Human Activity Recognition

This project is to build a model that predicts the human activities such as Walking, Walking_Upstairs, Walking_Downstairs, Sitting, Standing or Laying.

This dataset is collected from 30 persons(referred as subjects in this dataset), performing different activities with a smartphone to their waists. The data is recorded with the help of sensors (accelerometer and Gyroscope) in that smartphone. This experiment was video recorded to label the data manually.

How data was recorded

By using the sensors(Gyroscope and accelerometer) in a smartphone, they have captured '3-axial linear acceleration'(*tAcc-XYZ*) from accelerometer and '3-axial angular velocity' (*tGyro-XYZ*) from Gyroscope with several variations.

> prefix 't' in those metrics denotes time.
>
> suffix 'XYZ' represents 3-axial signals in X , Y, and Z directions.

Feature names

1. These sensor signals are preprocessed by applying noise filters and then sampled in fixed-width windows(sliding windows) of 2.56 seconds each with 50% overlap. ie., each window has 128 readings.
2. From Each window, a feature vector was obtianed by calculating variables from the time and frequency domain.

    > In our dataset, each datapoint represents a window with different readings

3. The accelertion signal was saperated into Body and Gravity acceleration signals(***tBodyAcc-XYZ*** and ***tGravityAcc-XYZ***) using some low pass filter with corner frequecy of 0.3Hz.
4. After that, the body linear acceleration and angular velocity were derived in time to obtian *jerk signals* (***tBodyAccJerk-XYZ*** and ***tBodyGyroJerk-XYZ***).
5. The magnitude of these 3-dimensional signals were calculated using the Euclidian norm. This magnitudes are represented as features with names like *tBodyAccMag*, *tGravityAccMag*, *tBodyAccJerkMag*, *tBodyGyroMag* and *tBodyGyroJerkMag*.
6. Finally, We've got frequency domain signals from some of the available signals by applying a FFT (Fast Fourier Transform). These signals obtained were labeled with ***prefix 'f'*** just like original signals with ***prefix 't'***. These signals are labeled as ***fBodyAcc-XYZ***, ***fBodyGyroMag*** etc.,.
7. These are the signals that we got so far.

    - tBodyAcc-XYZ
    - tGravityAcc-XYZ
    - tBodyAccJerk-XYZ
    - tBodyGyro-XYZ
    - tBodyGyroJerk-XYZ
    - tBodyAccMag
    - tGravityAccMag
    - tBodyAccJerkMag
    - tBodyGyroMag
    - tBodyGyroJerkMag
    - fBodyAcc-XYZ
    - fBodyAccJerk-XYZ
    - fBodyGyro-XYZ
    - fBodyAccMag
    - fBodyAccJerkMag
    - fBodyGyroMag
    - fBodyGyroJerkMag

8. We can esitmate some set of variables from the above signals. ie., We will estimate the following properties on each and every signal that we recoreded so far.

    - ***mean()***: Mean value
    - ***std()***: Standard deviation
    - ***mad()***: Median absolute deviation
    - ***max()***: Largest value in array
    - ***min()***: Smallest value in array
    - ***sma()***: Signal magnitude area
    - ***energy()***: Energy measure. Sum of the squares divided by the number of values.
    - ***iqr()***: Interquartile range

- ***entropy()***: Signal entropy
- ***arCoeff()***: Autorregresion coefficients with Burg order equal to 4
- ***correlation()***: correlation coefficient between two signals
- ***maxInds()***: index of the frequency component with largest magnitude
- ***meanFreq()***: Weighted average of the frequency components to obtain a mean frequency
- ***skewness()***: skewness of the frequency domain signal
- ***kurtosis()***: kurtosis of the frequency domain signal
- ***bandsEnergy()***: Energy of a frequency interval within the 64 bins of the FFT of each window.
- ***angle()***: Angle between to vectors.

9. We can obtain some other vectors by taking the average of signals in a single window sample. These are used on the angle() variable' `
    - gravityMean
    - tBodyAccMean
    - tBodyAccJerkMean
    - tBodyGyroMean
    - tBodyGyroJerkMean

Y_Labels(Encoded)

- In the dataset, Y_labels are represented as numbers from 1 to 6 as their identifiers.
    - WALKING as **1**
    - WALKING_UPSTAIRS as **2**
    - WALKING_DOWNSTAIRS as **3**
    - SITTING as **4**
    - STANDING as **5**
    - LAYING as **6**

Train and test data were saperated

- The readings from ***70%*** of the volunteers were taken as ***trianing data*** and remaining ***30%*** subjects recordings were taken for ***test data***

Data

- All the data is present in 'UCI_HAR_dataset/' folder in present working directory.
    - Feature names are present in 'UCI_HAR_dataset/features.txt'
    - ***Train Data***
        - 'UCI_HAR_dataset/train/X_train.txt'
        - 'UCI_HAR_dataset/train/subject_train.txt'
        - 'UCI_HAR_dataset/train/y_train.txt'
    - ***Test Data***
        - 'UCI_HAR_dataset/test/X_test.txt'
        - 'UCI_HAR_dataset/test/subject_test.txt'
        - 'UCI_HAR_dataset/test/y_test.txt'

Data Size :

> 27 MB

Quick overview of the dataset :

- Accelerometer and Gyroscope readings are taken from 30 volunteers(referred as subjects) while performing the following 6 Activities.

    1. Walking
    2. WalkingUpstairs
    3. WalkingDownstairs
    4. Standing
    5. Sitting
    6. Lying.

- Readings are divided into a window of 2.56 seconds with 50% overlapping.
- Accelerometer readings are divided into gravity acceleration and body acceleration readings, which has x,y and z components each.
- Gyroscope readings are the measure of angular velocities which has x,y and z components.
- Jerk signals are calculated for BodyAcceleration readings.
- Fourier Transforms are made on the above time readings to obtain frequency readings.
- Now, on all the base signal readings., mean, max, mad, sma, arcoefficient, engerybands,entropy etc., are calculated for each window.
- We get a feature vector of 561 features and these features are given in the dataset.
- Each window of readings is a datapoint of 561 features.

Problem Framework

* 30 subjects(volunteers) data is randomly split to 70%(21) test and 30%(7) train data.
* Each datapoint corresponds one of the 6 Activities.

Problem Statement

* Given a new datapoint we have to predict the Activity

In [1]:

```
# Importing necessary libraries

import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
import datetime as dt

# Machine Learning
from sklearn.manifold import TSNE
from sklearn.metrics import confusion_matrix
from sklearn import linear_model, metrics
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC, SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

# Deep Learning
import tensorflow as tf
import keras
from keras import backend as K
from keras import regularizers, optimizers
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv1D, MaxPooling1D
from keras.layers.normalization import BatchNormalization
from keras.models import Sequential
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import to_categorical
```

```
Using TensorFlow backend.
```

In [2]:

```
# get the features from the file features.txt

with open('features.txt') as f:
    # dividing "i" into two parts and considering 2nd part. ['1 tBodyAcc-mean()-X'], here 1 is 1st part and rest is 2nd part.
    features= [i.split()[1] for i in f.readlines()]

print('Sample features:\n',features[:5])
print('\nTotal no of features: ', len(features))
```

```
Sample features:
 ['tBodyAcc-mean()-X', 'tBodyAcc-mean()-Y', 'tBodyAcc-mean()-Z', 'tBodyAcc-std()-X', 'tBodyAcc-std()-Y']

Total no of features:  561
```

Obtain the train data

In [3]:

```
# get the data from .txt files to pandas dataframe
# delim_whitespace : bool, default False => Specifies whether or not whitespace (e.g. '' or ' ') will be used as the sep.
# Equivalent to setting sep='\s+'. If this option is set to True, nothing should be passed in for the delimiter parameter.

X_train= pd.read_csv('X_train.txt', delim_whitespace= True, header= None)
X_train.head()
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	...	551	552	553	554	
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.074323	-0.298676	-0.710304	-0.112754	0.0
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	0.158075	-0.595051	-0.861499	0.053477	0.0
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	0.414503	-0.390748	-0.760104	-0.118559	0.1
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	0.404573	-0.117290	-0.482845	-0.036788	0.0
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	0.087753	-0.351471	-0.699205	0.123320	0.1

5 rows × 561 columns

In [4]:

```
X_train.columns = features
```

In [5]:

```
X_train.head()
```

Out[5]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyGyroJerkMag-meanFreq()	f
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.074323	
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	0.158075	
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	0.414503	
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	0.404573	
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	0.087753	

5 rows × 561 columns

In [6]:

```
# add subject column to the dataframe

X_train['subject'] = pd.read_csv('subject_train.txt')
```

In [7]:

```
# squeeze : If the parsed data only contains one column then return a Series. (or else we cannot 'map' the below dict)

y_train = pd.read_csv('y_train.txt', names=['Activity'], squeeze= True)
y_train.head()
```

Out[7]:

```
0    5
1    5
2    5
3    5
4    5
Name: Activity, dtype: int64
```

In [8]:

```
y_train.value_counts()
```

Out[8]:

```
6    1407
5    1374
4    1286
1    1226
2    1073
3     986
Name: Activity, dtype: int64
```

In [9]:

```
# Labelling the classes in y.
label = {1:'WALKING', 2:'WALKING_UPSTAIRS', 3:'WALKING_DOWNSTAIRS', 4:'SITTING', 5:'STANDING', 6:'LAYING'}
```

In [10]:

```
y_train_labels = y_train.map(label)
y_train_labels.head()
```

Out[10]:

```
0    STANDING
1    STANDING
2    STANDING
3    STANDING
4    STANDING
Name: Activity, dtype: object
```

In [11]:

```
y_train_labels.value_counts()
```

Out[11]:

```
LAYING                1407
STANDING              1374
SITTING               1286
WALKING               1226
WALKING_UPSTAIRS      1073
WALKING_DOWNSTAIRS     986
Name: Activity, dtype: int64
```

In [12]:

```
# put all columns in a single dataframe

train = X_train
train['Activity'] = y_train
train['ActivityName'] = y_train_labels

# A sample row to check
train.sample()
```

Out[12]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	angle(tBodyAccMean,grav
205	0.278667	-0.017999	-0.110063	-0.997637	-0.990508	-0.989375	-0.998024	-0.989577	-0.991111	-0.940484	...	0.125

1 rows × 564 columns

In [13]:

```
train.shape
```

Out[13]:

```
(7352, 564)
```

Obtain the test data

In [31]:

```
# get the data from .txt files to pandas dataframe
# delim_whitespace : bool, default False => Specifies whether or not whitespace (e.g. '' or ' ') will be used as the sep.
# Equivalent to setting sep='\s+'. If this option is set to True, nothing should be passed in for the delimiter parameter.

X_test= pd.read_csv('X_test.txt', delim_whitespace= True, header= None)
X_test.head()
```

Out[31]:

	0	1	2	3	4	5	6	7	8	9	...	551	552	553	554	
0	0.257178	-0.023285	-0.014654	-0.938404	-0.920091	-0.667683	-0.952501	-0.925249	-0.674302	-0.894088	...	0.071645	-0.330370	-0.705974	0.006462	0.1
1	0.286027	-0.013163	-0.119083	-0.975415	-0.967458	-0.944958	-0.986799	-0.968401	-0.945823	-0.894088	...	-0.401189	-0.121845	-0.594944	-0.083495	0.0
2	0.275485	-0.026050	-0.118152	-0.993819	-0.969926	-0.962748	-0.994403	-0.970735	-0.963483	-0.939260	...	0.062891	-0.190422	-0.640736	-0.034956	0.2
3	0.270298	-0.032614	-0.117520	-0.994743	-0.973268	-0.967091	-0.995274	-0.974471	-0.968897	-0.938610	...	0.116695	-0.344418	-0.736124	-0.017067	0.1
4	0.274833	-0.027848	-0.129527	-0.993852	-0.967445	-0.978295	-0.994111	-0.965953	-0.977346	-0.938610	...	-0.121711	-0.534685	-0.846595	-0.002223	0.0

5 rows × 561 columns

In [32]:

```
X_test.columns = features
```

In [33]:

```
X_test.head()
```

Out[33]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyGyroJerkMag-meanFreq()	f
0	0.257178	-0.023285	-0.014654	-0.938404	-0.920091	-0.667683	-0.952501	-0.925249	-0.674302	-0.894088	...	0.071645	
1	0.286027	-0.013163	-0.119083	-0.975415	-0.967458	-0.944958	-0.986799	-0.968401	-0.945823	-0.894088	...	-0.401189	
2	0.275485	-0.026050	-0.118152	-0.993819	-0.969926	-0.962748	-0.994403	-0.970735	-0.963483	-0.939260	...	0.062891	
3	0.270298	-0.032614	-0.117520	-0.994743	-0.973268	-0.967091	-0.995274	-0.974471	-0.968897	-0.938610	...	0.116695	
4	0.274833	-0.027848	-0.129527	-0.993852	-0.967445	-0.978295	-0.994111	-0.965953	-0.977346	-0.938610	...	-0.121711	

5 rows × 561 columns

In [34]:

```
# add subject column to the dataframe

X_test['subject'] = pd.read_csv('subject_test.txt')
```

In [35]:

```
# squeeze : If the parsed data only contains one column then return a Series. (or else we cannot 'map' the below dict)

y_test = pd.read_csv('y_test.txt', names=['Activity'], squeeze= True)
y_test.head()
```

Out[35]:

```
0    5
1    5
2    5
3    5
4    5
Name: Activity, dtype: int64
```

In [36]:

```
y_test_labels = y_test.map(label)
y_test_labels.head()
```

Out[36]:

```
0    STANDING
1    STANDING
2    STANDING
3    STANDING
4    STANDING
Name: Activity, dtype: object
```

In [37]:

```
# put all columns in a single dataframe

test = X_test
test['Activity'] = y_test
test['ActivityName'] = y_test_labels

# A sample row to check
test.sample()
```

Out[37]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	angle(tBodyAccMean,gra
1896	0.276843	-0.01662	-0.109958	-0.996723	-0.992401	-0.993	-0.996914	-0.991882	-0.991154	-0.940405	...	0.17

1 rows × 564 columns

In [38]:

```
test.shape
```

Out[38]:

(2947, 564)

Data Cleaning

1. Check for Duplicates

In [48]:

```
print("There are {} of duplicates in train".format(len(train[train.duplicated()])))
print("There are {} of duplicates in test".format(len(test[test.duplicated()])))
```

There are 0 of duplicates in train
There are 0 of duplicates in test

2. Checking for NaN/null values

In [68]:

```
# Found one NaN row in train dataframe
train.isnull().values.sum()
```

Out[68]:

1

In [65]:

```
# Found one NaN row in test dataframe
test.isnull().values.sum()
```

Out[65]:

1

In [73]:

```
# https://stackoverflow.com/a/14247708/10219869
# The column is Subject and 7351 row in train dataset

train[train.isnull().any(axis=1)]
```

Out[73]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	angle(tBodyAccMean,gra
7351	0.351503	-0.012423	-0.203867	-0.26927	-0.087212	0.177404	-0.377404	-0.038678	0.22943	0.269013	...	-0.28

1 rows × 564 columns

In [82]:

