CNN-LSTM Model for Human Activity Recognition

A PROJECT REPORT

Submitted by

Moukhik Misra - 19BCE2190
Mohit Suhasaria - 19BCE2167

Course Code: CSE 3013
Course Title: Artificial Intelligence

Under the guidance of
Dr. S. Anto
Associate Professor, SCOPE,
VIT , Vellore.

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

December, 2021

CNN-LSTM Model for Human Activity Recognition

Moukhik Misra – 19BCE2190 , Mohit Suhasaria – 19BCE2167

Abstract– **Time series data is analysed using various statistical models to forecast datapoints and trends. A multivariate time series has variables sharing dependencies with other parameters in the dataset. Generally, approaches like Convolutional Neural Networks (CNN), Vector Auto Regressive Moving Averages (VARMA), Long Short Term Memory (LSTM) are suitable for multivariate time series analysis, however each comes with specific drawbacks associated with the models. In this paper, we aim to develop and implement a hybrid CNN-LSTM model to overcome the drawbacks encountered while using each of these models individually by combining their advantages together. CNN allows us to extract features having high impact in the dataset while LSTM enables us to find interdependencies in the multivariate time series data. The UCI HAR Dataset having six activity labels is used for implementing the model and suitable performance metrics are computed to determine its accuracy.**

Index Terms– Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long Short Term Memory (LSTM), CNN-LSTM Hybrid.

I. INTRODUCTION

Multivariate time series data can be analysed using many mathematical models such as VARMA, CNN, LSTM and others to predict upcoming datapoints based on its trends. Each of the above mentioned models have their sets of pros and cons such as; the automatic feature extraction capability of CNNs without explicit feature description, or the ability of LSTMs to maintain context information for undecided periods and process temporal data efficiently but simply applying one of these models fails to capture all the characteristics of a dataset in a relatively simple manner to reach optimal conclusions through iterative parameter optimisation. A multivariate time series has multiple variables sharing dependencies with other parameters in the dataset, that is, the association between features and datapoints is highly interdependent. The forecast is only as reliable as the accuracy obtained from the model so it becomes necessary to optimise each model to its highest accuracy. The said models may be suitable for using in specific conditions but each has its own set of drawbacks which can't be overcome by optimisation alone as it is rooted in the statistics used as the basis of these models. Convolutional Neural Networks are exceptionally advantageous when extracting features of high impact from the dataset. Similarly, Long Short Term Memory model is ideal for the classification and prediction of time series data. We intend to use the unique advantages of both these models and combine them together into a hybrid CNN-LSTM model having higher overall performance metrics when compared to each of these models independently. For implementing our model, we will use the Human Activity Recognition dataset from the UCI database. The dataset contains six activity labels; standing, walking, sitting, lying down, walking upstairs and walking downstairs. The overall accuracy of the model will be increased by optimising various parameters and compared to that of existing models.

II. LITERATURE SURVEY

A) TIME SERIES CLASSIFICATION WITH MULTIVARIATE CONVOLUTIONAL NEURAL NETWORK

The authors of this paper emphasise on the ability of deep learning architectures to consolidate feature learning and classification in a single model and propose to extend its capabilities even further by developing a multivariate convolutional neural network architecture using a tensor based approach. They have used the PHM or Prognostics and Health Management dataset to compare the performance of their model with existing literature. The standard performance metric used to evaluate the PHM dataset forecasts is the prediction score. An in depth analysis apart from performance metric computation is done to provide a wholistic view of their model. The results show that the model developed by them is good at learning features from the given dataset and hence it is good at feature extraction from raw data. Their model has an accuracy of 97.7%. Additionally, They suggest that visualising the extraction of features may be a very insightful approach to feature learning.

B) A HYBRID CNN–LSTM NETWORK FOR THE CLASSIFICATION OF HUMAN ACTIVITIES BASED ON MICRO-DOPPLER RADAR

In this paper, the authors explain the existing methodology employed for human activity recognition using deep learning models for radar based raw data. The general approach used is to convert this raw data to a 2-dimensional spectrogram. To do so, a short-time Fourier transform or STFT is used. This spectrogram is usually processed as an optical image which can be processed through 2-dimensional deep learning models such as 2-D Convolutional neural Networks. Even though these models are able to produce a large number of feature maps from the spectrogram, these networks fail to simplify the relationships between the parameters and are likely to produce overly complex networks. These complex networks still aren't able to predict data points accurately as this approach neglects the temporal characteristics of the dataset. The novelty of the above mentioned paper is to approach the same radar based data as a multi channel time series. The authors implement this by designing a deep learning model with 1-D CNNs and LSTM to make a hybrid model. Their results found that this hybrid approach is superior to the existing methods in terms of accuracy of prediction and recognition by considering both spatial and temporal characteristics of the data. They were able to achieve a peak accuracy of 98.65% by adjusting the model's hyper parameters.

