	0	0	3
LAYING	0	29	503	0	0	0
WALKING	0	0	0	487	5	4
WALKING_DOWNSTAIRS	0	0	0	10	384	26
WALKING_UPSTAIRS	0	0	0	30	5	436

	Precision	Recall	F-Score	Support
STANDING	1.000000	1.000000	1.000000	537
SITTING	0.934091	0.837067	0.882922	491
LAYING	0.867241	0.945489	0.904676	532
WALKING	0.924099	0.981855	0.952102	496
WALKING_DOWNSTAIRS	0.974619	0.914286	0.943489	420
WALKING_UPSTAIRS	0.929638	0.925690	0.927660	471

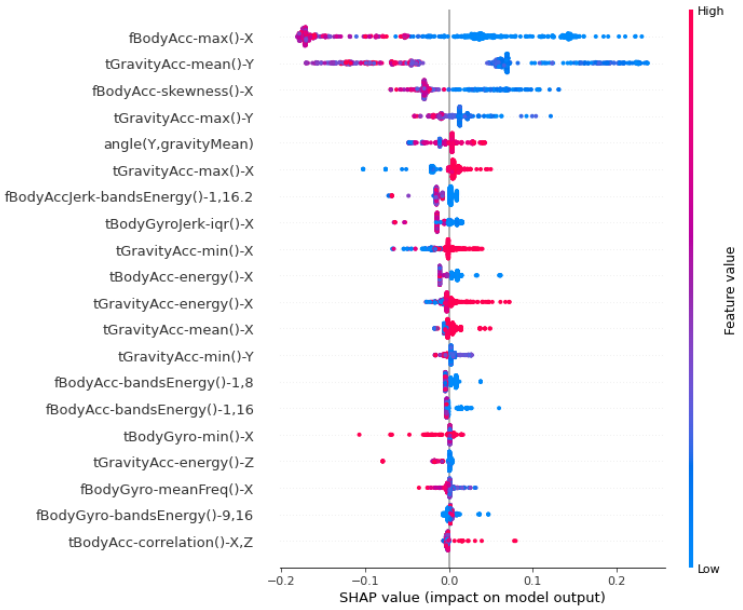
After obtaining these results, SHAP was used for explainable AI. SHAP stands for SHapley Additive exPlanations. SHAP values for each feature represent the change in the expected model prediction when conditioning on that feature. SHAP value explains the contribution to explain the difference between the average model prediction and the actual prediction of the instance for each feature.

```
shap.summary_plot(shap_values[0], X_test_new)
```

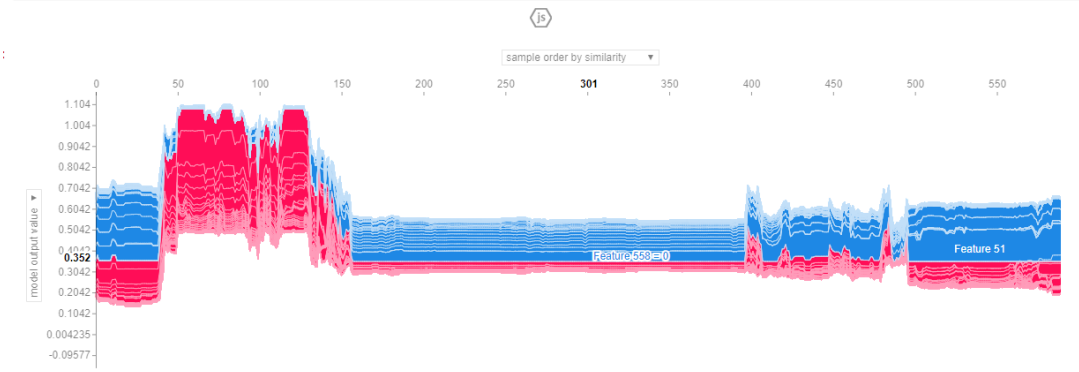




Global view of SHAP values for each feature: Below is the global view plot, this is very helpful view that shows the direction in which each feature contributes as compared to the average model prediction. Right side y-axis indicates the respective feature value being low vs high. Each dot represents 1 instance in the data, and hence we can observe cluster of dots indicating there are many instances in the data with that particular SHAP value.