```
# The column is Subject and 2946 row in test dataset

test[test.isnull().any(axis=1)]
```

Out[82]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	angle(tBodyAccMean,gra
2946	0.153627	-0.018437	-0.137018	-0.330046	-0.195253	-0.164339	-0.430974	-0.218295	-0.229933	-0.111527	...	0.59

1 rows × 564 columns

In [86]:

```
# deleting the particular rows

train.drop([7351], inplace= True)
test.drop([2946], inplace= True)
```

In [89]:

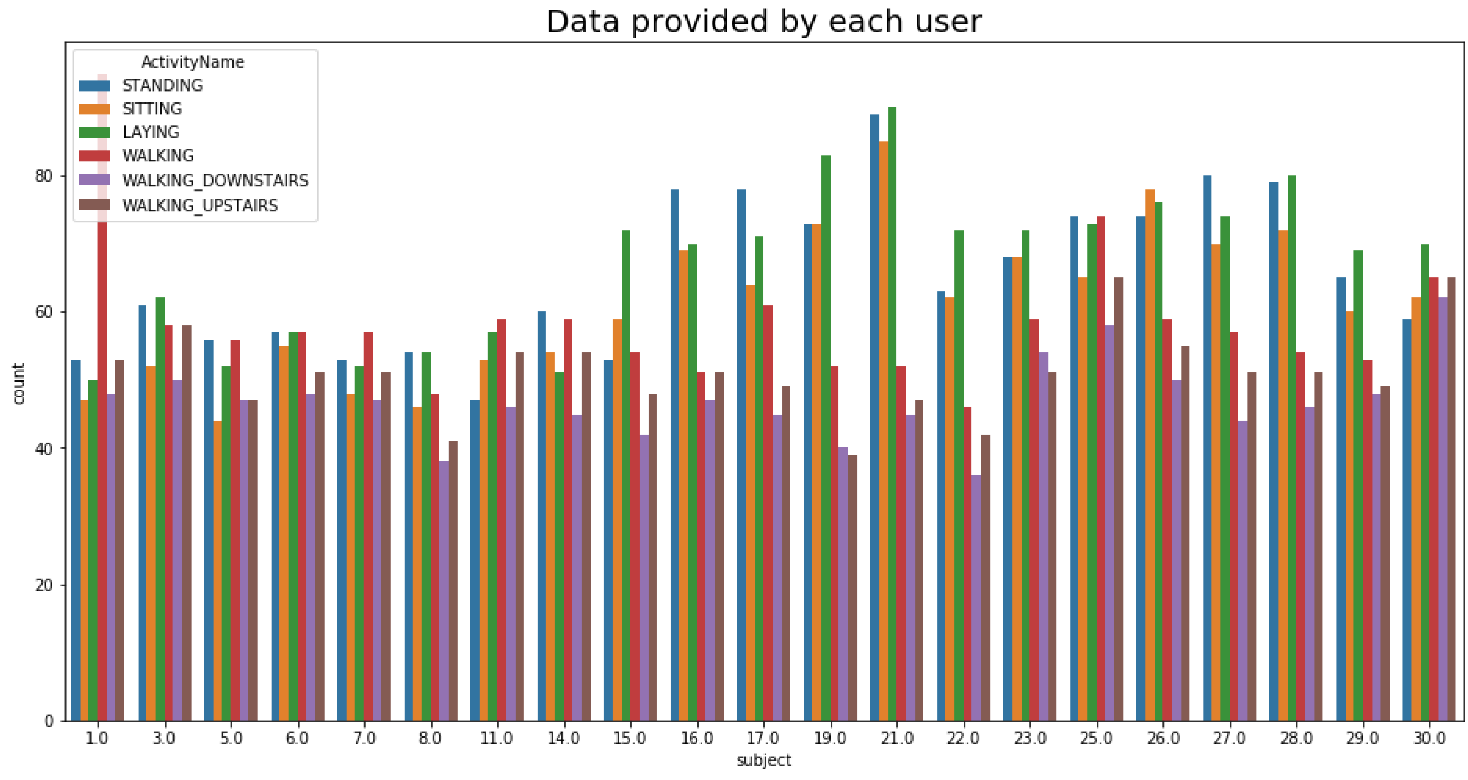
```
print("There are {} of NaN values in train".format(train.isnull().sum().sum()))
print("There are {} of NaN values in test".format(test.isnull().sum().sum()))
```

There are 0 of NaN values in train
There are 0 of NaN values in test

3. Check for data imbalance

In [90]:

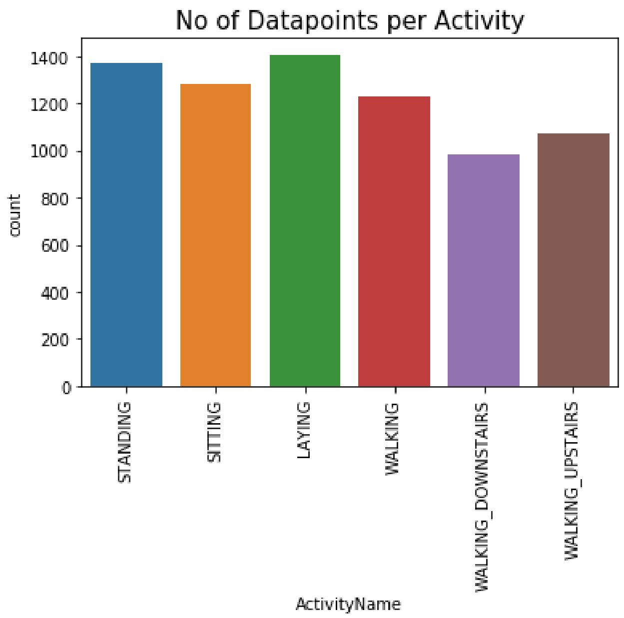
```
plt.figure(figsize=(16,8))
plt.title('Data provided by each user', fontsize=20)
sns.countplot(x='subject',hue='ActivityName', data = train)
plt.show()
```



> We have got almost same number of reading from all the subjects

In [91]:

```
plt.title('No of Datapoints per Activity', fontsize=15)
sns.countplot(train['ActivityName'])
plt.xticks(rotation=90)
plt.show()
```



Observation

> Our data is well balanced (almost)
>
> ## 4. Changing feature names

In [95]:

```
train.columns
```

Out[95]:

```
Index(['tBodyAcc-mean()-X', 'tBodyAcc-mean()-Y', 'tBodyAcc-mean()-Z',
       'tBodyAcc-std()-X', 'tBodyAcc-std()-Y', 'tBodyAcc-std()-Z',
       'tBodyAcc-mad()-X', 'tBodyAcc-mad()-Y', 'tBodyAcc-mad()-Z',
       'tBodyAcc-max()-X',
       ...
       'angle(tBodyAccMean,gravity)', 'angle(tBodyAccJerkMean),gravityMean)',
       'angle(tBodyGyroMean,gravityMean)',
       'angle(tBodyGyroJerkMean,gravityMean)', 'angle(X,gravityMean)',
       'angle(Y,gravityMean)', 'angle(Z,gravityMean)', 'subject', 'Activity',
       'ActivityName'],
      dtype='object', length=564)
```

In [105]:

```
# Removing '()','-',',' from column names
# https://stackoverflow.com/a/39741442/10219869
# why we use this "[]"? by  Bowen Liu. Because it means regex for matching only '()' or '-' or ','. By jezrael

train.columns = train.columns.str.replace(r'[()]','')
train.columns = train.columns.str.replace(r'[-]','')
train.columns = train.columns.str.replace(r'[,]','')
train.columns
```

Out[105]:

```
Index(['tBodyAccmeanX', 'tBodyAccmeanY', 'tBodyAccmeanZ', 'tBodyAccstdX',
```

'tBodyAccstdY', 'tBodyAccstdZ', 'tBodyAccmadX', 'tBodyAccmadY',
'tBodyAccmadZ', 'tBodyAccmaxX',
...
'angletBodyAccMeangravity', 'angletBodyAccJerkMeangravityMean',
'angletBodyGyroMeangravityMean', 'angletBodyGyroJerkMeangravityMean',
'angleXgravityMean', 'angleYgravityMean', 'angleZgravityMean',
'subject', 'Activity', 'ActivityName'],
dtype='object', length=564)

5. Save this dataframe in a csv files

In [106]:

```
train.to_csv('train_new.csv', index=False)
test.to_csv('test_new.csv', index=False)
```

Exploratory Data Analysis

"***Without domain knowledge EDA has no meaning, without EDA a problem has no soul.***"

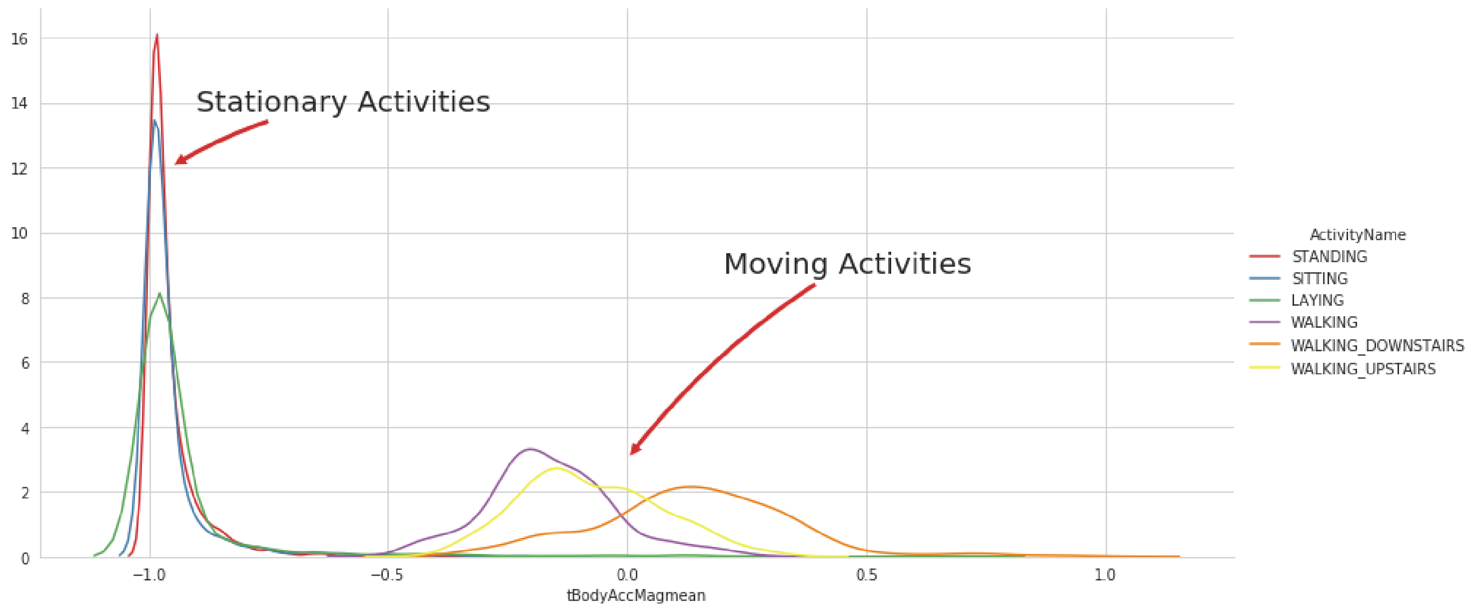
1. Featuring Engineering from Domain Knowledge

- **Static and Dynamic Activities**
  - In static activities (sit, stand, lie down) motion information will not be very useful.
  - In the dynamic activities (Walking, WalkingUpstairs,WalkingDownstairs) motion info will be significant.

2. Stationary and Moving activities are completely different

In [111]:

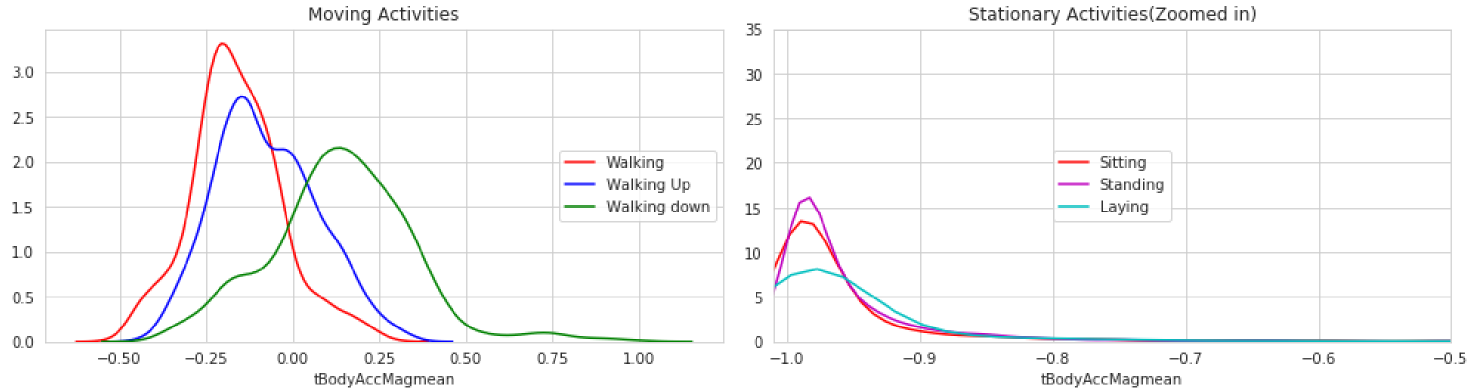
```
sns.set_style('whitegrid')

sns.set_palette("Set1", desat=0.80)

facetgrid = sns.FacetGrid(train, hue='ActivityName', size=6,aspect=2)

facetgrid.map(sns.distplot,'tBodyAccMagmean', hist=False).add_legend()

plt.annotate("Stationary Activities", xy= (-0.956,12), xytext= (-0.9, 14), size= 20,va= 'center', ha= 'left',
            arrowprops= dict(arrowstyle= "simple", connectionstyle= "arc3, rad= 0.1"))

plt.annotate("Moving Activities", xy= (0,3), xytext= (0.2, 9), size= 20, va= 'center', ha= 'left',
            arrowprops= dict(arrowstyle= "simple", connectionstyle= "arc3, rad= 0.1"))

plt.show()
```



In [113]:

```
plt.figure(figsize=(14,7))
plt.subplot(2,2,1)
```

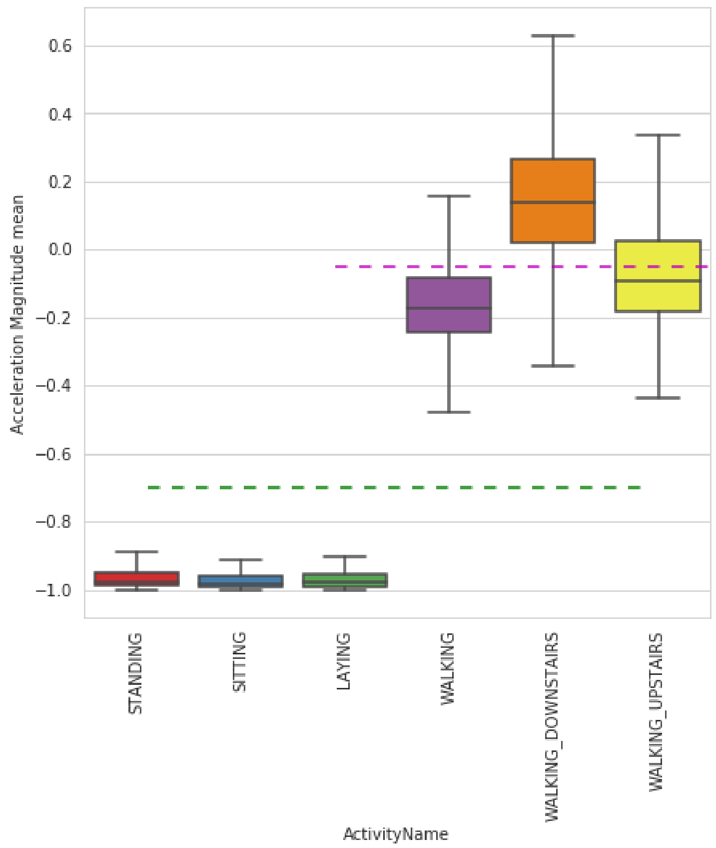
```
plt.title('Moving Activities')
sns.distplot(train[train['Activity']==1]['tBodyAccMagmean'],color = 'red',hist = False, label = 'Walking')
sns.distplot(train[train['Activity']==2]['tBodyAccMagmean'],color = 'blue',hist = False,label = 'Walking Up')
sns.distplot(train[train['Activity']==3]['tBodyAccMagmean'],color = 'green',hist = False, label = 'Walking down')
plt.legend(loc='center right')

plt.subplot(2,2,2)
plt.title('Stationary Activities(Zoomed in)')
sns.distplot(train[train['Activity']==4]['tBodyAccMagmean'],color = 'r',hist = False, label = 'Sitting')
sns.distplot(train[train['Activity']==5]['tBodyAccMagmean'],color = 'm',hist = False,label = 'Standing')
sns.distplot(train[train['Activity']==6]['tBodyAccMagmean'],color = 'c',hist = False, label = 'Laying')
plt.axis([-1.01, -0.5, 0, 35])
plt.legend(loc='center')

plt.tight_layout()
plt.show()
```



3. Magnitude of an acceleration can separate it well

In [114]:

```
plt.figure(figsize=(7,7))
sns.boxplot(x='ActivityName', y='tBodyAccMagmean',data=train, showfliers=False, saturation=1)
plt.ylabel('Acceleration Magnitude mean')
plt.axhline(y=-0.7, xmin=0.1, xmax=0.9,dashes=(5,5), c='g')
plt.axhline(y=-0.05, xmin=0.4, dashes=(5,5), c='m')
plt.xticks(rotation=90)
plt.show()
```



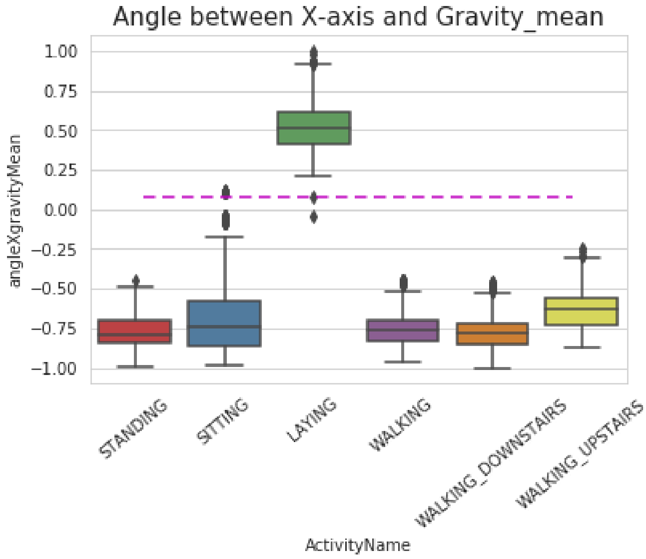
Observations:

- If tAccMean is < -0.8 then the Activities are either Standing or Sitting or Laying.
- If tAccMean is > -0.6 then the Activities are either Walking or WalkingDownstairs or WalkingUpstairs.
- If tAccMean > 0.1 then the Activity is WalkingDownstairs.
- We can classify almost 75% the Acitivity labels with some errors.

4. Position of Gravity Acceleration Components also matters

In [115]:

```
sns.boxplot(x='ActivityName', y='angleXgravityMean', data=train)
plt.axhline(y=0.08, xmin=0.1, xmax=0.9,c='m',dashes=(5,3))
plt.title('Angle between X-axis and Gravity_mean', fontsize=15)
plt.xticks(rotation = 40)
plt.show()
```

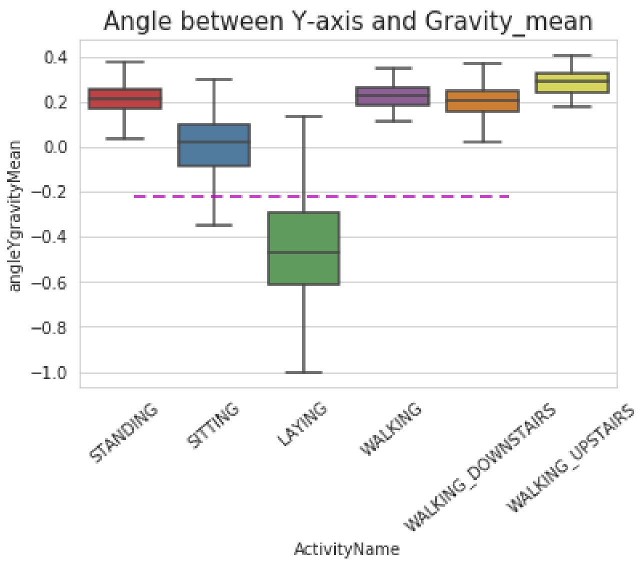


Observations:

- If angleX, gravityMean > 0 then Activity is Laying.
- We can classify all datapoints belonging to Laying activity with just a single if else statement.

In [116]:

```
sns.boxplot(x='ActivityName', y='angleYgravityMean', data = train, showfliers=False)
plt.title('Angle between Y-axis and Gravity_mean', fontsize=15)
plt.xticks(rotation = 40)
plt.axhline(y=-0.22, xmin=0.1, xmax=0.8, dashes=(5,3), c='m')
plt.show()
```



Apply T-SNE on the data

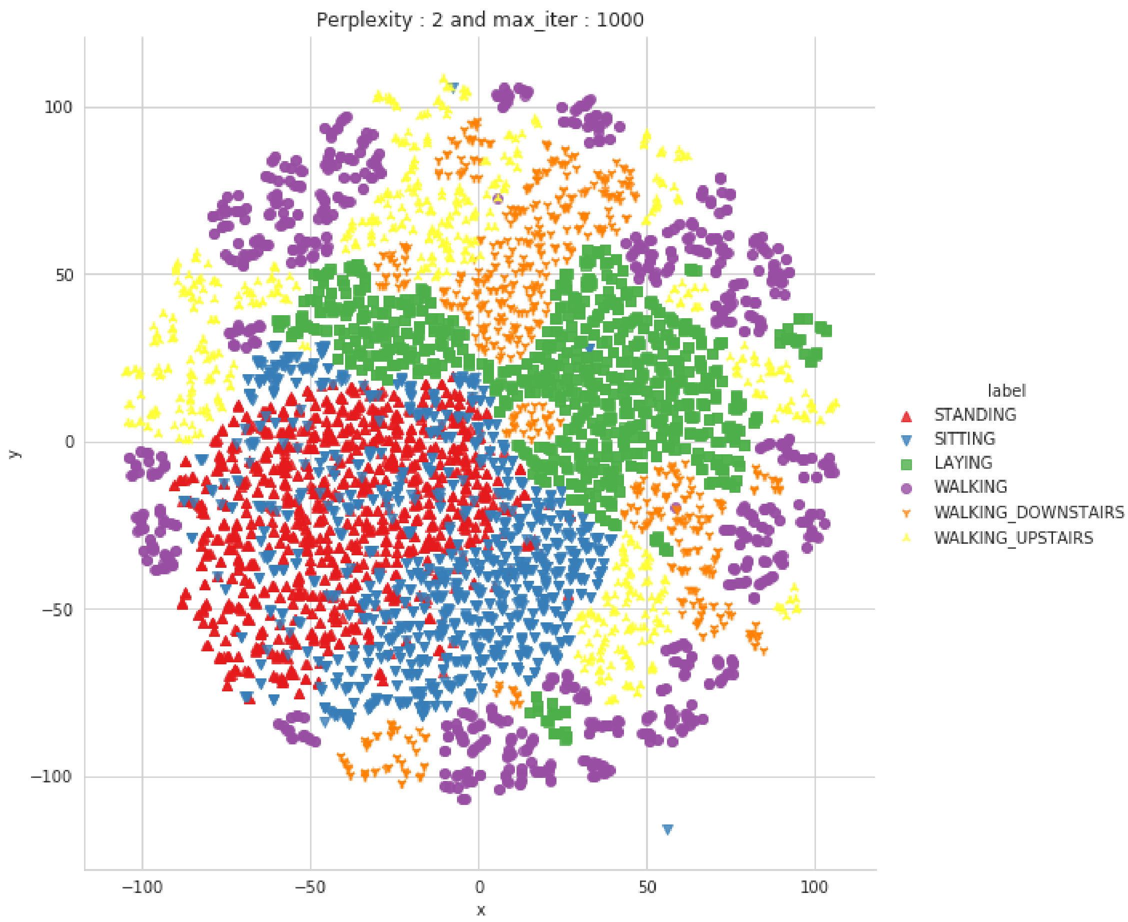
In [128]:

```
# performs t-sne with different perplexity values and their repective plots..

def perform_tsne(X, y, perplexities, n_iter=1000, img_name_prefix='T-SNE'):

    for index, perplexity in enumerate(perplexities):
        # perform t-sne
        print('\nPerforming tsne with perplexity {} and with {} iterations at max\n'.format(perplexity, n_iter))
        X_reduced = TSNE(verbose= 2, perplexity= perplexity).fit_transform(X)
        print('Done..')

        # prepare the data for seaborn
        print('\nCreating plot for this T-SNE visualization..\n')
        df = pd.DataFrame({'x': X_reduced[:,0], 'y': X_reduced[:,1] ,'label': y})

        # draw the plot in appropriate place in the grid
        # fit_reg : If 'True', estimate and plot a regression model relating the x and y variables.
        sns.lmplot(data= df, x= 'x', y= 'y', hue= 'label', fit_reg= False, size= 8, palette="Set1",
                   markers=['^', 'v', 's', 'o', '1', '2'])

        plt.title("Perplexity : {} and max_iter : {}".format(perplexity, n_iter))
        img_name = img_name_prefix + 'New_perp_{}_iter_{}.png'.format(perplexity, n_iter)
        print('\nSaving this plot as image in present working directory...')
        plt.savefig(img_name)
        plt.show()
        print('Done')
```

In [129]:

```
start = dt.datetime.now()

perform_tsne(X= train.drop(['subject', 'Activity','ActivityName'], axis=1) , y= train['ActivityName'],
            perplexities =[2,5,10,30,50])

print('Time to run the program: ',dt.datetime.now()-start)
```

```
Performing tsne with perplexity 2 and with 1000 iterations at max

[t-SNE] Computing 7 nearest neighbors...
[t-SNE] Indexed 7351 samples in 0.135s...
[t-SNE] Computed neighbors for 7351 samples in 30.775s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7351
[t-SNE] Computed conditional probabilities for sample 2000 / 7351
[t-SNE] Computed conditional probabilities for sample 3000 / 7351
[t-SNE] Computed conditional probabilities for sample 4000 / 7351
[t-SNE] Computed conditional probabilities for sample 5000 / 7351
[t-SNE] Computed conditional probabilities for sample 6000 / 7351
[t-SNE] Computed conditional probabilities for sample 7000 / 7351
[t-SNE] Computed conditional probabilities for sample 7351 / 7351
[t-SNE] Mean sigma: 0.635915
[t-SNE] Computed conditional probabilities in 0.063s
[t-SNE] Iteration 50: error = 124.6781464, gradient norm = 0.0262113 (50 iterations in 4.723s)
[t-SNE] Iteration 100: error = 107.3815079, gradient norm = 0.0308419 (50 iterations in 3.293s)
[t-SNE] Iteration 150: error = 101.1956329, gradient norm = 0.0186720 (50 iterations in 2.541s)
[t-SNE] Iteration 200: error = 97.8194199, gradient norm = 0.0183977 (50 iterations in 2.469s)
[t-SNE] Iteration 250: error = 95.4880829, gradient norm = 0.0138392 (50 iterations in 2.453s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 95.488083
[t-SNE] Iteration 300: error = 4.1237102, gradient norm = 0.0015637 (50 iterations in 2.158s)
[t-SNE] Iteration 350: error = 3.2142746, gradient norm = 0.0009976 (50 iterations in 2.027s)
[t-SNE] Iteration 400: error = 2.7847106, gradient norm = 0.0007132 (50 iterations in 2.036s)
[t-SNE] Iteration 450: error = 2.5202210, gradient norm = 0.0005710 (50 iterations in 2.061s)
[t-SNE] Iteration 500: error = 2.3367610, gradient norm = 0.0004885 (50 iterations in 2.055s)
[t-SNE] Iteration 550: error = 2.1992407, gradient norm = 0.0004189 (50 iterations in 2.066s)
[t-SNE] Iteration 600: error = 2.0901325, gradient norm = 0.0003677 (50 iterations in 2.072s)
[t-SNE] Iteration 650: error = 2.0004115, gradient norm = 0.0003331 (50 iterations in 2.090s)
[t-SNE] Iteration 700: error = 1.9249381, gradient norm = 0.0002990 (50 iterations in 2.088s)
[t-SNE] Iteration 750: error = 1.8601872, gradient norm = 0.0002726 (50 iterations in 2.116s)
[t-SNE] Iteration 800: error = 1.8036064, gradient norm = 0.0002561 (50 iterations in 2.122s)
[t-SNE] Iteration 850: error = 1.7536288, gradient norm = 0.0002380 (50 iterations in 2.126s)
[t-SNE] Iteration 900: error = 1.7090988, gradient norm = 0.0002241 (50 iterations in 2.140s)
[t-SNE] Iteration 950: error = 1.6690996, gradient norm = 0.0002148 (50 iterations in 2.143s)
[t-SNE] Iteration 1000: error = 1.6328787, gradient norm = 0.0001981 (50 iterations in 2.141s)
[t-SNE] KL divergence after 1000 iterations: 1.632879
Done..

Creating plot for this T-SNE visualization..