C) LSTM-CNN ARCHITECTURE FOR HUMAN ACTIVITY RECOGNITION

The authors of this paper consider traditional models used for pattern recognition which need manually engineered features for learning which are unsuitable for generalisation of the learning process of the model. These models have been outgrown by the applications of deep learning algorithms for pattern recognition in data obtained from wearable sensors. They aim to develop a hybrid model using CNN and LSTM enabling the model to classify activities by automatic feature extraction and parameter updation. They use LSTM for temporal characteristic recognition in their model as it is a form of Recurrent Neural Network. They used data obtained from wearable mobile sensors in the raw form as input to two layers of Long Short Term Memory networks. This was succeeded by convolutional neural network layers. Instead of using a fully connected layer, they replaced it with an average pooling layer that considers global averages for downsampling of nodes to reduce model parameters after convolutional layers. They enhanced the efficiency of their processing by adding batch normalisation. Their paper covered three datasets; UCI HAR, WISDM and OPPORTUNITY. The accuracy obtained for UCI HAR Dataset was 95.78% showing that their model has superior activity recognition capabilities as compared to traditional deep learning approaches while generating less number of features.

D) TSE-CNN: A TWO-STAGE END-TO-END CNN FOR HUMAN ACTIVITY RECOGNITION

Here, the authors focus on the healthcare applications of human activity recognition using deep learning algorithms for monitoring various activities associated with elderly supervision, exercise monitoring and observation of rehabilitation progress. They suggest that human activity recognition from inertial data provided by wearable sensors is more robust to external noise factors, allowing the models better input data for recognition. The authors point out the major drawback of this approach as relatively low recognition accuracy for complex human activity patterns. They have also considered a few other drawbacks of the said approach as; lack of large sets of training data from test subjects in realistic scenarios, the high computational requirements and power consumption costs associated with it. The authors then move on to propose a solution for the same by designing an end-to-end two stage CNN along with data augmentation techniques. They attempt to reduce the computational complexity by minimising the number of sensors used for obtaining training data to a single accelerometer. This would not only reduce the hardware costs but it also significantly cuts down the volume of data being processed by the model. Their results show a decreased complexity in computation and higher accuracy of prediction with an accuracy score of 95.7%.

E) A NEURAL NETWORKS BASED METHOD FOR MULTIVARIATE TIME-SERIES FORECASTING

In this paper, the authors focus on multivariate time series analysis using neural networks as the basis of their model. Neural networks have shown to be significantly more accurate than other auto regressive models but so far, even these models often fail to consider the various characteristics of the input data. To properly capture the impact of long term as well as short term trends in the data, the authors propose the use of deep learning algorithms using CNNs and RNNs to predict time series data. They have also added autoregressive features in their model to incorporate proportionate output for inputs of different scales. The input of the model are constructed based on period characteristics; long term and short term historical data. The datasets used in their paper include; Traffic, Solar-Energy, Electricity and Exchange-Rate. They successfully implemented this algorithm to achieve higher accuracies of prediction.

F) FEATURE LEARNING FOR HUMAN ACTIVITY RECOGNITION USING CONVOLUTIONAL NEURAL NETWORKS

The authors discuss the use of CNNs as a prevalent tool for designing models that have the ability to extract features, specifically in the application of Human Activity Recognition. The major advantage Convolutional neural networks have over traditional machine learning algorithms is that they can extract features from input data on their own and hence do not require the need for manual engineering of meaningful parameters by intensive technical study of the subject. However, appropriate training of the convolutional networks is required for suitable output making cold-start a relevant problem. The authors present a case study of using pre-trained CNNs for evaluating accuracy of classification of human activities through feature extraction in realistic conditions. They have assessed different models with various parameters to find the most suitable fit for human activity recognition to obtain a pre-trained CNN feature extractor in the first stage of their case study. The selected model is then tested in a real world scenario using large scale dataset in the second stage. For human activity recognition using inertial signal readings, they used UCI HAR Dataset and for audio data based human activity recognition they used the DCASE 2017 dataset to obtain accuracies as 91.98% and 92.30 percent respectively.

