HUMAN ACTIVITY RECOGNITION

By Alhiet Orbegoso & Raghavi Suryadevara

1. INTRODUCTION

Health is considered as the most important aspect of a human being. One way to optimize health is to know and recognize the habitual behavior of a human being and modify lifestyle accordingly. Human Activity recognition is a study which recognizes the behavior of human beings using the information provided by the mobile and classify human activity. With the help of sensors, accelerometer and gyroscope, and machine learning techniques, we predict the activity.

2. MOTIVATION

Mobile / Smart phone contain several sensors such as accelerometer, gyroscope, magnetometer, proximity sensor, etc., Accelerometer and gyroscope are two important sensors which aid in predicting the human activity. Here, the accelerometer measures acceleration while the gyroscope measures angular velocity. These two features play a major role in determining the human activity into one of the 6 categories: walking, standing, sitting, walking up, walking down and laying.

The accelerometer and gyroscope measure the 3-axial data, measured with respect to the time. The 6-time series data, 3 each for accelerometer and gyroscope, is considered as the input. This is fed to our model. Based on this, the data can be classified into one of the 6 categories walking, standing, sitting, walking up, walking down and laying. This is clearly an example of multi class classification. In multi-class classification, with the help of accuracy matrix, confusion matrix and measuring multi class log-loss, we can classify the activity into one of the six categories.

3. DATABASE

3.1 Data Collection

The dataset is sourced from UCI (University of California, Irvine) machine learning repository. The data is recorded from 30 people, each with a smart phone, performing different activities. A video was recorded while performing these tasks to manually label the data. Data is then processed using signal processing techniques to obtain the dataset that we used here.

The data set can be downloaded from the link. (Provided at the end of the report)

Below is a list of files that can be found in the original data set

- Train & test folders containing the information regarding the data for modeling (the first 70% is the train data and the rest 30% is the test data).

- A readme.txt file containing details regarding the dataset and contents of unzipped files.

- A features.txt file containing technical description of the features engineered in the experiment.

Important elements to be noted from the train folder:

- Inertial signals folder contains the preprocessed data.

- X_train.txt file contains engineered features to fit a model.

- Y_train.txt file contains the class labels for each observation, starting from 1 through 6.

- The subject_train.txt file that contains mapping of each line in the data files with subject identifier, from 1 through 30.

Remark: It is important to mention that the original database was reconstructed to perform current preprocessing explained in the next sections. Original database can be downloaded from UCI repository, but database used for this project can be presented if required.

	body_acc_x	body_acc_y	body_acc_z	body_gyro_x	body_gyro_y	body_gyro_z	total_acc_x	total_acc_y	total_acc_z	Activity
0	0.000181	0.010767	0.055561	0.030191	0.066014	0.022859	1.012817	-0.123217	0.102934	5
1	0.010139	0.006579	0.055125	0.043711	0.042699	0.010316	1.022833	-0.126876	0.105687	5
2	0.009276	0.008929	0.048405	0.035688	0.074850	0.013250	1.022028	-0.124004	0.102102	5
3	0.005066	0.007489	0.049775	0.040402	0.057320	0.017751	1.017877	-0.124928	0.106553	5
4	0.010810	0.006141	0.043013	0.047097	0.052343	0.002553	1.023680	-0.125767	0.102814	5
...	...	...	...	...	...	...	...	...	...	...
684731	-0.061667	-0.175584	0.151117	0.148295	-0.015923	0.109040	0.908386	-0.423054	-0.092933	2
684732	-0.070890	-0.145071	0.181814	0.143136	-0.024389	0.006547	0.898984	-0.392272	-0.063138	2
684733	-0.050755	-0.104717	0.173271	0.095931	-0.021024	-0.051342	0.918862	-0.351680	-0.072539	2
684734	-0.019807	-0.020764	0.195638	0.090708	-0.041893	-0.078877	0.949475	-0.267526	-0.050975	2
684735	-0.011040	0.052439	0.218432	0.055943	-0.102402	-0.046268	0.957835	-0.194160	-0.028925	2