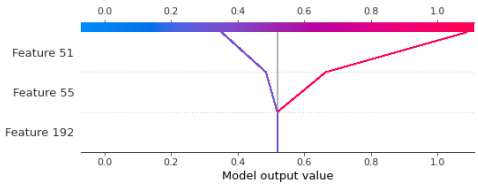


Individual instance view: Analysing the below individual instances view plot helps to identify the instances that are properly predicted and instances that are incorrectly predicted. This gives an idea of which features are causing wrong prediction. The base/average value is mentioned and each feature's contribution to the prediction is shown below. Features indicating red contribute to the prediction being higher than base thus causing a default, while features in blue contribute to the prediction being lower than base hence lean towards non-default.



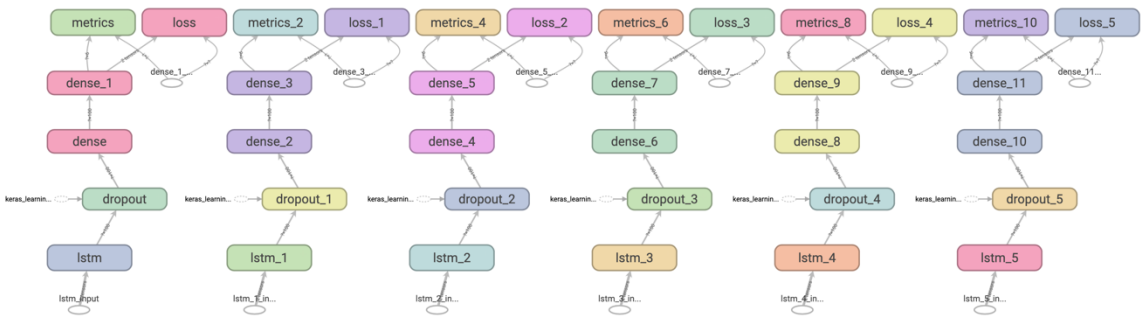
Below we can observe that feature importance, Feature 52 and 55 are the ones which contribute very heavily for class identification.

```
shap.decision_plot(explainer.expected_value[0], shap_values[0], feature_display_range=slice(None, -4, -1),ignore_warnings=True)
```



From sklear.multiclass onevsrestclassifier function is imported to fit to XGBClassifier model and then perform a grid search for hyper-parameter tuning. Model is then fitted with best parameters to obtain a confusion matrix.

LSTM



As mentioned earlier, LSTM network models can learn and remember over long sequences of input data. The model can support multiple parallel sequences of input data, in this case we have the accelerometer and gyroscope data. The model extracts features from sequences of observations and maps the internal features to different activities. The HAR dataset has three main signal types: 1) total acceleration 2) body acceleration, and 3) body gyroscope. Each has 3 axis's thus totalling up-to nine variables for each time step.

We initially load all data into a single 3D NumPy array, with array dimension [samples, time steps, features]. This means that there are 128-time steps and nine features, and the number of samples is the number of rows in the data. This can be implemented by the function load_group(). The dstack() NumPy function stacks the 3D arrays into a single 3D array. The load_dataset_group() function loads input and output data for a single group using the constant naming format.

We then load the train and test datasets. We must one hot encode the output data class integers to fit a neural network multi-class classification model. The function evaluate_model() fits a model on the training dataset, evaluates the model on the test dataset, and returns an estimate of the model's performance.

LSTM model is defined using the Keras deep learning library. The input for this model is a three-dimensional input of the form: [samples, time steps, features]. The output for the model will be a 6-element vector with the probability of a given window fitting to each of the 6-activity class.

Sequential Keras model is defined, with single LSTM hidden layer, followed by a dropout layer intended to reduce overfitting and a dense connected layer to understand the features extracted by the LSTM hidden layer and finally an output layer to make predictions. Adam optimizes the network, and the categorical cross entropy loss function is used as it is a multi-class classification problem.

```
model = Sequential()
model.add(LSTM(100, input_shape=(n_timesteps,n_features)))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(n_outputs, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=log_dir, histogram_freq=1, update_freq = 2000)
```

The model is fit for 15 epochs with a batch size of 64 samples i.e.; 64 windows of data will be exposed to the model, only then the weights are updated. Once the model is fit, it is evaluated on the test dataset and the accuracy of the fit model on the test dataset is returned.