Saving this plot as image in present working directory...
```



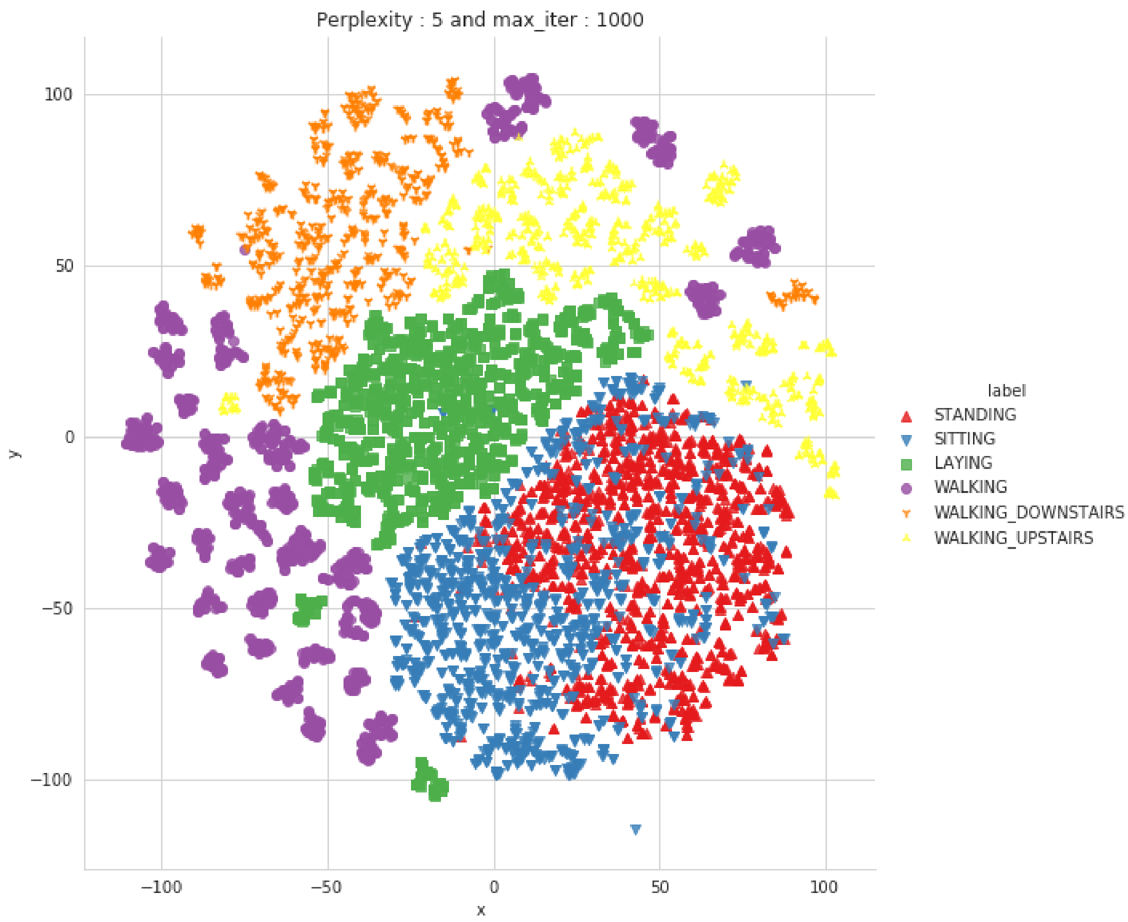
Done

Performing tsne with perplexity 5 and with 1000 iterations at max