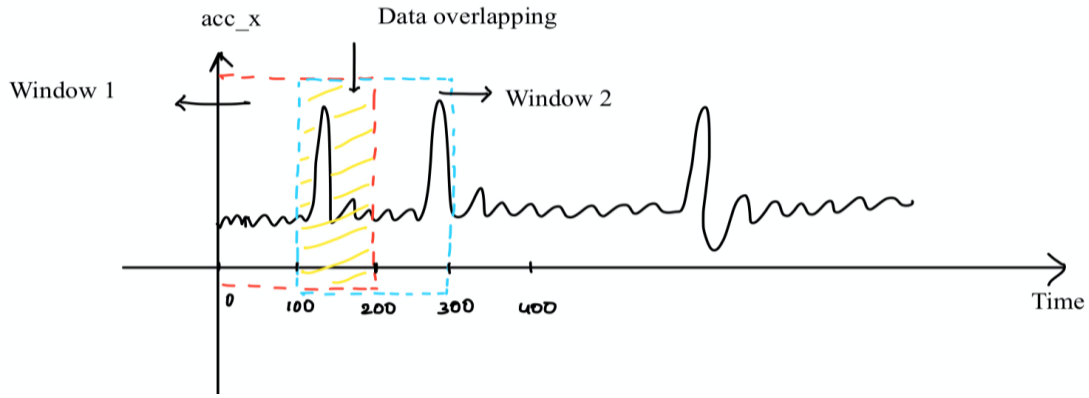
684736 rows × 10 columns

Main database reconstructed

3.2 Data Preprocessing

From the database, it is recategorized all the time series data by category (activity). Then a window is selected in which the size is a parameter of the experiment. The size is defined by the number of elements in the window, for example a value of size 100 means that a window has 100 elements (time series of all sensors). In addition, there is an offset parameter, which defines how many overlapping elements will be presented for window. For example, if offset is set to 50 means that the next window will start at the element 50 of the previous window.

Here, we used 200 number of elements with an overlap 100 elements. This implies that we have 100 elements in each window with an additional 100 overlapping elements common to two adjacent windows.



Window size and overlapping

A feature vector from each window is obtained by calculating variables from time and frequency domains. This process is repeated for each window for the 9-time series signals to obtain data. Finally, data generated is divided in training and testing data. Training data is used to fit neural network and testing data is used as validation.

For the window size and overlapping proposed, are generated from database 8614 data points for training and 3691 data points for testing.

```
# Length of data verification #elements = #datatrain + #datatest + #dataval
total_data = len(datatrain)+len(datatest)
print("Length of datatrain:",len(datatrain),"(",round(len(datatrain)/total_data*100,3),"%)")
print("Length of datatest:",len(datatest),"(",round(len(datatest)/total_data*100,3),"%)")
print("Total length:",len(datatrain)+len(datatest))
```

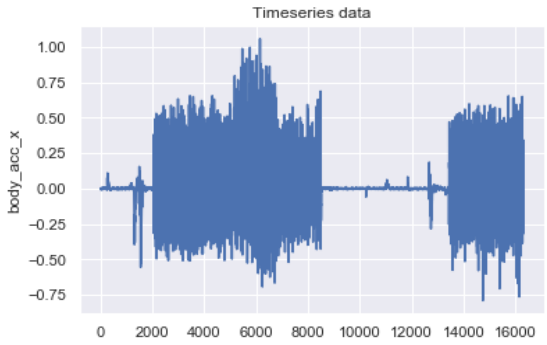
```
Length of datatrain: 8614 ( 70.004 %)
Length of datatest: 3691 ( 29.996 %)
Total length: 12305
```

Data points generated

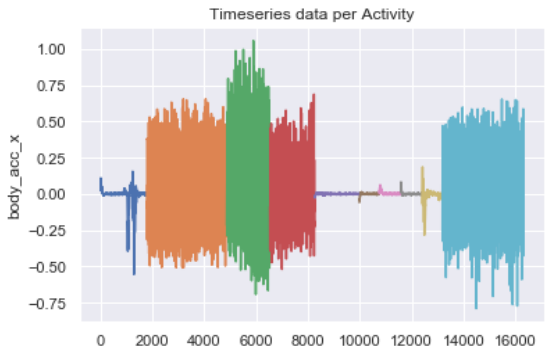
It is important to verify the uniformity of the data. A histogram is presented of the data points generated per activity. If some activity has relative lower frequency in training or testing data, could not be completely learned (training) or verified (testing). In this case database has almost uniform distributed activities.

Activities Histogram Train/Test

The main objective is to convert unlabeled data from sensor to labeled data in which is possible to visualize and classify the activities being executed by interval of time.