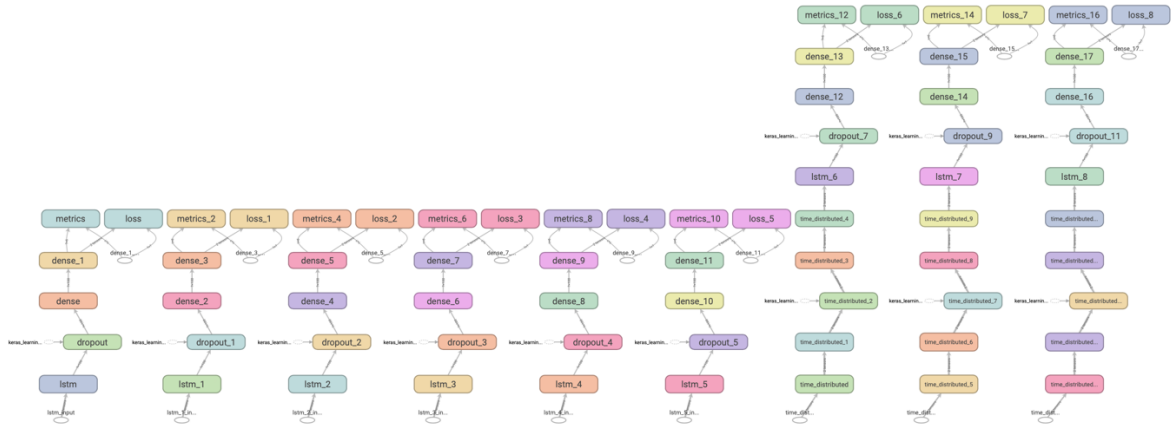
```
# fit and evaluate a model
def evaluate_model(trainX, trainy, testX, testy, model):
        verbose, epochs, batch_size = 1, 15, 64
        n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2], trainy.shape[1]
        # fit network
        model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size, verbose=verbose, callbacks=[tensorboard_callback])
        # evaluate model
        loss, accuracy = model.evaluate(testX, testy, batch_size=batch_size, verbose=0)
        return accuracy
```

We can finally summarize the model performance for each of the 10 runs before a final summary of the model's performance on the test set is reported.

Hyper-Parameter Tuning: From talos.model.normalizers lr_normalizer function is imported and used to normalize the dataset. Hyper-parameter tuning is performed on the LSTM model

```
from tensorflow.keras.optimizers import Adam, Nadam, RMSprop
from tensorflow.keras.losses import logcosh, categorical_crossentropy
from tensorflow.keras.activations import relu, elu, sigmoid
p = {'lr': (0.5, 5, 10),
     'batch_size': (2, 30, 10),
     'epochs': [15],
     'dropout': (0,0.5, 5),
     'optimizer': [Adam],
     'losses': [categorical_crossentropy],
     'activation':[relu, elu],
     'last_activation': [sigmoid]}
```

CNN LSTM



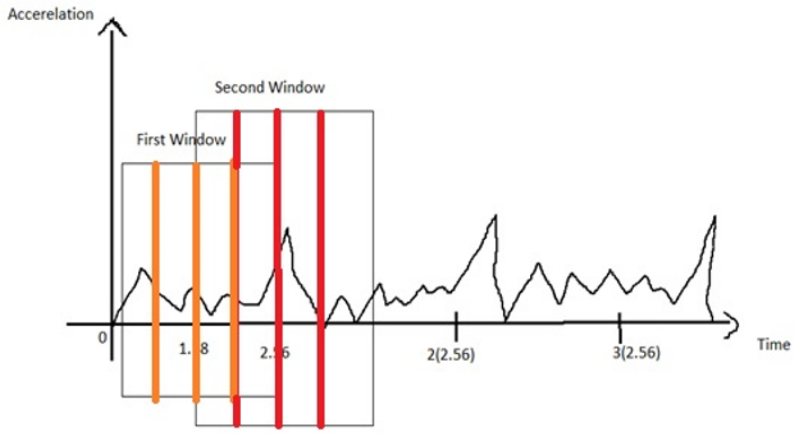
A simple CNN model for accelerometer data is developed, where each axis of the data is fed into separate convolutional layers, pooling layers, then concatenated before being interpreted by hidden fully connected layers.

Convolutional layers use kernel to read an input in small segments at a time and steps across the entire input field. Each result in the input that is projected onto a filter map and represents an internal interpretation of the input.