```
[t-SNE] Computing 16 nearest neighbors...
[t-SNE] Indexed 7351 samples in 0.127s...
[t-SNE] Computed neighbors for 7351 samples in 30.790s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7351
[t-SNE] Computed conditional probabilities for sample 2000 / 7351
[t-SNE] Computed conditional probabilities for sample 3000 / 7351
[t-SNE] Computed conditional probabilities for sample 4000 / 7351
[t-SNE] Computed conditional probabilities for sample 5000 / 7351
[t-SNE] Computed conditional probabilities for sample 6000 / 7351
[t-SNE] Computed conditional probabilities for sample 7000 / 7351
[t-SNE] Computed conditional probabilities for sample 7351 / 7351
[t-SNE] Mean sigma: 0.961278
[t-SNE] Computed conditional probabilities in 0.077s
[t-SNE] Iteration 50: error = 114.0015182, gradient norm = 0.0204809 (50 iterations in 8.726s)
[t-SNE] Iteration 100: error = 97.6693344, gradient norm = 0.0169447 (50 iterations in 2.548s)
[t-SNE] Iteration 150: error = 93.2642059, gradient norm = 0.0089610 (50 iterations in 2.071s)
[t-SNE] Iteration 200: error = 91.2998428, gradient norm = 0.0072925 (50 iterations in 2.006s)
[t-SNE] Iteration 250: error = 90.1137543, gradient norm = 0.0046948 (50 iterations in 1.976s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 90.113754
[t-SNE] Iteration 300: error = 3.5745916, gradient norm = 0.0014597 (50 iterations in 1.956s)
[t-SNE] Iteration 350: error = 2.8179624, gradient norm = 0.0007535 (50 iterations in 1.959s)
[t-SNE] Iteration 400: error = 2.4369547, gradient norm = 0.0005287 (50 iterations in 1.973s)
[t-SNE] Iteration 450: error = 2.2189448, gradient norm = 0.0004023 (50 iterations in 2.007s)
[t-SNE] Iteration 500: error = 2.0741067, gradient norm = 0.0003303 (50 iterations in 2.012s)
[t-SNE] Iteration 550: error = 1.9687753, gradient norm = 0.0002813 (50 iterations in 2.040s)
[t-SNE] Iteration 600: error = 1.8874637, gradient norm = 0.0002458 (50 iterations in 2.050s)
[t-SNE] Iteration 650: error = 1.8225235, gradient norm = 0.0002170 (50 iterations in 2.050s)
[t-SNE] Iteration 700: error = 1.7688591, gradient norm = 0.0001972 (50 iterations in 2.062s)
[t-SNE] Iteration 750: error = 1.7235245, gradient norm = 0.0001807 (50 iterations in 2.068s)
[t-SNE] Iteration 800: error = 1.6846262, gradient norm = 0.0001649 (50 iterations in 2.064s)
[t-SNE] Iteration 850: error = 1.6506555, gradient norm = 0.0001541 (50 iterations in 2.057s)
[t-SNE] Iteration 900: error = 1.6211171, gradient norm = 0.0001427 (50 iterations in 2.055s)
[t-SNE] Iteration 950: error = 1.5946183, gradient norm = 0.0001323 (50 iterations in 2.055s)
[t-SNE] Iteration 1000: error = 1.5709391, gradient norm = 0.0001268 (50 iterations in 2.052s)
[t-SNE] KL divergence after 1000 iterations: 1.570939
Done..
```

Creating plot for this T-SNE visualization..

Saving this plot as image in present working directory...



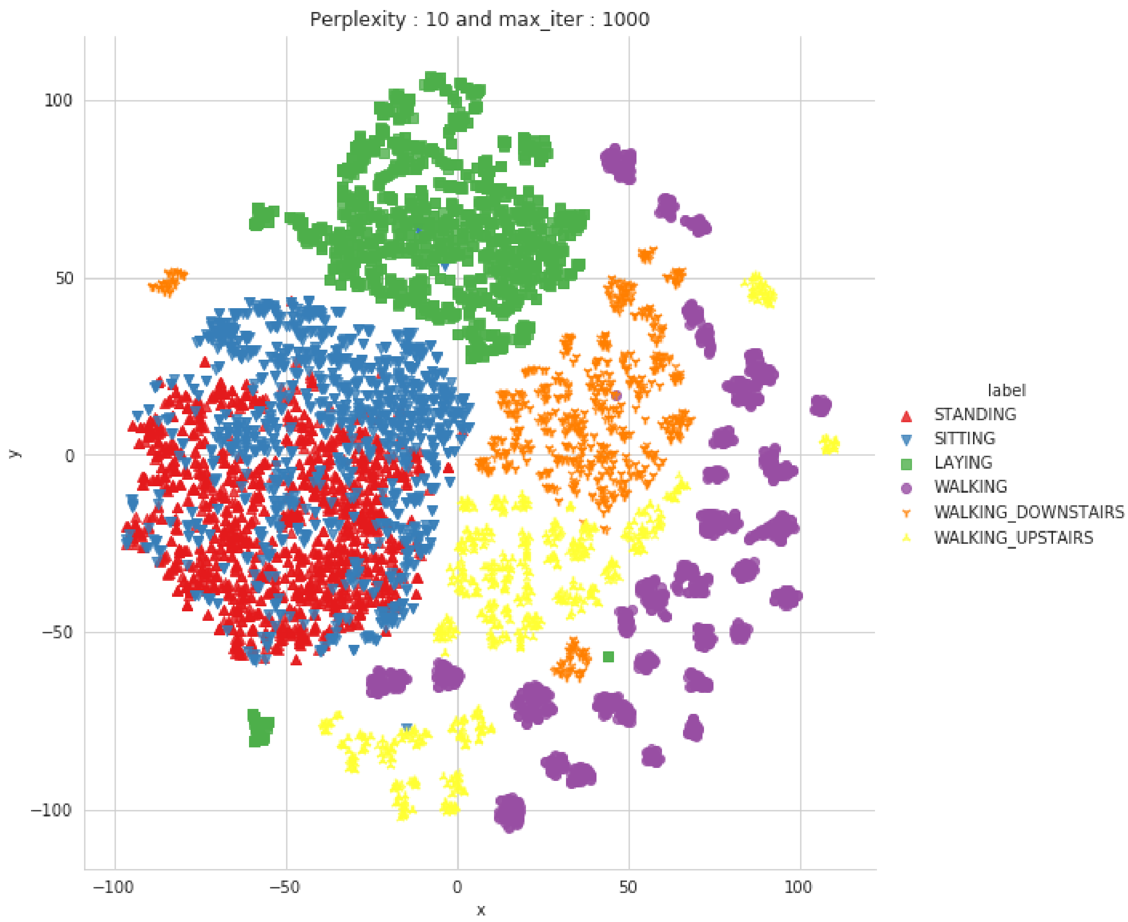
Done

Performing tsne with perplexity 10 and with 1000 iterations at max

[t-SNE] Computing 31 nearest neighbors...
[t-SNE] Indexed 7351 samples in 0.129s...
[t-SNE] Computed neighbors for 7351 samples in 31.246s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7351
[t-SNE] Computed conditional probabilities for sample 2000 / 7351
[t-SNE] Computed conditional probabilities for sample 3000 / 7351
[t-SNE] Computed conditional probabilities for sample 4000 / 7351
[t-SNE] Computed conditional probabilities for sample 5000 / 7351
[t-SNE] Computed conditional probabilities for sample 6000 / 7351
[t-SNE] Computed conditional probabilities for sample 7000 / 7351
[t-SNE] Computed conditional probabilities for sample 7351 / 7351
[t-SNE] Mean sigma: 1.133834
[t-SNE] Computed conditional probabilities in 0.138s
[t-SNE] Iteration 50: error = 105.7958527, gradient norm = 0.0176003 (50 iterations in 3.840s)
[t-SNE] Iteration 100: error = 90.1865311, gradient norm = 0.0132296 (50 iterations in 2.602s)
[t-SNE] Iteration 150: error = 87.2278900, gradient norm = 0.0053869 (50 iterations in 2.371s)
[t-SNE] Iteration 200: error = 86.0240326, gradient norm = 0.0060858 (50 iterations in 2.321s)
[t-SNE] Iteration 250: error = 85.3396454, gradient norm = 0.0045784 (50 iterations in 2.310s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.339645
[t-SNE] Iteration 300: error = 3.1301837, gradient norm = 0.0013851 (50 iterations in 2.166s)
[t-SNE] Iteration 350: error = 2.4884892, gradient norm = 0.0006497 (50 iterations in 2.035s)
[t-SNE] Iteration 400: error = 2.1681504, gradient norm = 0.0004282 (50 iterations in 2.061s)
[t-SNE] Iteration 450: error = 1.9845430, gradient norm = 0.0003161 (50 iterations in 2.065s)
[t-SNE] Iteration 500: error = 1.8663766, gradient norm = 0.0002508 (50 iterations in 2.102s)
[t-SNE] Iteration 550: error = 1.7828290, gradient norm = 0.0002096 (50 iterations in 2.088s)
[t-SNE] Iteration 600: error = 1.7198648, gradient norm = 0.0001806 (50 iterations in 2.104s)
[t-SNE] Iteration 650: error = 1.6710820, gradient norm = 0.0001585 (50 iterations in 2.085s)
[t-SNE] Iteration 700: error = 1.6317736, gradient norm = 0.0001428 (50 iterations in 2.092s)
[t-SNE] Iteration 750: error = 1.5992377, gradient norm = 0.0001300 (50 iterations in 2.092s)
[t-SNE] Iteration 800: error = 1.5719700, gradient norm = 0.0001185 (50 iterations in 2.090s)
[t-SNE] Iteration 850: error = 1.5487494, gradient norm = 0.0001125 (50 iterations in 2.096s)
[t-SNE] Iteration 900: error = 1.5292450, gradient norm = 0.0001048 (50 iterations in 2.116s)
[t-SNE] Iteration 950: error = 1.5126431, gradient norm = 0.0000998 (50 iterations in 2.118s)
[t-SNE] Iteration 1000: error = 1.4984311, gradient norm = 0.0000934 (50 iterations in 2.126s)
[t-SNE] KL divergence after 1000 iterations: 1.498431
Done..

Creating plot for this T-SNE visualization..

Saving this plot as image in present working directory...



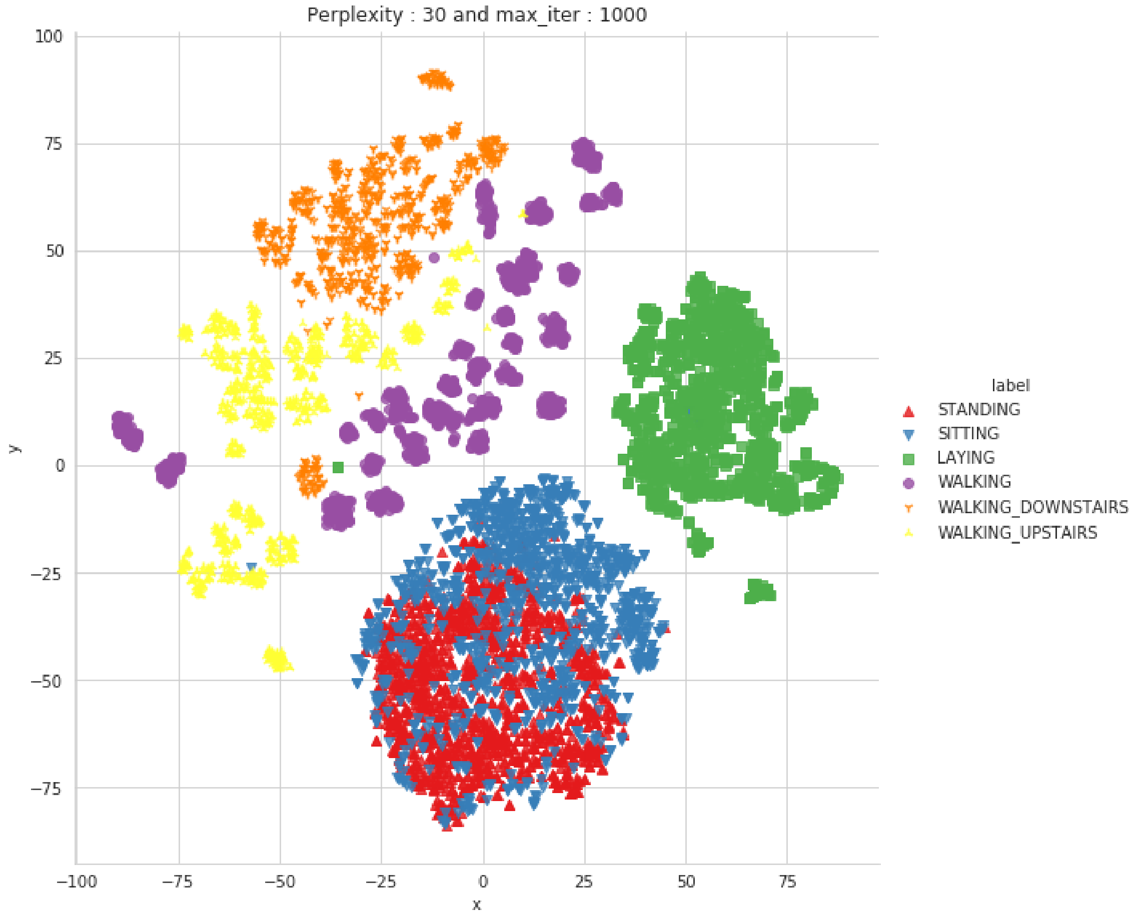
Done

Performing tsne with perplexity 30 and with 1000 iterations at max

[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 7351 samples in 0.138s...
[t-SNE] Computed neighbors for 7351 samples in 33.190s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7351
[t-SNE] Computed conditional probabilities for sample 2000 / 7351
[t-SNE] Computed conditional probabilities for sample 3000 / 7351
[t-SNE] Computed conditional probabilities for sample 4000 / 7351
[t-SNE] Computed conditional probabilities for sample 5000 / 7351
[t-SNE] Computed conditional probabilities for sample 6000 / 7351
[t-SNE] Computed conditional probabilities for sample 7000 / 7351
[t-SNE] Computed conditional probabilities for sample 7351 / 7351
[t-SNE] Mean sigma: 1.348514
[t-SNE] Computed conditional probabilities in 0.389s
[t-SNE] Iteration 50: error = 91.7951050, gradient norm = 0.0282668 (50 iterations in 4.036s)
[t-SNE] Iteration 100: error = 80.4392471, gradient norm = 0.0043850 (50 iterations in 3.046s)
[t-SNE] Iteration 150: error = 78.8470230, gradient norm = 0.0029605 (50 iterations in 2.704s)
[t-SNE] Iteration 200: error = 78.3061218, gradient norm = 0.0023445 (50 iterations in 2.727s)
[t-SNE] Iteration 250: error = 78.0242996, gradient norm = 0.0017019 (50 iterations in 2.729s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 78.024300
[t-SNE] Iteration 300: error = 2.4515841, gradient norm = 0.0012605 (50 iterations in 2.532s)
[t-SNE] Iteration 350: error = 1.9794496, gradient norm = 0.0005343 (50 iterations in 2.428s)
[t-SNE] Iteration 400: error = 1.7674819, gradient norm = 0.0003214 (50 iterations in 2.449s)
[t-SNE] Iteration 450: error = 1.6460981, gradient norm = 0.0002195 (50 iterations in 2.499s)
[t-SNE] Iteration 500: error = 1.5669320, gradient norm = 0.0001676 (50 iterations in 2.478s)
[t-SNE] Iteration 550: error = 1.5127892, gradient norm = 0.0001359 (50 iterations in 2.481s)
[t-SNE] Iteration 600: error = 1.4738222, gradient norm = 0.0001171 (50 iterations in 2.500s)
[t-SNE] Iteration 650: error = 1.4452159, gradient norm = 0.0001085 (50 iterations in 2.500s)
[t-SNE] Iteration 700: error = 1.4241163, gradient norm = 0.0000926 (50 iterations in 2.494s)
[t-SNE] Iteration 750: error = 1.4076157, gradient norm = 0.0000858 (50 iterations in 2.481s)
[t-SNE] Iteration 800: error = 1.3945322, gradient norm = 0.0000798 (50 iterations in 2.497s)
[t-SNE] Iteration 850: error = 1.3838215, gradient norm = 0.0000744 (50 iterations in 2.497s)
[t-SNE] Iteration 900: error = 1.3747298, gradient norm = 0.0000760 (50 iterations in 2.502s)
[t-SNE] Iteration 950: error = 1.3668323, gradient norm = 0.0000719 (50 iterations in 2.536s)
[t-SNE] Iteration 1000: error = 1.3600103, gradient norm = 0.0000712 (50 iterations in 2.520s)
[t-SNE] KL divergence after 1000 iterations: 1.360010
Done..

Creating plot for this T-SNE visualization..

Saving this plot as image in present working directory...



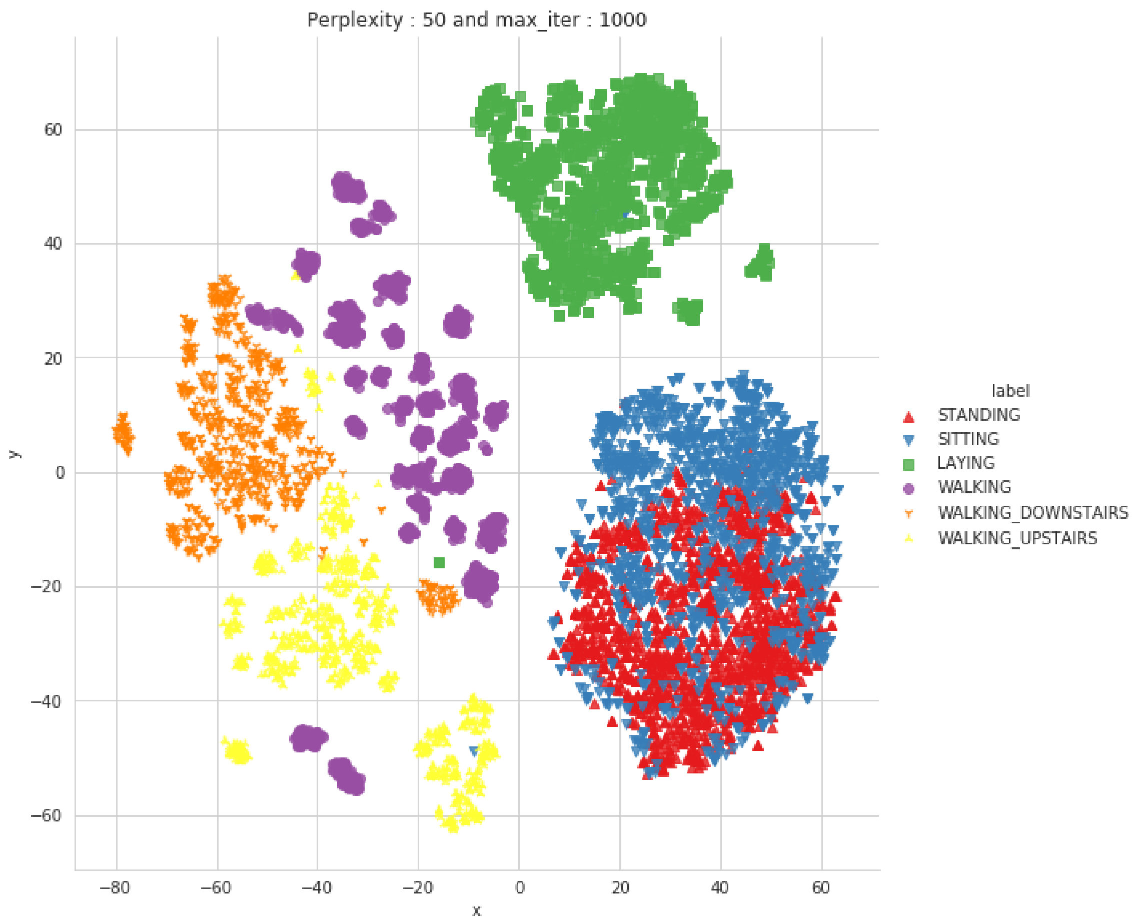
Done

Performing tsne with perplexity 50 and with 1000 iterations at max

[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 7351 samples in 0.162s...
[t-SNE] Computed neighbors for 7351 samples in 35.874s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7351
[t-SNE] Computed conditional probabilities for sample 2000 / 7351
[t-SNE] Computed conditional probabilities for sample 3000 / 7351
[t-SNE] Computed conditional probabilities for sample 4000 / 7351
[t-SNE] Computed conditional probabilities for sample 5000 / 7351
[t-SNE] Computed conditional probabilities for sample 6000 / 7351
[t-SNE] Computed conditional probabilities for sample 7000 / 7351
[t-SNE] Computed conditional probabilities for sample 7351 / 7351
[t-SNE] Mean sigma: 1.437667
[t-SNE] Computed conditional probabilities in 0.645s
[t-SNE] Iteration 50: error = 85.8026886, gradient norm = 0.0274312 (50 iterations in 4.849s)
[t-SNE] Iteration 100: error = 75.5111008, gradient norm = 0.0039719 (50 iterations in 4.618s)
[t-SNE] Iteration 150: error = 74.5812683, gradient norm = 0.0020083 (50 iterations in 3.521s)
[t-SNE] Iteration 200: error = 74.2340546, gradient norm = 0.0018849 (50 iterations in 3.597s)
[t-SNE] Iteration 250: error = 74.0605164, gradient norm = 0.0012819 (50 iterations in 3.708s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.060516
[t-SNE] Iteration 300: error = 2.1528261, gradient norm = 0.0011935 (50 iterations in 3.272s)
[t-SNE] Iteration 350: error = 1.7556928, gradient norm = 0.0004826 (50 iterations in 2.869s)
[t-SNE] Iteration 400: error = 1.5864977, gradient norm = 0.0002839 (50 iterations in 2.830s)
[t-SNE] Iteration 450: error = 1.4932637, gradient norm = 0.0001896 (50 iterations in 2.832s)
[t-SNE] Iteration 500: error = 1.4335964, gradient norm = 0.0001399 (50 iterations in 2.848s)
[t-SNE] Iteration 550: error = 1.3925201, gradient norm = 0.0001118 (50 iterations in 2.851s)
[t-SNE] Iteration 600: error = 1.3632234, gradient norm = 0.0000974 (50 iterations in 2.845s)
[t-SNE] Iteration 650: error = 1.3423645, gradient norm = 0.0000826 (50 iterations in 2.835s)
[t-SNE] Iteration 700: error = 1.3267196, gradient norm = 0.0000743 (50 iterations in 2.829s)
[t-SNE] Iteration 750: error = 1.3152233, gradient norm = 0.0000672 (50 iterations in 2.831s)
[t-SNE] Iteration 800: error = 1.3060203, gradient norm = 0.0000633 (50 iterations in 2.850s)
[t-SNE] Iteration 850: error = 1.2988074, gradient norm = 0.0000600 (50 iterations in 2.864s)
[t-SNE] Iteration 900: error = 1.2930111, gradient norm = 0.0000568 (50 iterations in 2.863s)
[t-SNE] Iteration 950: error = 1.2882552, gradient norm = 0.0000555 (50 iterations in 2.855s)
[t-SNE] Iteration 1000: error = 1.2842467, gradient norm = 0.0000519 (50 iterations in 2.834s)
[t-SNE] KL divergence after 1000 iterations: 1.284247
Done..

Creating plot for this T-SNE visualization..

Saving this plot as image in present working directory...



Done
Time to run the program: 0:07:04.016172

Function to plot the confusion matrix

In [131]:

```
def plot_confusion_matrix(cm, classes,normalize=False, title='Confusion matrix', cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center", color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Generic function to run any model specified

In [157]:

```
def perform_model(model, X_train, y_train, X_test, y_test, class_labels, cm_normalize=True, print_cm=True,
                 cm_cmap=plt.cm.Greens):


    # to store results at various phases
    results = dict()

    # time at which model starts training
    train_start = dt.datetime.now()
```

```
    train_start = dt.datetime.now()
    print('Training the model..')
    model.fit(X_train, y_train)
    print('Done \n \n')
    results['training_time'] =  dt.datetime.now() - train_start
    print('training_time(HH:MM:SS.ms) - {}\n\n'.format(results['training_time']))


    # predict test data
    print('Predicting test data')
    test_start = dt.datetime.now()
    y_pred = model.predict(X_test)
    print('Done \n \n')
    results['testing_time'] = dt.datetime.now() - test_start
    print('testing time(HH:MM:SS:ms) - {}\n\n'.format(results['testing_time']))
    results['predicted'] = y_pred


    # calculate overall accuracty of the model
    accuracy = metrics.accuracy_score(y_true= y_test, y_pred= y_pred)
    # store accuracy in results
    results['accuracy'] = accuracy
    print('--------------------')
    print('|      Accuracy      |')
    print('--------------------')
    print('\n    {}\n\n'.format(accuracy))


    # confusion matrix
    cm = metrics.confusion_matrix(y_test, y_pred)
    results['confusion_matrix'] = cm
    if print_cm:
        print('--------------------')
        print('| Confusion Matrix |')
        print('--------------------')
        print('\n {}'.format(cm))

    # plot confusin matrix
    plt.figure(figsize=(8,8))
    plt.grid(b=False)
    plot_confusion_matrix(cm, classes=class_labels, normalize=True, title='Normalized confusion matrix', cmap = cm_cmap)
    plt.show()

    # get classification report
    print('-------------------------')
    print('| Classification Report |')
    print('-------------------------')
    classification_report = metrics.classification_report(y_test, y_pred)
    # store report in results
    results['classification_report'] = classification_report
    print(classification_report)

    # add the trained  model to the results
    results['model'] = model

    return results
```

Method to print the gridsearch Attributes

In [134]:

```
def print_grid_search_attributes(model):

    # Estimator that gave highest score among all the estimators formed in GridSearch
    print('--------------------------')
    print('|      Best Estimator     |')
    print('--------------------------')
    print('\n\t{}\n'.format(model.best_estimator_))


    # parameters that gave best results while performing grid search
    print('--------------------------')
    print('|     Best parameters     |')
    print('--------------------------')
    print('\tParameters of best estimator : \n\n\t{}\n'.format(model.best_params_))


    #  number of cross validation splits
    print('---------------------------------')
    print('|   No of CrossValidation sets   |')
    print('---------------------------------')
```

```
print('-------------------------')
print('\n\tTotal numbre of cross validation sets: {}\n'.format(model.n_splits_))


# Average cross validated score of the best estimator, from the Grid Search
print('--------------------------')
print('|        Best Score       |')
print('--------------------------')
print('\n\tAverage Cross Validate scores of best estimator : \n\n\t{}\n'.format(model.best_score_))
```