Unlabeled data



Labeled data

4. EXPERIMENTAL SETUP

This section presents how the database information and neural network are used to solve the clustering problem. It is provided information on data setup and architecture of neural network.

4.1 Input/Output Data

Each data point is represented by a 2-dimensional matrix of size [MxN] where M is the length of the window timeseries and N is the number of lectures. Moreover, each datapoint is related to one label (activity). In summary a matrix [MxN] is labeled to only 1 activity.

Datapoint Label

Timeseries [200] $\begin{bmatrix} x_{t0}^1 & x_{t0}^2 & \dots & x_{t0}^8 & x_{t0}^9 \\ x_{t1}^1 & x_{t1}^2 & \dots & x_{t1}^8 & x_{t1}^9 \\ x_{t2}^1 & x_{t2}^2 & \dots & x_{t2}^8 & x_{t2}^9 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{t200}^1 & x_{t200}^2 & \dots & x_{t200}^8 & x_{t200}^9 \end{bmatrix} \rightarrow [y_1]$

Lectures [9]

For this configuration there are 8614 datapoints (each containing the [200x9] matrices and its respective label) for training and other 3691 for testing. Before using this configuration in the neural network, it is mandatory to convert the label information to the one hot format, given that neural network will solve a classification problem.

Label One-Hot format

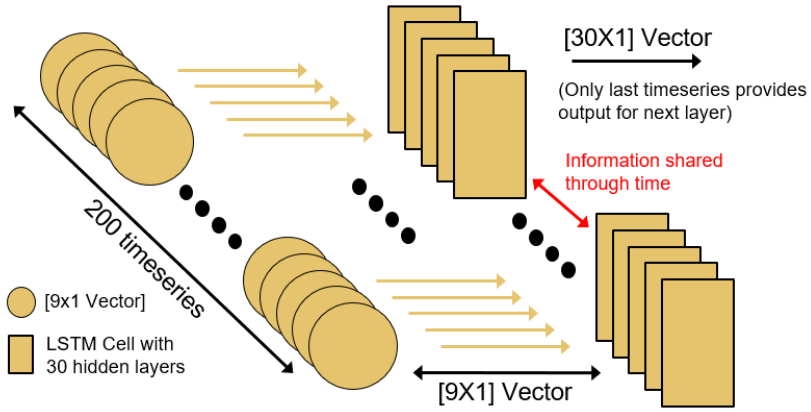
$\begin{bmatrix} y_1 \\ y_3 \\ y_5 \\ \vdots \\ y_2 \end{bmatrix}$ $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

One-hot format it is a binary vector in which all are zeros expect the position which correspond to the label number.

4.2 Neural Network Structure

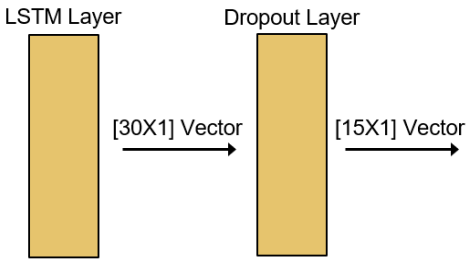
The neural network is implemented in python using the package tensor flow 2.0 and the Keras library. The neural network is composed of 3 layers explained below:

- LSTM layer: This layer is core element in the neural network. The LSTM can recognize patterns in timeseries data. The LSTM has 30 hidden layers. The input is the datapoint ([200x9] matrix) and the output is a [30x1] size vector.



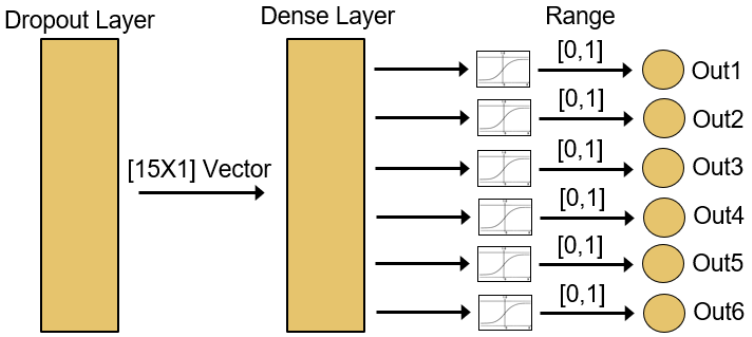
LSTM Layer

- Dropout layer: This layer will randomly "shutdown" 50% of the nodes. This is useful to avoid overfitting.