III. METHODOLOGY

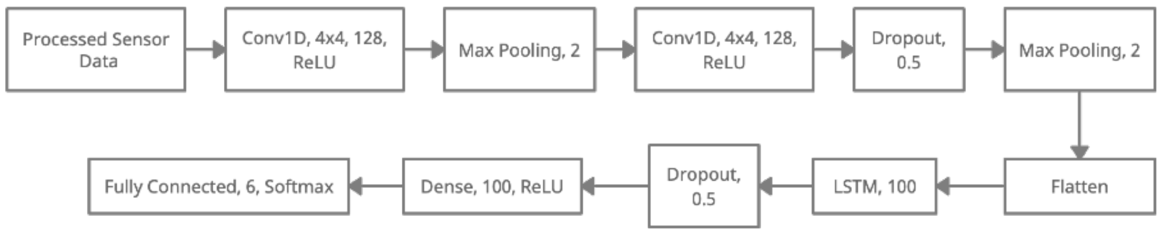
We start by exploring the UCI HAR Dataset which contains the inertial signal data of thirty test subjects performing any of the six activities labeled as; walking, sitting, laying, walking upstairs, standing and walking downstairs. The data used was measured through the triaxial gyroscope and triaxial accelerometer of a smartphone attached to the waist of each subject. The accelerometer reading was further resolved into total acceleration and body acceleration by separating the gravitational acceleration component. The dataset has a fixed width of separation and there are 128 readings in each of these windows. The dataset was split into 70-30 for train and test purposes respectively, that is there are readings from 21 subjects in the training set and 9 subject readings are used in the testing set. For each row of data, the label for the subject performing the activity as well as the activity associated with that row is given. Since the data is separated into nine different readings; total acceleration, body acceleration and body gyroscope for each of the three axes; x, y and z, they have been merged into a single file to produce the train and test datasets used by the model.

The Convolutional Neural Network layer is the first layer of our model. It is a deep learning algorithm that takes input data and convolves it with filters to extract various features from it. Then different weights and biases are assigned to each of these features so that the model can distinguish them from each other. The convolution involves the filter sliding over each element of the input matrix, computing the product of corresponding elements of the input matrix that coincide with the elements of the filter and finally obtaining the sum of these values to be stored in the resultant matrix. The output of a CNN layer is smaller than the size of its input. The number of filters applied in a CNN layer is associated with the number of feature matrices generated by that layer. These features matrices are passed on to the next layers for further processing. The same process can be applied in our case where the input data is one dimensional; that is the input matrix to this layer would be 1-D. By using CNN our model can extract features from the raw time-series data directly without the need for manual feature definition.

The activation function for the CNN layer is used for transforming the weighted sum for a given node into the activation for output. The activation function is a non-linear transformation that allows the model to incorporate complex relationships in the data. We use the Rectified Linear Activation Function (ReL) which appears to be a linear function but is in fact a non-linear transformation. ReLU also allows to train the model with back propagation of errors, that is the weights of the model are adjusted based on the errors obtained in previous iterations or epochs. ReLU simply takes the input value and returns the same value if it is non-negative and it returns zero if the input is negative.

Dropout layers are used to prevent overfitting at various stages of the model. A dropout layer ensures this by processing input nodes with a given probability and removing nodes with probability [1- given probability]. The dropout layer also eliminates the edges associated with the nodes being dropped. This is performed during each epoch to regularise the interdependencies between nodes in the model.

The output feature matrices obtained from the CNN layer are sensitive to the position of the features in input data. The feature matrices can be made more resistant to changes in the location of the features in the input data by down sampling the feature matrices. This can be achieved by adding a pooling layer to the model in places where downsampling is required. Pooling works by summarising the presence of features in blocks of the feature matrix. There are two approaches to pooling; average



(Fig. 1. LSTM-CNN Model Architecture)

pooling and max pooling. We use max pooling in our model which calculates the largest value in each block of each feature matrix. The resultant matrices are down sampled feature matrices containing the most dominant features within each block.

After obtaining the pooled feature matrices from the previous layers, we add a flatten layer to merge all these matrices into a single column vector by adding the values of each feature map row by row. This single column vector is used as the input for artificial neural networks like LSTM.

Long Short Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) with the ability to learn sequential dependencies in time series data. All RNNs store an internal state representing context information about past inputs for a certain period depending upon the weights associated with these inputs. This means that all RNNs can store context information for an undecided period, they are robust against noise and the parameters guiding the model are trainable. They contain modules from previous iterations that feed context to upcoming time steps. LSTM is a type of RNN which is resistant to the variations in lags between time steps. LSTMs achieve this ability by maintaining Constant Error Carrousels inside special cells. Each of these memory cells has two gate units that open and close the access to the contents inside the cell, where one gate is responsible for protecting the context from perturbations in input data and the second gate protects other cells from irrelevant context. These cells and units constitute memory blocks that form the structure of the LSTM network. The LSTM argument determines the dimensionality of the output data.

We use the output from the LSTM layer as the input for the dense layer, which is also known as the fully connected layer. Every node outputted by the previous layer is used as the input for this layer. This layer performs matrix vector multiplication on its inputs where the row vector must have the same number of columns as the column vector. The resultant matrix gives the values for the trained parameters which can be adjusted through back propagation.