Pooling layers take the feature map projections and cleans them to the most essential elements. The convolution and pooling layers can be repeated at depth, providing multiple layers of abstraction of the input signals. The output layer consists of one or more fully connected layers that interpret the internal representation to a class value.

Conceptually there is a single CNN model and a sequence of LSTM models. For every input image, the CNN model is applied and passed on as an output to the LSTM as a single time step. This can be done by wrapping the entire CNN input model in a TimeDistributed layer. This can be simply described as, a model where the CNN learns a representation for a sub-sequence of observations, then the LSTM learns across these subsequences. The CNN LSTM reads subsequence's as blocks, extract features from them and finally allows the LSTM layer to interpret the extracted features.

Each window of 128-time steps is split into four subsequences of 32-time steps. Thus, this CNN model reads in sequences with a length of 32-time steps and 9-features.



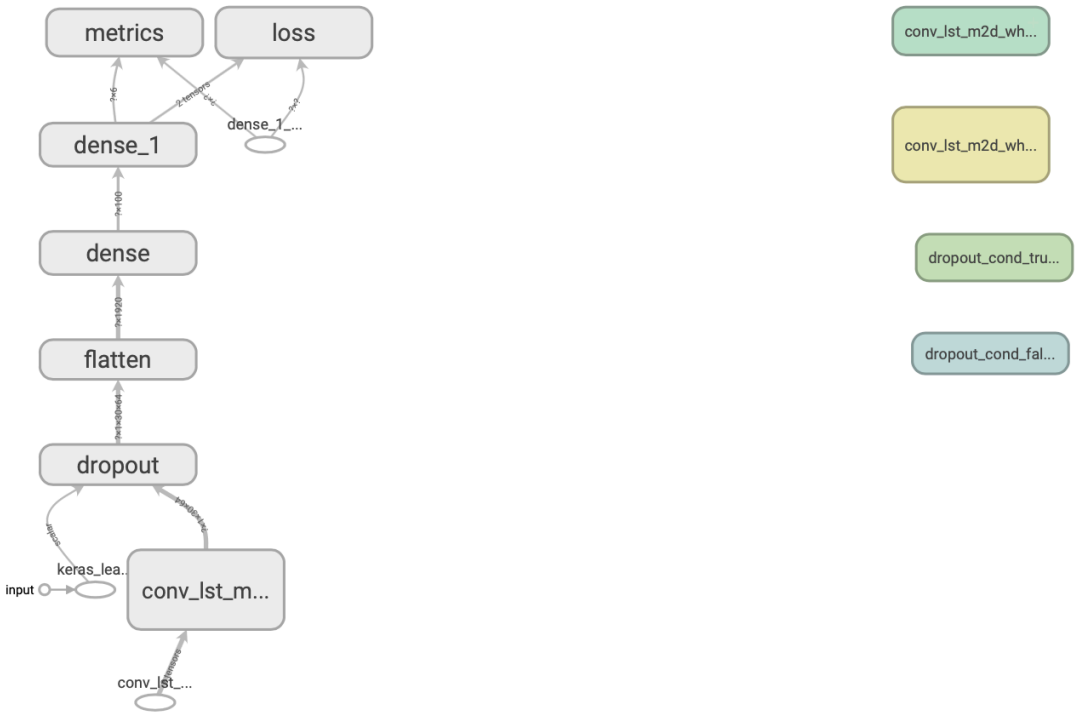
Dividing data into windows of size 2.56 sec

Features which were extracted will now be flattened and transferred to LSTM model, extracting its own features before a final mapping to an activity is made. All further steps of training and evaluating the model are same as previous.

```
# reshape data into time steps of sub-sequences
n_steps, n_length = 4, 32
# define model
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=3, activation='relu'), input_shape=(None,n_length,n_features)))
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=3, activation='relu')))
model.add(TimeDistributed(Dropout(0.5)))
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(100))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(n_outputs, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

We can finally evaluate the model performance for all the 10 runs, get an average of the 10 and draw a final summary of the model's performance on the test dataset.

CONVLSTM



The ConvoLSTM2D class is obtained from Keras library. In the ConvLSTM2D class, the input data is of the shape, where each time step is defined as an image of (rows * columns) data points. CNN-LSTM had divided the window of data (128-time steps) into four subsequences of 32-time steps. We can use this same subsequence approach in defining the ConvLSTM2D input where the number of time steps is the number of subsequences in the window, the number of rows is 1 and the number of columns is the number of time steps i.e. 32. Activation function ReLU is used, the output must be flattened into one long vector before it can be taken by a dense layer. Then the model will be evaluated.