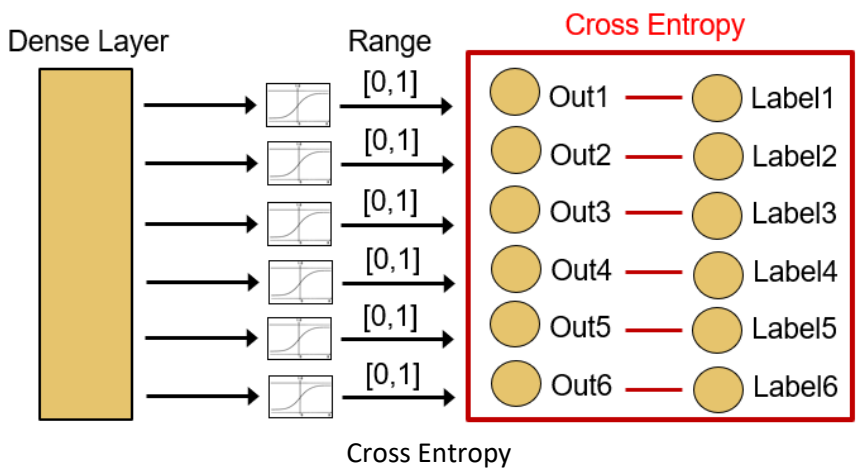
Dropout Layer

- Dense Layer: This last layer will collapse the dropout vector to a [6x1] vector which has the same length as activities labels. Function used as activation is sigmoid, it ensures that output range is between 0 and 1.



Dense Layer

The structure described is used to classify the time series data. As loss function is used the cross entropy between neural network output and the label from training data.



The error from cross entropy calculation is used for backpropagation of neural network. The neural network is optimized using RMS-prop gradient.

Below are detailed the main parameters of the neural network:

- LSTM hidden layers: 30
- Dropout level: 50%
- Dense Layer: 6
- Activation function: Sigmoid
- Loss Function: Cross Entropy
- Optimization function: RMS-prop gradient
- Learning rate: 0.001
- Epochs: 30

The neural network presented has in total 4986 trainable parameters.

5 RESULTS

Given the number of parameters that need to be tuned for neural network performance, this project will only focus on LSTM hidden layers, window elements and overlapping. Below is presented different results varying these parameters:

- Hidden Layers: 15, Window length: 200, Overlapping: 50

Cross Entropy

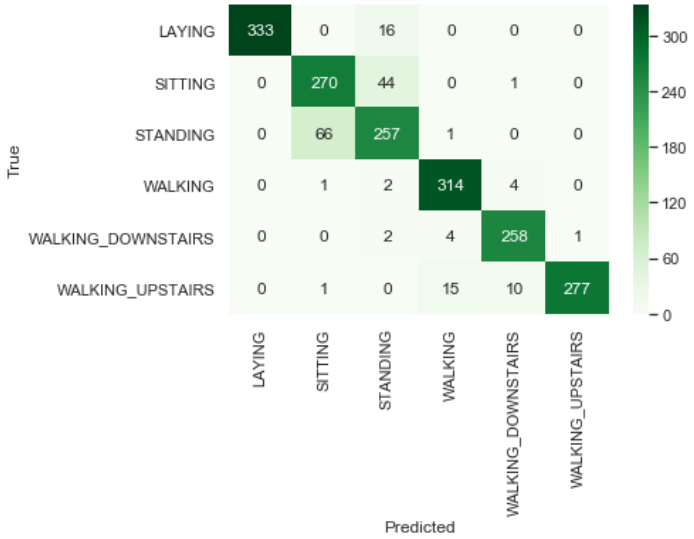
Accuracy

Confusion Matrix

- Hidden Layers: 15, Window length: 200, Overlapping: 100

Error

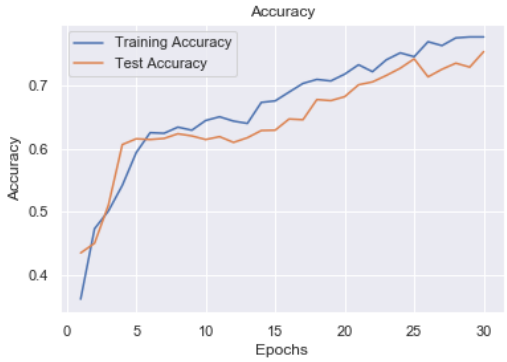
Accuracy



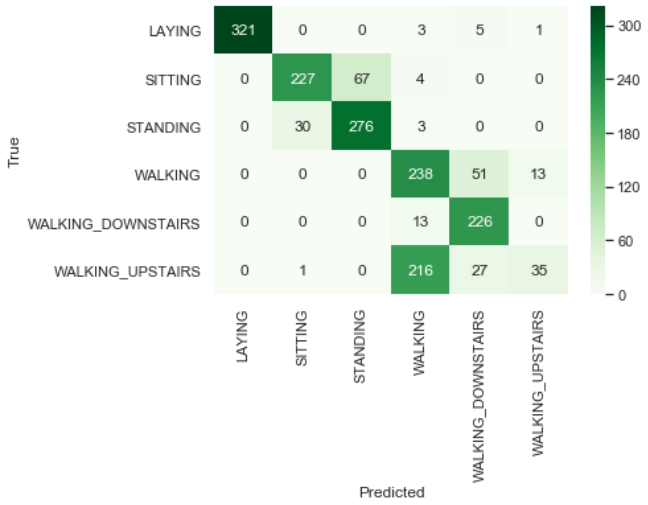
Confusion Matrix

- Hidden Layers: 30, Window length: 300, Overlapping: 100

Error



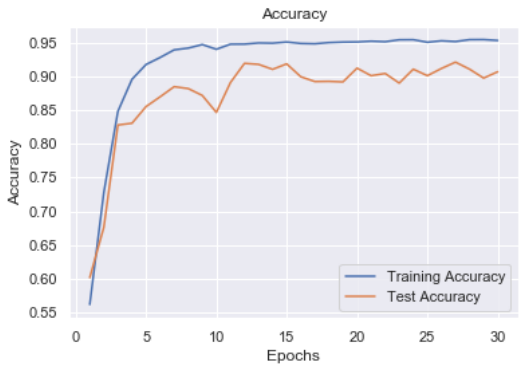
Accuracy



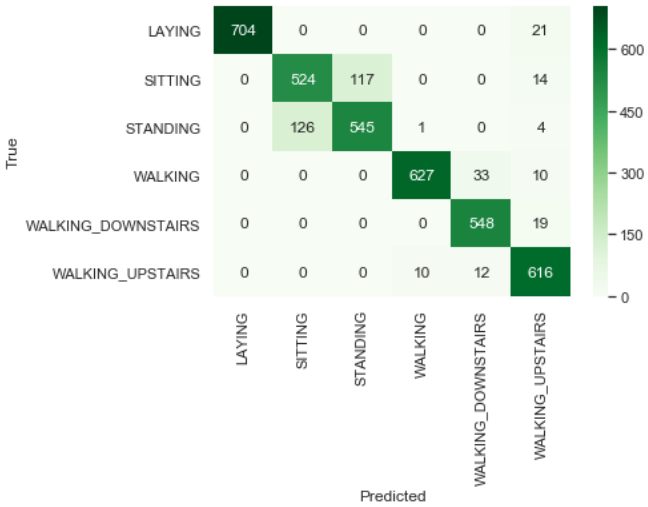
Confusion Matrix

- Hidden Layers: 35, Window length: 100, Overlapping: 50

Error



Accuracy



Confusion Matrix

For values more than 200 window size the performance is worst, that is observed from 300 window test. For window size 200 and overlapping 100 is the best result, which can be deduced from the confusion matrix.

6 CONCLUSIONS

- From the experiment we performed, we verified that LTSM is an excellent tool for sequential data learning. Nevertheless, all activities cannot be completely separated. For example, the data points collected during standing and sitting can sometimes be hard to be separated.

- Overlapping could be method for data augmentation in HAR but more testing is needed.

- It is important to verify the “uniformity” in train/test for all labels. If this step is not done, training or testing could be biased towards the most common activities in database.

- Tensorflow and Keras make the neural network implementation easy. Here, the task that utilized most of the effort is database preprocessing.

- Database seems to be insufficient for the neural network proposed. Larger databases should be tested.

- The datapoints for neural network proposed must have the same timeseries length and all must belong to the same label.