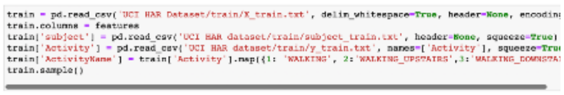
The Softmax function is used for activation in the final layer of the model which converts the input vector containing numbers to proportional relative to the magnitude of each value in the input vector. It is used to normalise the output weighted sums to probabilities that add up to one giving each class a certain probability out of the total classes defined in the classification problem.

For optimising the model, Adam optimisation algorithm is used for iterative updation of network weights over given epochs. Instead of the classical optimisation techniques that maintain a single learning rate for all network weights, Adam maintains different learning rates for each weight parameter. The learning rates are adapted based on both; the mean and the uncentered variance. The moving averages for the gradients and squared gradients are calculated by Adam optimiser and the parameters controlling the decay rates of these moving averages is set accordingly. For further optimisation a loss function is used based on which the model decides the learning parameters and adjusts them with the goal of minimising the loss function. We have used categorical cross entropy as our loss function since we have multiple output labels. The output labels are one-hot encoded and if the output label is present in integer format, it is converted to categorical encoding.

The performance metrics used to evaluate the model are; accuracy, precision, recall and F1-score. The accuracy will provide the percentage of accurately predicted values from the test set, that is the number of correct predictions divided by total forecasts. Precision of the model is given by the relevant examples or true positives out of all examples predicted within a given class. The third metric being used is recall which tells us the ratio of the test cases predicted to be in a given class to the total number of cases that truly fall inside that class. The final metric is the weighted average of precision and recall incorporating false positives as well as false negatives, giving us the F1-score.

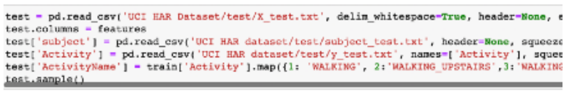
IV. IMPLEMENTATION

The model is implemented using Jupyter Notebook and python along with a few libraries designed for python; numpy, pandas, seaborn, Keras, tensorflow, scikit-learn and Matplotlib.



```
train = pd.read_csv('UCI HAR Dataset/train/X_train.txt', delim_whitespace=True, header=None, encodin
train.columns = features
train['subject'] = pd.read_csv('UCI HAR dataset/train/subject_train.txt', header=None, squeeze=True)
train['Activity'] = pd.read_csv('UCI HAR dataset/train/y_train.txt', names=['Activity'], squeeze=Tru
train['ActivityName'] = train['Activity'].map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTA
train.sample()
```

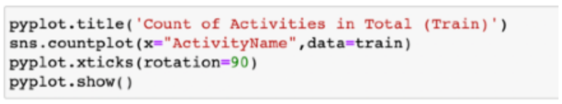
(Fig. 2. Loading Train Data)



```
test = pd.read_csv('UCI HAR Dataset/test/X_test.txt', delim_whitespace=True, header=None, e
test.columns = features
test['subject'] = pd.read_csv('UCI HAR dataset/test/subject_test.txt', header=None, squeeze
test['Activity'] = pd.read_csv('UCI HAR dataset/test/y_test.txt', names=['Activity'], squee
test['ActivityName'] = train['Activity'].map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING
test.sample()
```

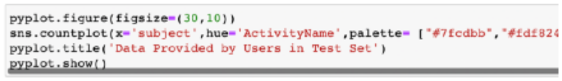
(Fig. 3. Loading Test Data)

We start by loading all the necessary data into memory. The train data is stored into three files; the processed inertial readings, the subject labels and the activity labels. So, these files are loaded into memory and organised into pandas data frames. The activity labels are represented as numerical values so they are mapped to their physical labels for ease of classification. Similarly, the test data is also read and arranged inside the data frames and all the data loaded is checked for duplicate values or null instances.



```
pyplot.title('Count of Activities in Total (Train)')
sns.countplot(x="ActivityName",data=train)
pyplot.xticks(rotation=90)
pyplot.show()
```

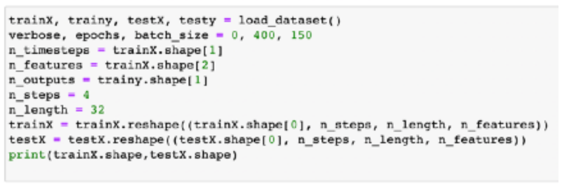
(Fig. 4. Count Plot of Activities in Training Set)



```
pyplot.figure(figsize=(30,10))
sns.countplot(x='subject',hue='ActivityName',palette= ["#7fcdbb","#fdf824
pyplot.title('Data Provided by Users in Test Set')
pyplot.show()
```