1. Logistic Regression with Grid Search

In [142]:

```
# deleting the particular rows in y

y_train.drop([7351], inplace= True)
y_test.drop([2946], inplace= True)
```

In [155]:

```
# get X_train and y_train from csv files
X_train = train.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_train = train['ActivityName']

X_test = test.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_test = test['ActivityName']

print('X_train and y_train : ({},{})'.format(X_train.shape, y_train.shape))
print('X_test  and y_test  : ({},{})'.format(X_test.shape, y_test.shape))
```

```
X_train and y_train : ((7351, 561),(7351,))
X_test  and y_test  : ((2946, 561),(2946,))
```

In [170]:

```
y_test.value_counts()
```

Out[170]:

```
LAYING                537
STANDING              532
WALKING               496
SITTING               491
WALKING_UPSTAIRS      470
WALKING_DOWNSTAIRS    420
Name: ActivityName, dtype: int64
```

In [171]:

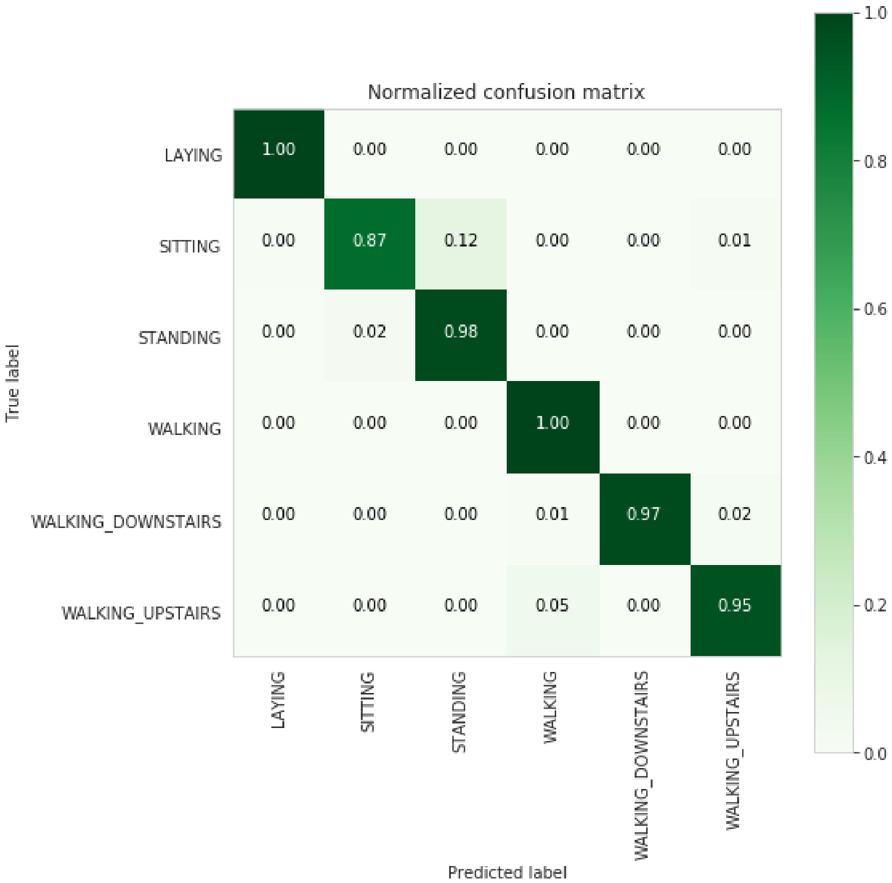
```
# start Grid search
parameters = {'C':[0.0001, 0.001, 0.01, 0.1, 1, 10, 20, 30], 'penalty':['l2','l1']}

labels= ['LAYING', 'SITTING','STANDING','WALKING','WALKING_DOWNSTAIRS','WALKING_UPSTAIRS']
log_reg_grid = GridSearchCV(linear_model.LogisticRegression(), param_grid= parameters, verbose=1, n_jobs=-1)
log_reg_grid_results =  perform_model(log_reg_grid, X_train, y_train, X_test, y_test, class_labels= labels)
```

```
Training the model..
Fitting 3 folds for each of 16 candidates, totalling 48 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 out of  48 | elapsed:   53.1s finished
```

```
Done


training_time(HH:MM:SS.ms) - 0:01:02.688927


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.029037
```

```
--------------------
|      Accuracy      |
--------------------

    0.9626612355736592


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  2 428  57   0   0   4]
 [  0  12 519   1   0   0]
 [  0   0   0 495   1   0]
 [  0   0   0   3 409   8]
 [  0   0   0  22   0 448]]
```




```
-------------------------
| Classification Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.87      0.92       491
          STANDING       0.90      0.98      0.94       532
           WALKING       0.95      1.00      0.97       496
WALKING_DOWNSTAIRS       1.00      0.97      0.99       420
  WALKING_UPSTAIRS       0.97      0.95      0.96       470

          accuracy                           0.96      2946
         macro avg       0.97      0.96      0.96      2946
      weighted avg       0.96      0.96      0.96      2946
```

In [172]:

```python
plt.figure(figsize=(8,8))
plt.grid(b=False)
plot_confusion_matrix(log_reg_grid_results['confusion_matrix'], classes=labels, cmap=plt.cm.Greens, )
plt.show()
```

In [175]:

```
# observe the attributes of the model

print_grid_search_attributes(log_reg_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

 LogisticRegression(C=30, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator :

 {'C': 30, 'penalty': 'l2'}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

 Average Cross Validate scores of best estimator :

 0.946129778261461
```

2. Linear SVC with GridSearch

In [177]:

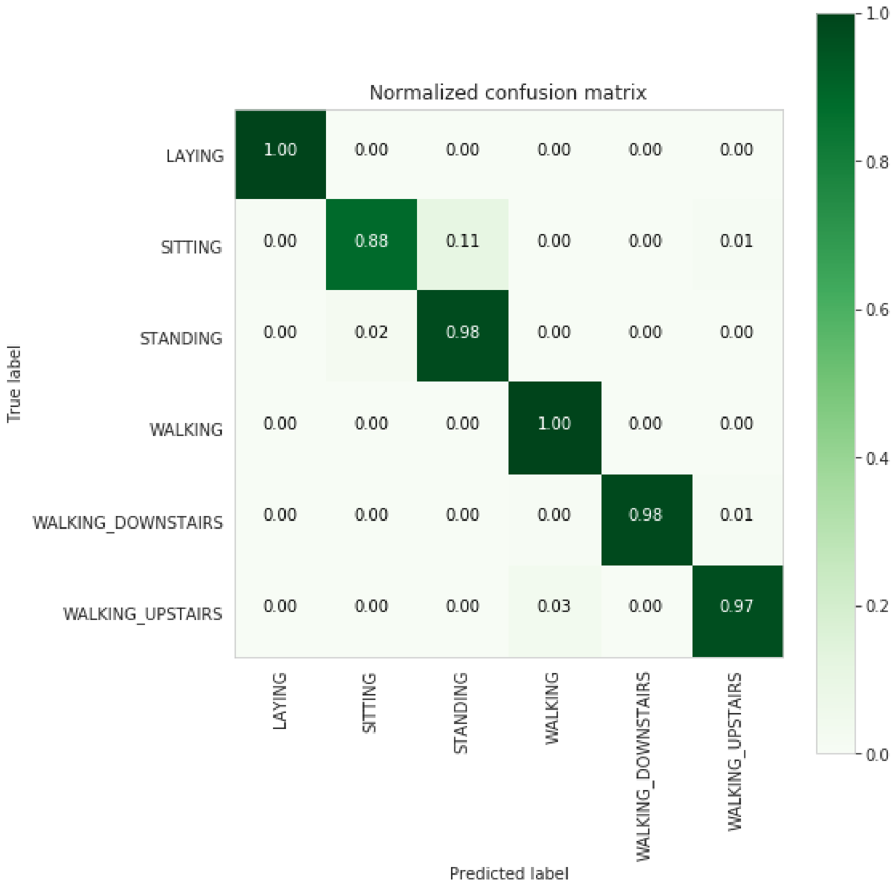
```
parameters = {'C':[0.125, 0.5, 1, 2, 8, 16]}

lr_svc = LinearSVC(tol=0.00005)
lr_svc_grid = GridSearchCV(lr_svc, param_grid=parameters, n_jobs=-1, verbose=1)
```

```
lr_svc_grid_results = perform_model(lr_svc_grid, X_train, y_train, X_test, y_test, class_labels=labels)
```

```
Training the model..
Fitting 3 folds for each of 6 candidates, totalling 18 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed:   15.6s finished
```

```
Done


training_time(HH:MM:SS.ms) - 0:00:19.859823


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.028057


---------------------
|      Accuracy      |
---------------------

    0.9677528852681602


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  2 432  53   0   0   4]
 [  0  12 519   1   0   0]
 [  0   0   0 496   0   0]
 [  0   0   0   2 413   5]
 [  0   0   0  15   1 454]]
```




```
-------------------------
| Classification Report |
-------------------------
                  precision    recall  f1-score   support

          LAYING       1.00      1.00      1.00       537
```

```
           SITTING       0.97      0.88      0.92       491
          STANDING       0.91      0.98      0.94       532
           WALKING       0.96      1.00      0.98       496
WALKING_DOWNSTAIRS       1.00      0.98      0.99       420
  WALKING_UPSTAIRS       0.98      0.97      0.97       470

          accuracy                           0.97      2946
         macro avg       0.97      0.97      0.97      2946
      weighted avg       0.97      0.97      0.97      2946
```

In [178]:

```
print_grid_search_attributes(lr_svc_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

 LinearSVC(C=2, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=5e-05,
          verbose=0)

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator : 

 {'C': 2}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

 Average Cross Validate scores of best estimator : 

 0.946129778261461
```

3. Kernel SVM with GridSearch

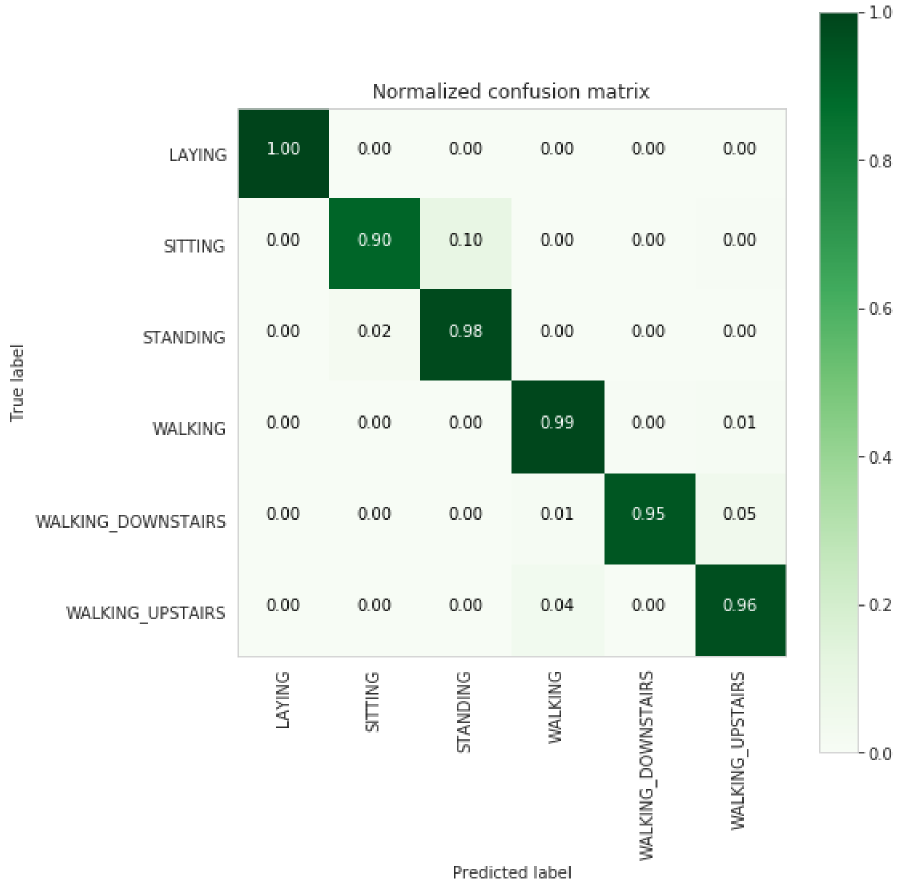
In [180]:

```
parameters = {'C':[2,8,16], 'gamma': [ 0.0078125, 0.125, 2]}

rbf_svm = SVC(kernel='rbf')
rbf_svm_grid = GridSearchCV(rbf_svm,param_grid=parameters, n_jobs=-1)

rbf_svm_grid_results = perform_model(rbf_svm_grid, X_train, y_train, X_test, y_test, class_labels=labels)
```

```
Training the model..
Done 


training_time(HH:MM:SS.ms) - 0:02:07.059750


Predicting test data
Done 


testing time(HH:MM:SS:ms) - 0:00:02.218311


---------------------
|      Accuracy      |
---------------------

    0.9626612355736592
```

| Confusion Matrix |

```
[[537   0   0   0   0   0]
 [  0 441  48   0   0   2]
 [  0  12 520   0   0   0]
 [  0   0   0 489   2   5]
 [  0   0   0   4 397  19]
 [  0   0   0  17   1 452]]
```



| Classification Report |

```
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.90      0.93       491
          STANDING       0.92      0.98      0.95       532
           WALKING       0.96      0.99      0.97       496
WALKING_DOWNSTAIRS       0.99      0.95      0.97       420
  WALKING_UPSTAIRS       0.95      0.96      0.95       470

          accuracy                           0.96      2946
         macro avg       0.96      0.96      0.96      2946
      weighted avg       0.96      0.96      0.96      2946
```

In [181]:

```
print_grid_search_attributes(rbf_svm_grid_results['model'])
```

| Best Estimator |

```
SVC(C=16, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.0078125, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

| Best parameters |

Parameters of best estimator :

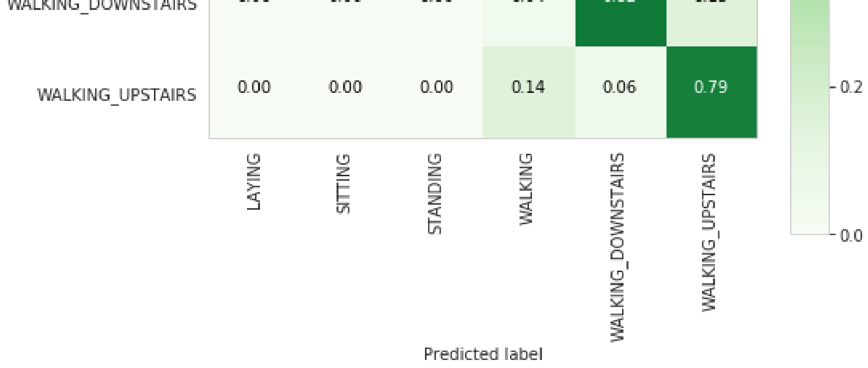
```
{'C': 16, 'gamma': 0.0078125}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