7 PYTHON CODE

```
import numpy as np
import tensorflow as tf
import pandas as pd
import os
import math
import random
import datetime
import seaborn as sn
from matplotlib import pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Data preprocessing parameters
n_classes = 6 # Number of trainable classes
elements_len = 200 # Number of elements per window
elements_offset = 50 # Offset for next window
train_ratio = 70; # Percentage of elements in training data
rndGeneration = False # Randomize position of training/test elements before LSTM execution

# Neural network parameters
epochs = 30 # Epochs in Neural Network
n_hidden = 30 # Number of hidden layers
batch_size = 10 # Batch size for training

# Activities Labels
Activity = ['WALKING','WALKING_UPSTAIRS','WALKING_DOWNSTAIRS','SITTING','STANDING','LAYING']

# Open Database
projectpath = os.path.dirname(os.getcwd())
datapath = projectpath + "\\Database\\UCI_HAR_Dataset"
dataname = "MainDatabase.csv"

# Load data as matrix (last column labels)
data = pd.read_csv(datapath + "\\" + dataname,sep=',')
sensor_names = data.columns
data_M = data.values

# Creating List timeseries by Activity
act_list = []
datasensor = []
num_sensor = data_M.shape[1]
datasensor.append(data_M[0,0:num_sensor]) # Sensors value for first timeserie (t=0)
prev_label = data_M[0,num_sensor-1] # Last column has activity label
for i in range(1,len(data_M)):
    label = data_M[i,num_sensor-1]
    if (label == prev_label):
        datasensor.append(data_M[i,0:num_sensor])
    else:
        act_list.append(np.vstack(datasensor)) # Convert vectors in datasensor to array and saves in act_list
        datasensor = []
        datasensor.append(data_M[i,0:num_sensor])
    prev_label = label
```

```
act_list.append(np.vstack(datasensor))

# Timeseries plot per Activity (only for visualization)
N = 12
acc = 0
for j in range(0,num_sensor):
   plt.figure(j)
   acc = 0
   for i in range(2,N):
      pp, = plt.plot(acc + np.arange(0,len(act_list[i])),act_list[i][:,j])
      acc = len(act_list[i]) + acc
   plt.title('Timeseries data per Activity')
   plt.ylabel(sensor_names[j])
init = len(act_list[i])
last = acc + len(act_list[i])

# Timeseries sensor values
for k in range(0,num_sensor-1):
   plt.figure(k)
   plt.plot(data_M[init:last,k])
   plt.title('Timeseries data')
   plt.ylabel(sensor_names[k])

# Dividing each element in activity list to a shorter version
elements = []
for i in range(0,len(act_list)):
   num_elements = math.floor((len(act_list[i])-elements_len)/elements_offset+1) # Number of elements with complete
timeseries
   for j in range(0,num_elements):
      start = elements_offset*j
      end = start + elements_len
      elements.append(act_list[i][start:end,0:num_sensor])

# Training and Test Data Generation
tt_list = elements.copy()
train_len = round(train_ratio*len(tt_list)/100)

# Randomize elements if rndGeneration = True
if (rndGeneration):
   random_list = random.sample(tt_list,len(tt_list))
   tt_list = random_list.copy()

# Data train Generation
datatrain = np.stack(tt_list[0:train_len]) # From list of arrays to tensor(:,:,:)
datatrain_x = datatrain[:,:,0:num_sensor-1]
datatrain_y = datatrain[:,5,num_sensor-1]

# Data test Generation
datatest = np.stack(tt_list[train_len:len(tt_list)]) # From list of arrays to tensor(:,:,:)
datatest_x = datatest[:,:,0:num_sensor-1]
datatest_y = datatest[:,5,num_sensor-1]

# Length of data verification #elements = #datatrain + #datatest + #dataval
total_data = len(datatrain)+len(datatest)
print("Length of datatrain:",len(datatrain),"(",round(len(datatrain)/total_data*100,3),"%)")
```

```
print("Length of datatest:",len(datatest),"(",round(len(datatest)/total_data*100,3),"%)")
print("Total length:",len(datatrain)+len(datatest))

# Folder for saving data statistics and results
ID = math.floor(random.uniform(0,99999))
result_path = projectpath +"\\Results\\Result_"+str(ID)
if not(os.path.exists(result_path)):
   os.mkdir(result_path)

# Parameters data
parameters_type =  ["Window length","Window offset","Train ratio","rndGeneration","Epoch","Hidden layers","Batch size"]