(Fig. 5. Count Plot of Activities per user in Test Set)

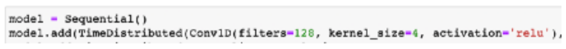
Some basic data exploration is performed so that we can observe; the distribution of count of cases falling inside each class of activity and the count of activities performed by each subject by plotting necessary histograms using Matplotlib and seaborn.



```
trainX, trainy, testX, testy = load_dataset()
verbose, epochs, batch_size = 0, 400, 150
n_timesteps = trainX.shape[1]
n_features = trainX.shape[2]
n_outputs = trainy.shape[1]
n_steps = 4
n_length = 32
trainX = trainX.reshape((trainX.shape[0], n_steps, n_length, n_features))
testX = testX.reshape((testX.shape[0], n_steps, n_length, n_features))
print(trainX.shape,testX.shape)
```

(Fig. 6. Reading Inertial Signals, Activity Labels and Subject Labels)

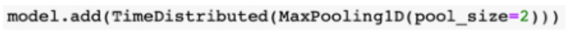
In the next step, we read the inertial signals for both train and test data from the nine files; total acceleration in al three axes, body acceleration in all three axes, and body gyroscope readings for all three axes for both train and test set. This data is appended for both sets and converted to categorical. Similarly, the corresponding labels stored in separate files are loaded and converted to categorical data. This means we end up with two 3-dimensional arrays of train data and test data and two 2-dimensional arrays of train labels and test labels. The parameters required by the model are set to initial values and the hyper parameters of the model are fixed.



```
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=128, kernel_size=4, activation='relu'),
```

(Fig. 7. Sequential Model, First 1-D Convolutional layer)

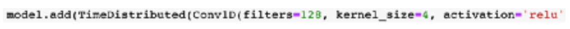
Now we can start preparing the model using the sequential function from Keras. The first layer is the convolutional layer. Since our input matrix is one dimensional, that is, time series we use a 1-dimensional convolutional network. We have used the ReLU activation approach as it allows the model to update parameters through back propagation of errors. The ReLU activation outputs a non-negative value and transforms the input matrix in a non-linear way. We have used 128 filters in the CNN layer and a kernel size of 4.



```
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
```

(Fig. 8. Max Pooling layer)

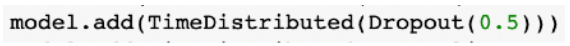
The next layer is a pooling layer to down sample the feature matrices to the most dominant features in each block of each feature map so we have added a max pooling layer with a pool size of 2.



```
model.add(TimeDistributed(Conv1D(filters=128, kernel_size=4, activation='relu'
```

(Fig. 9. Second 1-D Convolutional layer)

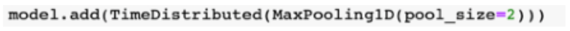
The pooled nodes are convolved again with a 1-dimensional CNN layer for more relevant feature extraction with 128 filters, kernel size 4 and ReLU activation.



```
model.add(TimeDistributed(Dropout(0.5)))
```

(Fig. 10. First Dropout layer)

To prevent overfitting and to regularise the samples, a dropout layer is added with a probability of processing nodes set to 0.5.



```
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
```

(Fig. 11. Second Max Pooling layer)

This is followed by another max pooling layer with pool size 2.

```
model.add(TimeDistributed(Flatten()))
model.add(LSTM(100))
```

(Fig. 12. Flatten and LSTM layers)

Since the next stage after feature extraction, down sampling and regularisation is to incorporate the time series in the model using LSTM, we must flatten the input feature maps because the LSTM takes a column vector as input. The dimensionality of the output space for LSTM layer is set to 100.

```
model.add(Dropout(0.5))
```

(Fig. 13. Second Dropout Layer)

We add another dropout layer with node processing probability of 0.5 to prevent overfitting of data.

```
model.add(Dense(100, activation='relu'))
```

(Fig. 14. First Dense Layer)

The model has two dense layers. The first dense layer is fully connected to the LSTM output neurone, that is 100 connections. This layer is activated by the ReLU activation function as well.

```
model.add(Dense(n_outputs, activation='softmax'))
```

(Fig. 15. Second Dense Layer)

The final layer of the model is a dense layer activated by the softmax function with the same number of neurons as that of the number of training cases.

```
history=model.fit(trainX, trainy, validation_data=(testX, testy), epochs=epochs,
loss , accuracy = model.evaluate(testX, testy, batch_size=batch_size, verbose=0)
```

(Fig. 16. Model Fit)

The model is compiled with categorical cross entropy loss function and Adam optimiser. The performance metrics used for evaluation are; accuracy, precision, recall and F1-score.