 Average Cross Validate scores of best estimator :

 0.9440892395592436
```

4. Decision Trees with GridSearchCV

In [185]:

```python
parameters = {'max_depth':np.arange(3,10,2)}

dt_grid = GridSearchCV(DecisionTreeClassifier(),param_grid=parameters, n_jobs=-1)
dt_grid_results = perform_model(dt_grid, X_train, y_train, X_test, y_test, class_labels=labels)
```

```
Training the model..
Done


training_time(HH:MM:SS.ms) - 0:00:09.106136


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.004960


---------------------
|      Accuracy      |
---------------------

    0.8652410047522063


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 386 105   0   0   0]
 [  0  93 439   0   0   0]
 [  0   0   0 471  17   8]
 [  0   0   0  15 343  62]
 [  0   0   0  68  29 373]]
```




```
-------------------------
| Classification Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.81      0.79      0.80       491
          STANDING       0.81      0.83      0.82       532
           WALKING       0.85      0.95      0.90       496
WALKING_DOWNSTAIRS       0.88      0.82      0.85       420
  WALKING_UPSTAIRS       0.84      0.79      0.82       470

          accuracy                           0.87      2946
         macro avg       0.86      0.86      0.86      2946
      weighted avg       0.87      0.87      0.86      2946
```

In [186]:

```
print_grid_search_attributes(dt_grid_results['model'])
```

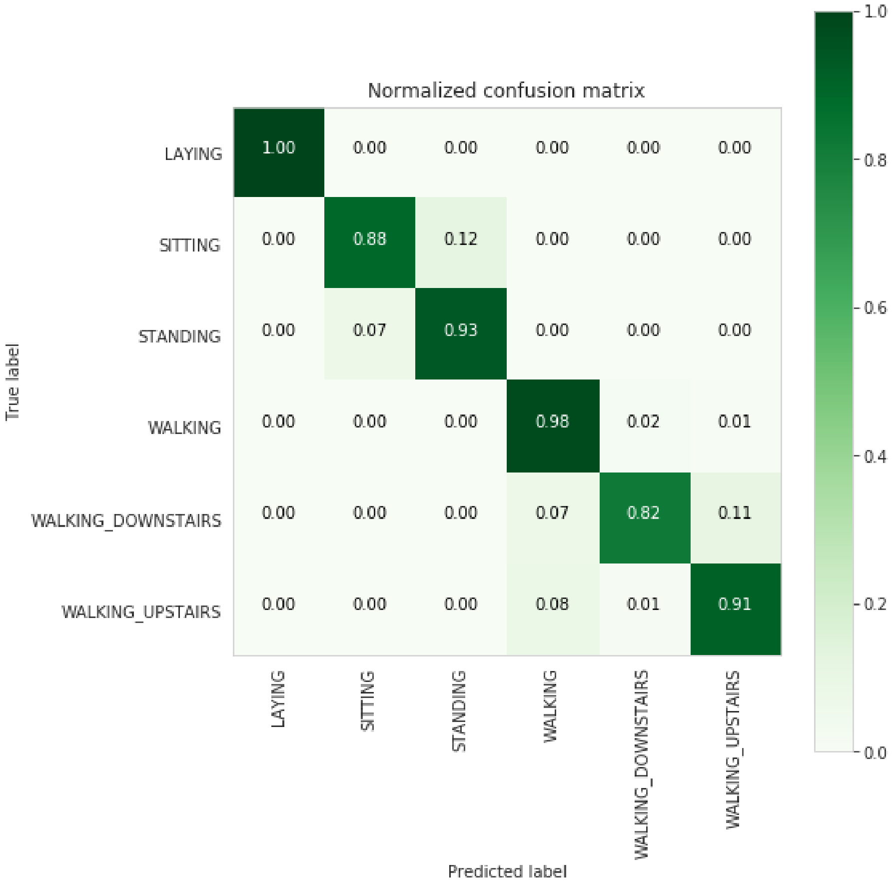
```
--------------------------
|      Best Estimator     |
--------------------------

 DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=7,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator :

 {'max_depth': 7}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

 Average Cross Validate scores of best estimator :

 0.8401578016596382
```

5. Random Forest Classifier with GridSearch

In [188]:

```
params = {'n_estimators': np.arange(10,201,20), 'max_depth':np.arange(3,15,2)}

rfc_grid = GridSearchCV(RandomForestClassifier(), param_grid=params, n_jobs=-1)
rfc_grid_results = perform_model(rfc_grid, X_train, y_train, X_test, y_test, class_labels=labels)
```

```
Training the model..
Done


training_time(HH:MM:SS.ms) - 0:03:08.064735


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.074593


--------------------
|      Accuracy      |
--------------------

    0.923285811269518


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 434  57   0   0   0]
 [  0  37 495   0   0   0]
 [  0   0   0 484   9   3]
 [  0   0   0  29 344  47]
 [  0   0   0  38   6 426]]
```




```
------------------------
| Classification Report |
------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.92      0.88      0.90       491
          STANDING       0.90      0.93      0.91       532
           WALKING       0.88      0.98      0.92       496
WALKING_DOWNSTAIRS       0.96      0.82      0.88       420
  WALKING_UPSTAIRS       0.89      0.91      0.90       470

          accuracy                           0.92      2946
         macro avg       0.92      0.92      0.92      2946
      weighted avg       0.93      0.92      0.92      2946
```

weighted avg   0.93   0.92   0.92   2946

In [189]:

```
print_grid_search_attributes(rfc_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

 RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                        max_depth=9, max_features='auto', max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=150,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False)

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator : 

 {'max_depth': 9, 'n_estimators': 150}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

 Average Cross Validate scores of best estimator : 

 0.9133451231125017
```

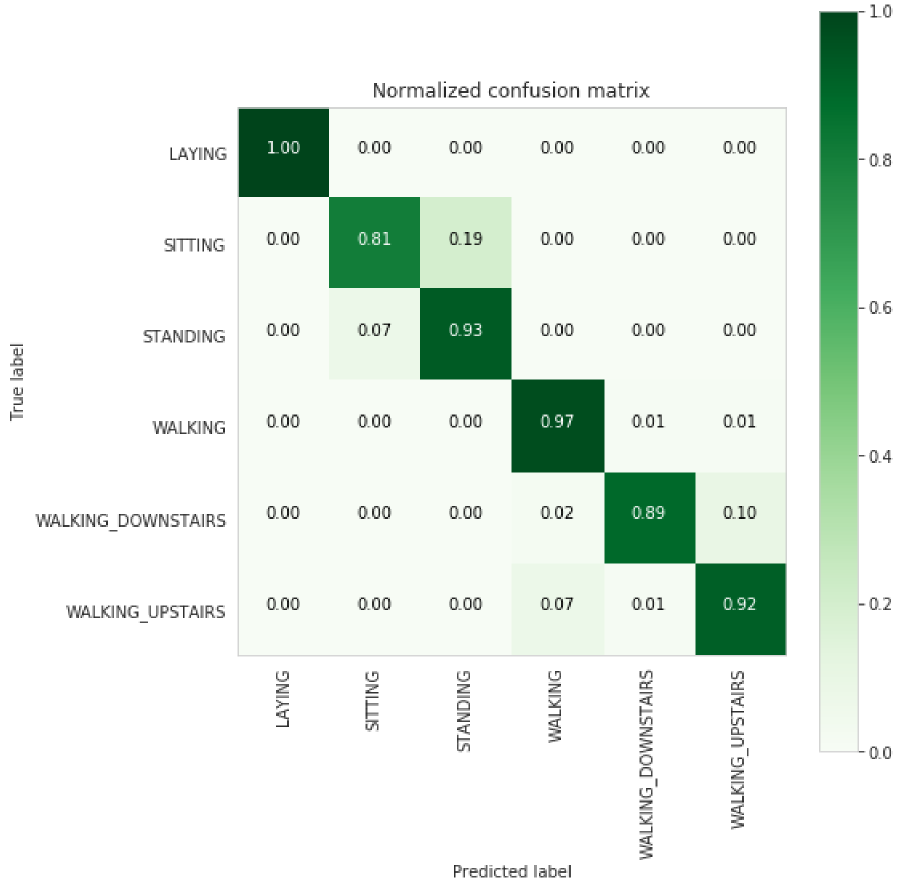
6. Gradient Boosted Decision Trees With GridSearch

In [191]:

```
param_grid = {'max_depth': np.arange(5,8,1), 'n_estimators':np.arange(130,170,10)}

gbdt_grid = GridSearchCV(GradientBoostingClassifier(), param_grid=param_grid, n_jobs=-1)
gbdt_grid_results = perform_model(gbdt_grid, X_train, y_train, X_test, y_test, class_labels=labels)
```

```
Training the model..
Done


training_time(HH:MM:SS.ms) - 0:29:52.221729


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.057789


---------------------
|      Accuracy      |
---------------------

    0.9202308214528174


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 396  94   0   0   1]
 [  0  39 493   0   0   0]
```

```
[ 0   0   0 482   7   7]
[ 0   0   0   7 372  41]
[ 0   1   0  34   4 431]]
```




```
------------------------
| Classification Report |
------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.91      0.81      0.85       491
          STANDING       0.84      0.93      0.88       532
           WALKING       0.92      0.97      0.95       496
WALKING_DOWNSTAIRS       0.97      0.89      0.93       420
  WALKING_UPSTAIRS       0.90      0.92      0.91       470

          accuracy                           0.92      2946
         macro avg       0.92      0.92      0.92      2946
      weighted avg       0.92      0.92      0.92      2946
```

In [192]:

```
print_grid_search_attributes(gbdt_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

 GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=5,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=140,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator :

 {'max_depth': 5, 'n_estimators': 140}
```

```
{'max_depth': 5, 'n_estimators': 140}

---------------------------------
|  No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 3

--------------------------
|       Best Score       |
--------------------------

 Average Cross Validate scores of best estimator : 

 0.9058631478710379
```

7. Comparing all models

In [194]:

```python
print('\n                     Accuracy     Error')
print('                     ----------   --------')
print('Logistic Regression : {:.04}%      {:.04}%'.format(log_reg_grid_results['accuracy'] * 100,\
                                                  100-(log_reg_grid_results['accuracy'] * 100)))

print('Linear SVC          : {:.04}%       {:.04}% '.format(lr_svc_grid_results['accuracy'] * 100,\
                                                        100-(lr_svc_grid_results['accuracy'] * 100)))

print('rbf SVM classifier  : {:.04}%      {:.04}% '.format(rbf_svm_grid_results['accuracy'] * 100,\
                                                          100-(rbf_svm_grid_results['accuracy'] * 100)))

print('DecisionTree        : {:.04}%      {:.04}% '.format(dt_grid_results['accuracy'] * 100,\
                                                        100-(dt_grid_results['accuracy'] * 100)))

print('Random Forest       : {:.04}%      {:.04}% '.format(rfc_grid_results['accuracy'] * 100,\
                                                           100-(rfc_grid_results['accuracy'] * 100)))
print('GradientBoosting DT : {:.04}%      {:.04}% '.format(rfc_grid_results['accuracy'] * 100,\
                                                        100-(rfc_grid_results['accuracy'] * 100)))
```

```
                     Accuracy     Error
                     ----------   --------
Logistic Regression : 96.27%      3.734%
Linear SVC          : 96.78%      3.225% 
rbf SVM classifier  : 96.27%      3.734% 
DecisionTree        : 86.52%      13.48% 
Random Forest       : 92.33%      7.671% 
GradientBoosting DT : 92.33%      7.671% 
```

> We can choose ***Logistic regression*** or ***Linear SVC*** or ***rbf SVM***.

In the real world, domain-knowledge, EDA and feature-engineering matter most.

LSTM

In [2]:

```python
# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
# Triaxial acceleration from the accelerometer is total acceleration

SIGNALS = ["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"]
```

```
# Labelling the classes in y.
label = {0:'WALKING', 1:'WALKING_UPSTAIRS', 2:'WALKING_DOWNSTAIRS', 3:'SITTING', 4:'STANDING', 5:'LAYING'}
```

In [5]:

```
def confusion_matrix(Y_true, Y_pred):
    Y_true = pd.Series([label[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([label[y] for y in np.argmax(Y_pred, axis=1)])

    return pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])

# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the load
def load_signals(subset):
    signals_data = []
    filename = 'body_acc_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_acc_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_acc_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())


    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))


def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = 'y_{}.txt'.format(subset)
    y = _read_csv(filename)[0]

    return pd.get_dummies(y).as_matrix()

def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_test = load_y('train'), load_y('test')

    return X_train, X_test, y_train, y_test
```

In [6]:

```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()

print('X_train shape is: ',X_train.shape)
print('Y_train shape is: ',Y_train.shape)
print('X_test shape is: ',X_test.shape)
print('Y_test shape is: ',Y_test.shape)
```

```
X_train shape is:  (7352, 128, 9)
Y_train shape is:  (7352, 6)
X_test shape is:  (2947, 128, 9)
Y_test shape is:  (2947, 6)
```

In [5]:

```
# Importing tensorflow
np.random.seed(42)
import tensorflow as tf
tf.set_random_seed(42)
```

In [6]:

```
# Configuring a session
session_conf = tf.ConfigProto(intra_op_parallelism_threads= 1,
                              inter_op_parallelism_threads= 1)
```

In [7]:

```
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

In [8]:

```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [9]:

```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)

print(timesteps)
print(input_dim)
print(len(X_train))
```

```
128
9
7352
```

- Defining the Architecture of LSTM and HyperParam Tuning

In [12]:

```
"""

# Initiliazing the sequential model
def best_model(units, activation, dropout_rate, optimizer):
    model = Sequential()
    # Configuring the parameters
    model.add(LSTM(units= units, activation= activation, recurrent_activation='sigmoid', use_bias=True,
             kernel_initializer= 'he_normal', recurrent_initializer='orthogonal', bias_initializer='zeros',
             unit_forget_bias= True, kernel_regularizer= regularizers.l2(0.001), recurrent_regularizer=None,
             bias_regularizer= None, activity_regularizer= None, kernel_constraint=None, recurrent_constraint=None,
             bias_constraint=None, dropout= dropout_rate, recurrent_dropout=0.0, implementation=2,
             return_sequences=False, return_state=False, go_backwards=False, stateful=False, unroll=False,
             input_shape=(timesteps, input_dim)))

    model.add(Dropout(dropout_rate))
    # Adding a dense output layer with sigmoid activation
    model.add(Dense(n_classes, activation='sigmoid'))
    model.compile(loss='categorical_crossentropy', optimizer= optimizer, metrics=['accuracy'])

    return model

"""
```

In [ ]:

```
"""

# https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/

parameters = {'units':        [64, 128],
          'dropout_rate': [0.4, 0.5],
```

```
            'activation':   ['relu', 'sigmoid'],
            'optimizer':    ['rmsprop', 'adam']
            }

model = KerasClassifier(build_fn= best_model, epochs= 30)

gscv = GridSearchCV(estimator = model, param_grid= parameters, n_jobs= -1, )
gscv_result = gscv.fit(X_train, Y_train)

"""
```

In [ ]:

```
"""
print(gscv_result.best_estimator_)
print(gscv_result.best_score_)
"""
```

TRYING VARIOUS ARCHITECTURES

ONE

In [11]:

```
# Initializing parameters
epochs = 100
batch_size = 70
n_hidden = 32
```

In [12]:

```
# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(n_hidden, kernel_initializer= 'he_normal', kernel_regularizer= regularizers.l2(0.001),
          input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))
model.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_1 (LSTM)                (None, 32)                5376      
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 198       
=================================================================
Total params: 5,574
Trainable params: 5,574
Non-trainable params: 0
_________________________________________________________________
```

In [13]:

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

In [14]:

```
# Training the model
model.fit(X_train, Y_train, batch_size=batch_size, validation_data=(X_test, Y_test), epochs=epochs)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow_core/python/ops/math_grad.py:1424: where (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
```

```
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables is deprec
ated. Please use tf.compat.v1.global_variables instead.