parameters_data =  [elements_len,elements_offset,train_ratio,rndGeneration,epochs,n_hidden,batch_size]
parameters_fr = pd.DataFrame(parameters_data)
parameters_fr.index = parameters_type
parameters_fr.to_csv(result_path+"\\Parameters.csv")

# Data Statistics
# Table
num_train = np.zeros(len(Activity))
num_test = np.zeros(len(Activity))
for i in range(0,len(Activity)):
   num_train[i] = sum(datatrain_y==i+1)
   num_test[i] = sum(datatest_y==i+1)

num_total = num_train + num_test
num_table = pd.DataFrame([num_train,num_test,num_total])
num_table.columns = Activity
num_table.index = ["Train","Test","Total"]
num_table.to_csv(result_path+"\\ActivityTable.csv")
num_table

# Data Statistics
# Bar Graphic
train_labels = datatrain_y.reshape(-1) # Total data train histogram
test_labels = datatest_y.reshape(-1) # Total data train histogram
data_labels = [train_labels,test_labels] # Total data histogram
plt.figure(1)
N = plt.hist(data_labels,bins=np.arange(0,7)+0.5,edgecolor='white',label=["Training","Testing"])
plt.xlabel('Activity ID')
plt.ylabel('Data points')
plt.ylim([0,max(N[0][0])+300])
plt.legend(loc='upper center', ncol=3)
plt.savefig(result_path+"\\DataStatistics.png")
plt.show()

# Converting training, test and validation data to one-hot format
datatrain_y_h = pd.get_dummies(datatrain_y).values
datatest_y_h = pd.get_dummies(datatest_y).values

# Initializing parameters
timesteps = len(datatrain_x[0])
input_dim = len(datatrain_x[0][0])

# Model Construction
model = Sequential()
```

```
# Configuring the parameters
model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim),return_sequences=False))
# Adding a dropout layer
model.add(Dropout(0.1))
# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))
model.summary()
# Save summary in results folder
with open(result_path + '\\report.txt','w') as fh:
   model.summary(print_fn=lambda x: fh.write(x + '\n'))

# Compiling model
model.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['accuracy'])

# Training the model
model_out = model.fit(datatrain_x,datatrain_y_h,batch_size=batch_size,
                validation_data=(datatest_x, datatest_y_h),epochs=epochs)

# Error and Accuracy over epochs (Tranning and Test)
# Error
train_err = model_out.history['loss']
test_err = model_out.history['val_loss']
# Plot Error
plt.plot(range(1,len(train_err)+1), train_err, label = 'Training Error')
plt.plot(range(1,len(train_err)+1), test_err, label = 'Test Error')
plt.legend()
plt.title("Error")
plt.xlabel('Epochs')
plt.ylabel('Error')
plt.savefig(result_path+"\\Error.png")
plt.show();

# Accuracy
train_acc = model_out.history['accuracy']
test_acc = model_out.history['val_accuracy']
# Plot Accuracy
plt.plot(range(1,len(train_acc)+1), train_acc, label = 'Training Accuracy')
plt.plot(range(1,len(train_acc)+1), test_acc, label = 'Test Accuracy')
plt.legend()
plt.title("Accuracy")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.savefig(result_path+"\\Accuracy.png")
plt.show();

# Confusion Matrix
datapred = model.predict(datatest_x)
Y_true = pd.Series([Activity[y] for y in np.argmax(datatest_y_h, axis=1)])
Y_pred = pd.Series([Activity[y] for y in np.argmax(datapred, axis=1)])
c_matrix = pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Predicted'])
sn.set(font_scale=1) # for label size
sn.heatmap(c_matrix, annot=True,cmap="Greens",fmt='g')
plt.savefig(result_path+"\\CMatrix.png",bbox_inches="tight")
```

```
# Rename folder according accuracy and error
last_acc = int(test_acc[len(test_acc)-1]*100000)
last_err = int(test_err[len(test_err)-1]*100000)
new_path = projectpath +"\\Results\\Result_A_"+str(last_acc)+"_E_"+str(last_err)
print(result_path)
print(new_path)
os.rename(result_path,new_path)
```

8 REFERENCES

1. www.machinelearningmaster.com
2. UCI (University of California, Irvine) machine learning repository
3. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
4. Deep Recurrent Neural Networks for Human Activity Recognition. Abduljamid Murad and Jae-Young Pyun
5. https://colah.github.io/posts/2015-08-Understanding-LSTMs/
6. Deep Residual Bidir-LSTM for Human Activity Recognition Using Wearable Sensors. Yu Zhao, R. Yang, G. Chevealier, X. Xu