```
# reshape into subsequences (samples, time steps, rows, cols, channels)
n_steps, n_length = 4, 32
# define model
model = Sequential()
model.add(ConvLSTM2D(filters=64, kernel_size=(1,3), activation='relu', input_shape=(n_steps, 1, n_length, n_features)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(n_outputs, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Just like other two models, we can finally evaluate the model performance for all the 10 runs, get an average of the 10 and draw a final summary of the model's performance on the test dataset.

Results & Conclusion

XGBoost

	STANDING	SITTING	LAYING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
STANDING	537	0	0	0	0	0
SITTING	0	411	77	0	0	3
LAYING	0	29	503	0	0	0
WALKING	0	0	0	487	5	4
WALKING_DOWNSTAIRS	0	0	0	10	384	26
WALKING_UPSTAIRS	0	0	0	30	5	436

	Precision	Recall	F-Score	Support
STANDING	1.000000	1.000000	1.000000	537
SITTING	0.824847	0.824847	0.824847	491
LAYING	0.844402	0.836466	0.840415	532
WALKING	0.897338	0.951613	0.923679	496
WALKING_DOWNSTAIRS	0.900000	0.878571	0.889157	420
WALKING_UPSTAIRS	0.892544	0.864119	0.878101	471

From the above confusion matrix, it can be observed that all the 537 points of standing are correctly classified, while only 411 sitting points are correctly classified. Among all the points that belongs to laying class, 503 points are classified correctly as laying and only 29 datapoints got misclassified as sitting class. Hence Standing has a precision, recall and F-score value of 1. While other classes are around 90%.

LSTM

The Accuracy of every run among the 10 repeats are listed below:

1. 90.261
2. 92.161
3. 92.229
4. 92.161
5. 91.890
6. 91.822
7. 91.720

8. 92.500
9. 91.381
10. 90.532

The summarized accuracy of the model is 91.666%.

Due to computational limitations, our hyperparameter tuning did not complete all the iterations. Only 22 iterations were completed with 15 epochs each. A best validation accuracy of 95.65% was obtained among these iterations.

CNN LSTM

The Accuracy of every run among the 10 repeats are listed below:

1. 91.788
2. 91.788
3. 91.856
4. 90.397
5. 91.923
6. 92.806
7. 93.315
8. 92.365
9. 93.315
10. 92.229

An average accuracy of 92.178% is observed. We can see that the model achieved a performance increase of about 1% from the LSTM average accuracy.

ConvLSTM

The Accuracy of every run among the 10 repeats are listed below:

1. 91.618
2. 90.091
3. 90.532
4. 91.618

5. 91.347
6. 91.991
7. 92.534
8. 91.957
9. 91.347
10. 91.788

An average accuracy of 91.483 is observed. It is similar to the LSTM model

The major advantage of using Deep Learning models is that we don't have to do any feature engineering. All the above models are ideal for comparison when all the models have hyperparameter tuning and the model is built from best parameters, but for this project it was not possible to achieve hyperparameter tuning due to computational limitations.

It can be concluded that the neural networks are almost always capable to correctly identify the human activity. It can be observed that there is difficulty in differentiating walking, walking upstairs and walking downstairs, probably due to similar movements in these activities.

HAR has numerous applications. For example, now during Coronavirus pandemic, HAR models can be used to predict the movements of a covid-19 positive patient during quarantining. If any walking activity is observed for a long duration, administration can be alerted of patient's movement thus enabling them to further restrict the quarantine violation. HAR models can be used for monitoring of elderly people in need.

References

D. Anguita, A. G.-O. (2012). Human Activity Recognition on Smartphones Using a Multiclass Hardware-Friendly Support. *Ambient Assisted Living and Home Care: 4th International Vitoria-Gasteiz,* (pp. 216-226). Spain: J. Bravo, R. Hervás, and M. Rodríguez, Eds., ed Berlin,.

Esbensen, K. (1998.). Multivariate analysis in practice. *Oslo: CAMO*.

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting. *The Annals of Statistics, vol. 29*, 189-1232.

Guestrin, T. C. (2016). XGBoost: A Scalable Tree Boosting System. *KDD*.

Johnson, M. K. (2013). Applied Predictive Modeling. *Springer*.