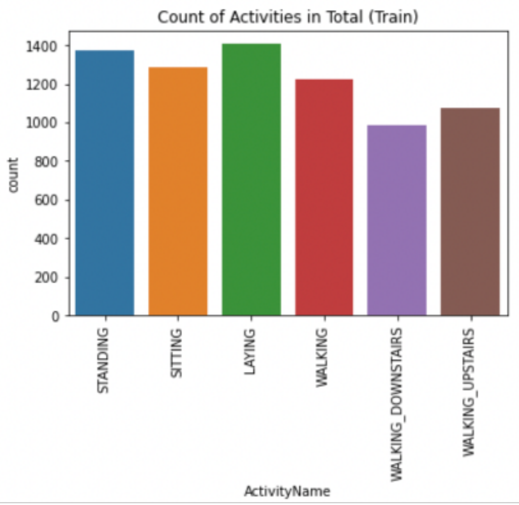
Once the model is prepared, we fit it to the training data over set batch size, verbose and epochs for the total iterations for model parameter updating. The performance metrics are obtained by passing the test data to the model.

```
matrix = metrics.confusion_matrix(max_ytest, max_ypred_test)
plt.figure(figsize=(6, 4))
sns.heatmap(matrix, cmap="YlGnBu", xticklabels=LABELS, ytickl
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```

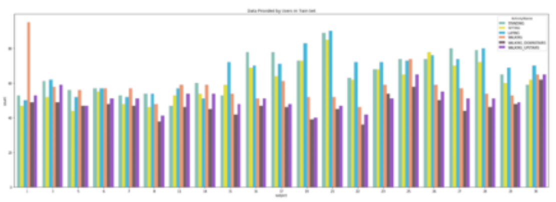
(Fig. 17. Confusion Matrix Generation)

We use all the predictions made by the model to generate its confusion matrix for the true labels and predicted labels.

V. RESULTS



(Fig. 18. Activity Count for Train Set)



(Fig. 19. Activity Count per Subject for Train Set)

We observed an even distribution of activities performed by the test subjects in the UCI HAR Dataset so it is not skewed in favour of a particular activity label.

```
(7352, 128, 9) (7352, 6) (2947, 128, 9) (2947, 6)
(7352, 4, 32, 9) (2947, 4, 32, 9)
```

(Fig. 20. Input Matrix Dimensions)

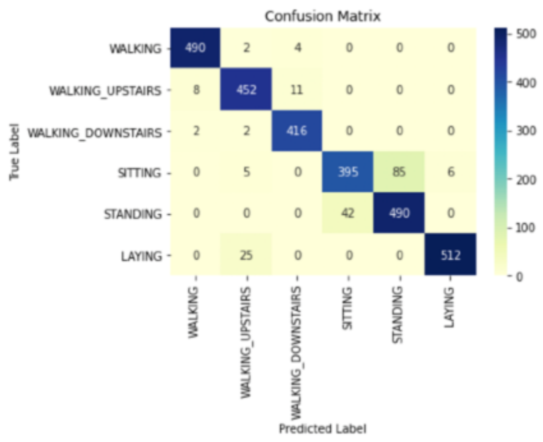
The 3-dimensional matrix containing the inertial signals from triaxial accelerometer and gyroscope for training and testing data was made and its dimensions were displayed.

```
Accuracy = 93.48490238189697
```

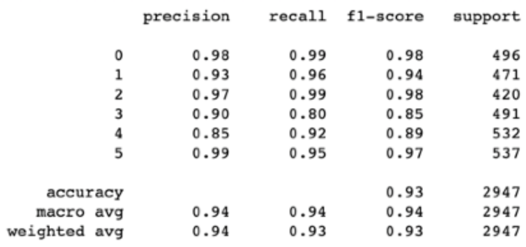
(Fig. 21. Accuracy of the Model)

The model was created, fit to the training set and the accuracy for the test set was obtained as 93.48%.

The confusion matrix hence formed from the predictions made by the model over all activity classes shows a large number of true positives for all activities but it can be observed that the model has a relatively lower accuracy while classifying



(Fig. 22. Confusion Matrix)



```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       496
           1       0.93      0.96      0.94       471
           2       0.97      0.99      0.98       420
           3       0.90      0.80      0.85       491
           4       0.85      0.92      0.89       532
           5       0.99      0.95      0.97       537

    accuracy                           0.93      2947
   macro avg       0.94      0.94      0.94      2947
weighted avg       0.94      0.93      0.93      2947
```

(Fig. 23. Performance Metrics)

activities under sitting and standing classes, but the overall accuracy of classification still remains high.

We made a comparison between our model and the models from other literature based on the accuracy metric and a higher accuracy in our model can be seen.

Human Activity Recognition – UCI HAR	CNN_LSTM (Mutegeki and Han)	CNN_LSTM_Dense (Mutegeki and Han)
Accuracy %	92.13%	91.55%

LSTM (Mutegeki and Han)	LSTM_Dense (Mutegeki and Han)	LSTM (T. Varma et al)
91.28%	91.40%	88%

CNN (T. Varma et al)	CNN-LSTM (T. Varma et al)	Fundamental CNN (K. Wang et al)
93%	88%	93.16%

CNN (K. Wang et al)	CNN (F. Cruciani et al)	CNN-LSTM (S. Deep and X. Zheng)
93.21%	91.98%	93.40%

LSTM (S. Deep and X. Zheng)	Res-LSTM (S. Deep and X. Zheng)	Baseline LSTM (S. Deep and X. Zheng)
92.98%	91.60%	90.80%

Bi-directional LSTM (F. Hernández et al)	Proposed Accuracy CNN - LSTM
92.67%	93.48%

VI. CONCLUSION

In this paper, we designed and implemented a deep learning model using 1-dimensional Convolutional Neural Networks for feature extraction and feature mapping, and LSTM model for time series analysis of the dataset, hence making a CNN-LSTM hybrid deep learning model for human activity recognition. We used the UCI HAR Dataset with six activity labels and successfully obtained a good accuracy score for classifying data into these labels. Furthermore, we optimised the model by testing it over a wide range of parameters like, batch size, kernel size and epochs. The best performance of the model was for a kernel size of 4 over 400 epochs, with an accuracy of 93.48%. These results have been compared with similar models and datasets from existing literature which conclusively shows higher accuracy for our model with given hyper parameters.

VII. REFERENCES

[1] C. L. Liu, W. H. Hsaio, and Y. C. Tu, “Time Series Classification with Multivariate Convolutional Neural Network,” IEEE Trans. Ind. Electron., vol. 66, no. 6, pp. 4788–4797, 2019, doi: 10.1109/TIE.2018.2864702.

[2] J. Zhu, H. Chen, and W. Ye, “A Hybrid CNN-LSTM Network for the Classification of Human Activities Based on Micro-Doppler Radar,” IEEE Access, vol. 8, pp. 24713–24720, 2020, doi: 10.1109/ACCESS.2020.2971064.

[3] K. Xia, J. Huang, and H. Wang, “LSTM-CNN Architecture for Human Activity Recognition,” IEEE Access, vol. 8, pp. 56855–56866, 2020, doi: 10.1109/ACCESS.2020.2982225.

[4] J. Huang, S. Lin, N. Wang, G. Dai, Y. Xie, and J. Zhou, “TSE-CNN: A Two-Stage End-to-End CNN for Human Activity Recognition,” IEEE J. Biomed. Heal. Informatics, vol. 24, no. 1, pp. 292–299, 2020, doi: 10.1109/JBHI.2019.2909688.

[5] S. Li, H. Huang, and W. Lu, “A Neural Networks Based Method for Multivariate Time-Series Forecasting,” IEEE Access, vol. 9, pp. 63915–63924, 2021, doi: 10.1109/ACCESS.2021.3075063.

[6] F. Cruciani et al., “Feature learning for Human Activity Recognition using Convolutional Neural Networks: A case study for Inertial Measurement Unit and audio data,” CCF Trans. Pervasive Comput. Interact., vol. 2, no. 1, pp. 18–32, 2020, doi: 10.1007/s42486-020-00026-2.

[7] R. Mutegeki and D. S. Han, “A CNN-LSTM Approach to Human Activity Recognition,” 2020 Int. Conf. Artif. Intell. Inf. Commun. ICAIIC 2020, pp. 362–366, 2020, doi: 10.1109/ICAIIC48513.2020.9065078.

[8] T. Varma, S. John, S. Joy, and J. James, “Real Time Human Activity Recognition : Smartphone Sensor based - using Deep Learning,” no. May, pp. 3772–3778, 2021.

[9] K. Wang, J. He, and L. Zhang, “Attention-based convolutional neural network for weakly labeled human activities’ recognition with wearable sensors,” IEEE Sens. J., vol. 19, no. 17, pp. 7598–7604, 2019, doi: 10.1109/JSEN.2019.2917225.

[10] S. Deep and X. Zheng, “Hybrid Model Featuring CNN and LSTM Architecture for Human Activity Recognition on Smartphone Sensor Data,” Proc. - 2019 20th Int. Conf. Parallel Distrib. Comput. Appl. Technol. PDCAT 2019, pp. 259–264, 2019, doi: 10.1109/PDCAT46702.2019.00055.

[11] F. Hernández, L. F. Suárez, J. Villamizar, and M. Altuve, “Human Activity Recognition on Smartphones Using a Bidirectional LSTM Network,” 2019 22nd Symp. Image, Signal Process. Artif. Vision, STSIVA 2019 - Conf. Proc., 2019, doi: 10.1109/STSIVA.2019.8730249.

VIII. APPENDIX

A) OBJECTIVES

The main objective of our project was to identify existing deep learning models for human activity recognition, recognise their advantages and drawbacks, and design a more effective model. We aimed to design a Hybrid LSTM-CNN model to classify human activities with a higher accuracy than similar models as seen in existing literature using the UCI HAR Dataset.

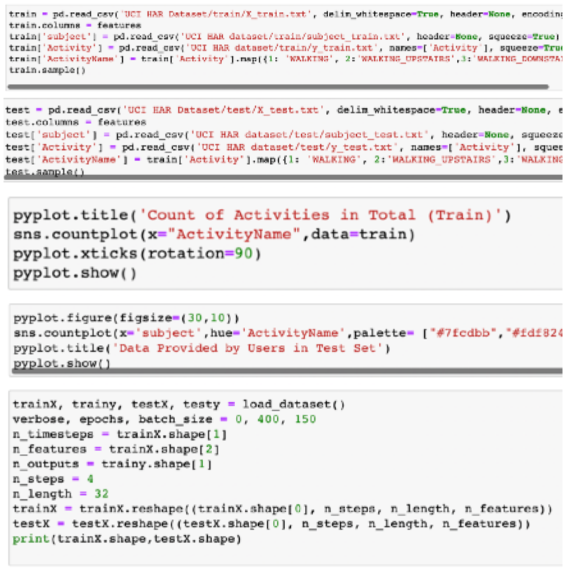
B) SOFTWARE AND HARDWARE USED

The code was executed on Jupyter Notebook using python kernel. Various libraries developed for python were used which include Numpy, pandas, Keras, Tensorflow, Matplotlib and seaborn.

All these software components are compatible with multiple operating systems but for our purposes we have tested the model on Windows 10 and MacOS 12.0.1 with slight modifications in the import method and environment setup.

The hardware for Windows used includes an i5 10th generation quad core intel processor with 16GB RAM whereas the hardware used with MacOS has ARM based 8-core Apple Silicon M1 processor, however the tensorflow library also makes use of the 7-core GPU in this case.

C) SOURCE CODE



```
train = pd.read_csv('UCI HAR Dataset/train/X_train.txt', delim_whitespace=True, header=None, encodin
train.columns = features
train['subject'] = pd.read_csv('UCI HAR dataset/train/subject_train.txt', header=None, squeeze=True)
train['Activity'] = pd.read_csv('UCI HAR dataset/train/y_train.txt', names=['Activity'], squeeze=Tru
train['ActivityName'] = train['Activity'].map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTA
train.sample()
```

```
test = pd.read_csv('UCI HAR Dataset/test/X_test.txt', delim_whitespace=True, header=None, e
test.columns = features
test['subject'] = pd.read_csv('UCI HAR dataset/test/subject_test.txt', header=None, squeeze
test['Activity'] = pd.read_csv('UCI HAR dataset/test/y_test.txt', names=['Activity'], squee
test['ActivityName'] = train['Activity'].map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING
test.sample()
```

```
pyplot.title('Count of Activities in Total (Train)')
sns.countplot(x="ActivityName",data=train)
pyplot.xticks(rotation=90)
pyplot.show()
```

```
pyplot.figure(figsize=(30,10))
sns.countplot(x='subject',hue='ActivityName',palette= ["#7fcdbb","#fdf824
pyplot.title('Data Provided by Users in Test Set')
pyplot.show()
```

```
trainX, trainy, testX, testy = load_dataset()
verbose, epochs, batch_size = 0, 400, 150
n_timesteps = trainX.shape[1]
n_features = trainX.shape[2]
n_outputs = trainy.shape[1]
n_steps = 4
n_length = 32
trainX = trainX.reshape((trainX.shape[0], n_steps, n_length, n_features))
testX = testX.reshape((testX.shape[0], n_steps, n_length, n_features))
print(trainX.shape,testX.shape)
```

```
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=128, kernel_size=4, activation='relu'),
```

```
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
```

```
model.add(TimeDistributed(Conv1D(filters=128, kernel_size=4, 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'))
```

```
history=model.fit(trainX, trainy, validation_data=(testX, testy), epochs=epochs,
loss , accuracy = model.evaluate(testX, testy, batch_size=batch_size, verbose=0)
```

```
matrix = metrics.confusion_matrix(max_ytest, max_ypred_test)
plt.figure(figsize=(6, 4))
sns.heatmap(matrix, cmap="YlGnBu", xticklabels=LABELS, ytickl
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```