Train on 7352 samples, validate on 2947 samples
Epoch 1/100
7352/7352 [==============================] - 26s 3ms/step - loss: 1.8124 - accuracy: 0.3704 - val_loss: 1.5818 - val_accuracy: 0.5616
Epoch 2/100
7352/7352 [==============================] - 22s 3ms/step - loss: 1.4089 - accuracy: 0.5627 - val_loss: 1.2032 - val_accuracy: 0.6223
Epoch 3/100
7352/7352 [==============================] - 22s 3ms/step - loss: 1.2056 - accuracy: 0.5953 - val_loss: 1.3546 - val_accuracy: 0.4795
Epoch 4/100
7352/7352 [==============================] - 22s 3ms/step - loss: 1.1868 - accuracy: 0.5842 - val_loss: 1.1455 - val_accuracy: 0.5762
Epoch 5/100
7352/7352 [==============================] - 22s 3ms/step - loss: 1.0489 - accuracy: 0.6428 - val_loss: 1.0075 - val_accuracy: 0.6515
Epoch 6/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.9469 - accuracy: 0.6534 - val_loss: 0.9429 - val_accuracy: 0.6804
Epoch 7/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.8995 - accuracy: 0.6628 - val_loss: 0.9055 - val_accuracy: 0.6831
Epoch 8/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.8686 - accuracy: 0.6733 - val_loss: 0.8967 - val_accuracy: 0.6485
Epoch 9/100
7352/7352 [==============================] - 23s 3ms/step - loss: 0.8962 - accuracy: 0.6600 - val_loss: 0.8877 - val_accuracy: 0.6440
Epoch 10/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.8285 - accuracy: 0.6774 - val_loss: 0.8673 - val_accuracy: 0.6505
Epoch 11/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.8148 - accuracy: 0.6798 - val_loss: 0.8626 - val_accuracy: 0.6366
Epoch 12/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.7922 - accuracy: 0.6789 - val_loss: 0.8427 - val_accuracy: 0.6529
Epoch 13/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.9901 - accuracy: 0.6196 - val_loss: 0.9305 - val_accuracy: 0.6342
Epoch 14/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.8447 - accuracy: 0.6586 - val_loss: 0.8576 - val_accuracy: 0.6607
Epoch 15/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.8408 - accuracy: 0.6635 - val_loss: 0.8692 - val_accuracy: 0.6386
Epoch 16/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.7951 - accuracy: 0.6766 - val_loss: 0.8400 - val_accuracy: 0.6468
Epoch 17/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.7838 - accuracy: 0.6802 - val_loss: 0.8432 - val_accuracy: 0.6539
Epoch 18/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.8038 - accuracy: 0.6772 - val_loss: 1.1282 - val_accuracy: 0.5684
Epoch 19/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.7984 - accuracy: 0.6817 - val_loss: 0.8324 - val_accuracy: 0.6607
Epoch 20/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.7603 - accuracy: 0.6892 - val_loss: 0.8187 - val_accuracy: 0.6780
Epoch 21/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.7450 - accuracy: 0.6989 - val_loss: 0.8106 - val_accuracy: 0.6824
Epoch 22/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.7337 - accuracy: 0.7050 - val_loss: 0.8018 - val_accuracy: 0.6797
Epoch 23/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.7849 - accuracy: 0.6980 - val_loss: 0.8009 - val_accuracy: 0.6983
Epoch 24/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.7308 - accuracy: 0.7110 - val_loss: 0.7807 - val_accuracy: 0.7160
Epoch 25/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.7125 - accuracy: 0.7251 - val_loss: 0.7742 - val_accuracy: 0.7163
Epoch 26/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.6914 - accuracy: 0.7417 - val_loss: 0.7541 - val_accuracy: 0.7228
Epoch 27/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.6575 - accuracy: 0.7606 - val_loss: 0.7307 - val_accuracy: 0.7458
Epoch 28/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.6313 - accuracy: 0.7675 - val_loss: 0.7103 - val_accuracy: 0.7509
Epoch 29/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.5943 - accuracy: 0.7907 - val_loss: 0.6915 - val_accuracy: 0.7570
Epoch 30/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.5788 - accuracy: 0.8033 - val_loss: 0.6784 - val_accuracy: 0.7615
Epoch 31/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.5540 - accuracy: 0.8175 - val_loss: 0.6481 - val_accuracy: 0.7798
Epoch 32/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.5382 - accuracy: 0.8234 - val_loss: 0.6060 - val_accuracy: 0.7872
Epoch 33/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.5053 - accuracy: 0.8421 - val_loss: 0.6240 - val_accuracy: 0.7872
Epoch 34/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.4695 - accuracy: 0.8554 - val_loss: 0.5625 - val_accuracy: 0.8096
Epoch 35/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.4757 - accuracy: 0.8509 - val_loss: 0.7608 - val_accuracy: 0.7391
Epoch 36/100
7352/7352 [==============================] - 23s 3ms/step - loss: 0.4411 - accuracy: 0.8709 - val_loss: 0.5505 - val_accuracy: 0.8347
Epoch 37/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.4228 - accuracy: 0.8798 - val_loss: 0.5020 - val_accuracy: 0.8432
Epoch 38/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.3764 - accuracy: 0.8996 - val_loss: 0.5481 - val_accuracy: 0.8409
Epoch 39/100
```

```
7352/7352 [==============================] - 22s 3ms/step - loss: 0.3616 - accuracy: 0.9067 - val_loss: 0.6206 - val_accuracy: 0.8202
Epoch 40/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.4180 - accuracy: 0.8911 - val_loss: 0.5625 - val_accuracy: 0.8303
Epoch 41/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.3616 - accuracy: 0.9112 - val_loss: 0.5497 - val_accuracy: 0.8554
Epoch 42/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.3804 - accuracy: 0.9053 - val_loss: 0.4738 - val_accuracy: 0.8575
Epoch 43/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.4428 - accuracy: 0.8840 - val_loss: 0.5099 - val_accuracy: 0.8649
Epoch 44/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.4078 - accuracy: 0.8954 - val_loss: 0.5439 - val_accuracy: 0.8429
Epoch 45/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.3850 - accuracy: 0.9015 - val_loss: 0.4243 - val_accuracy: 0.8833
Epoch 46/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.3182 - accuracy: 0.9257 - val_loss: 0.4284 - val_accuracy: 0.8819
Epoch 47/100
7352/7352 [==============================] - 23s 3ms/step - loss: 0.3299 - accuracy: 0.9223 - val_loss: 0.5850 - val_accuracy: 0.8202
Epoch 48/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.3575 - accuracy: 0.9102 - val_loss: 0.4485 - val_accuracy: 0.8778
Epoch 49/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.3074 - accuracy: 0.9244 - val_loss: 0.4829 - val_accuracy: 0.8697
Epoch 50/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2886 - accuracy: 0.9291 - val_loss: 0.4674 - val_accuracy: 0.8775
Epoch 51/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2719 - accuracy: 0.9363 - val_loss: 0.5100 - val_accuracy: 0.8633
Epoch 52/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2782 - accuracy: 0.9320 - val_loss: 0.4418 - val_accuracy: 0.8812
Epoch 53/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2621 - accuracy: 0.9329 - val_loss: 0.4193 - val_accuracy: 0.8911
Epoch 54/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2464 - accuracy: 0.9393 - val_loss: 0.4586 - val_accuracy: 0.8873
Epoch 55/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2435 - accuracy: 0.9395 - val_loss: 0.5382 - val_accuracy: 0.8748
Epoch 56/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2423 - accuracy: 0.9399 - val_loss: 0.4598 - val_accuracy: 0.8877
Epoch 57/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2388 - accuracy: 0.9389 - val_loss: 0.5739 - val_accuracy: 0.8415
Epoch 58/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.3130 - accuracy: 0.9165 - val_loss: 0.5043 - val_accuracy: 0.8612
Epoch 59/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2363 - accuracy: 0.9406 - val_loss: 0.4687 - val_accuracy: 0.8873
Epoch 60/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2654 - accuracy: 0.9266 - val_loss: 0.4306 - val_accuracy: 0.8935
Epoch 61/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2333 - accuracy: 0.9393 - val_loss: 0.4461 - val_accuracy: 0.8860
Epoch 62/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2222 - accuracy: 0.9427 - val_loss: 0.4169 - val_accuracy: 0.8863
Epoch 63/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2127 - accuracy: 0.9457 - val_loss: 0.4784 - val_accuracy: 0.8911
Epoch 64/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2813 - accuracy: 0.9255 - val_loss: 0.5611 - val_accuracy: 0.8537
Epoch 65/100
7352/7352 [==============================] - 23s 3ms/step - loss: 0.2370 - accuracy: 0.9391 - val_loss: 0.5156 - val_accuracy: 0.8653
Epoch 66/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2869 - accuracy: 0.9245 - val_loss: 0.4903 - val_accuracy: 0.8833
Epoch 67/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2524 - accuracy: 0.9338 - val_loss: 0.4734 - val_accuracy: 0.8755
Epoch 68/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2434 - accuracy: 0.9365 - val_loss: 0.3621 - val_accuracy: 0.9077
Epoch 69/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2098 - accuracy: 0.9446 - val_loss: 0.4156 - val_accuracy: 0.8999
Epoch 70/100
7352/7352 [==============================] - 23s 3ms/step - loss: 0.2059 - accuracy: 0.9480 - val_loss: 0.4653 - val_accuracy: 0.9070
Epoch 71/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.4911 - accuracy: 0.8558 - val_loss: 0.5397 - val_accuracy: 0.8521
Epoch 72/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.3542 - accuracy: 0.9007 - val_loss: 0.4428 - val_accuracy: 0.8873
Epoch 73/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.3139 - accuracy: 0.9180 - val_loss: 0.4176 - val_accuracy: 0.8873
Epoch 74/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2350 - accuracy: 0.9416 - val_loss: 0.4409 - val_accuracy: 0.8955
Epoch 75/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2262 - accuracy: 0.9465 - val_loss: 0.4739 - val_accuracy: 0.8823
Epoch 76/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2228 - accuracy: 0.9425 - val_loss: 0.4083 - val_accuracy: 0.8924
Epoch 77/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2161 - accuracy: 0.9463 - val_loss: 0.4039 - val_accuracy: 0.8945
Epoch 78/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2036 - accuracy: 0.9480 - val_loss: 0.4591 - val_accuracy: 0.8935
Epoch 79/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2017 - accuracy: 0.9482 - val_loss: 0.5514 - val_accuracy: 0.8853
Epoch 80/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2039 - accuracy: 0.9468 - val_loss: 0.5245 - val_accuracy: 0.8846
```

```
Epoch 81/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2001 - accuracy: 0.9497 - val_loss: 0.4733 - val_accuracy: 0.8972
Epoch 82/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2080 - accuracy: 0.9460 - val_loss: 0.4145 - val_accuracy: 0.8958
Epoch 83/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2697 - accuracy: 0.9233 - val_loss: 0.4511 - val_accuracy: 0.8843
Epoch 84/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2117 - accuracy: 0.9455 - val_loss: 0.4487 - val_accuracy: 0.8867
Epoch 85/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1915 - accuracy: 0.9464 - val_loss: 0.5055 - val_accuracy: 0.8965
Epoch 86/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1888 - accuracy: 0.9493 - val_loss: 0.4714 - val_accuracy: 0.8945
Epoch 87/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1917 - accuracy: 0.9442 - val_loss: 0.4936 - val_accuracy: 0.8979
Epoch 88/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1882 - accuracy: 0.9474 - val_loss: 0.4921 - val_accuracy: 0.8985
Epoch 89/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2437 - accuracy: 0.9363 - val_loss: 0.4782 - val_accuracy: 0.8870
Epoch 90/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1972 - accuracy: 0.9472 - val_loss: 0.4727 - val_accuracy: 0.8985
Epoch 91/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1878 - accuracy: 0.9504 - val_loss: 0.4704 - val_accuracy: 0.8962
Epoch 92/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.2061 - accuracy: 0.9464 - val_loss: 0.4528 - val_accuracy: 0.9033
Epoch 93/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1814 - accuracy: 0.9520 - val_loss: 0.4850 - val_accuracy: 0.9013
Epoch 94/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1813 - accuracy: 0.9502 - val_loss: 0.4627 - val_accuracy: 0.8921
Epoch 95/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1837 - accuracy: 0.9502 - val_loss: 0.4821 - val_accuracy: 0.8985
Epoch 96/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1730 - accuracy: 0.9535 - val_loss: 0.4960 - val_accuracy: 0.9006
Epoch 97/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1782 - accuracy: 0.9460 - val_loss: 0.5098 - val_accuracy: 0.8975
Epoch 98/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1733 - accuracy: 0.9544 - val_loss: 0.5519 - val_accuracy: 0.8962
Epoch 99/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1955 - accuracy: 0.9471 - val_loss: 0.5430 - val_accuracy: 0.8951
Epoch 100/100
7352/7352 [==============================] - 22s 3ms/step - loss: 0.1948 - accuracy: 0.9508 - val_loss: 0.5094 - val_accuracy: 0.9026
```

Out[14]:

```
<keras.callbacks.callbacks.History at 0x7f1de87e4ba8>
```

In [15]:

```python
# Confusion Matrix
print(confusion_matrix(Y_test, model.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 537        0         0        0                   0
SITTING                  0      408        57        0                   0
STANDING                 0      101       429        2                   0
WALKING                  0        1         0      452                  41
WALKING_DOWNSTAIRS       0        0         0        1                 411
WALKING_UPSTAIRS         0        2         1       15                  30

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                           26
STANDING                           0
WALKING                            2
WALKING_DOWNSTAIRS                 8
WALKING_UPSTAIRS                 423
```

In [16]:

```python
score = model.evaluate(X_test, Y_test)
```

```
2947/2947 [==============================] - 2s 699us/step
```

In [17]:

```python
score
```

Out[17]:

[0.5093596800744614, 0.9026128053665161]

TWO

In [11]:

```
# Initializing parameters
epochs = 100
batch_size = 70
n_hidden = 64
```

In [12]:

```
# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(n_hidden, kernel_initializer= 'he_normal', kernel_regularizer= regularizers.l2(0.001),
          input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))
model.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling BaseResource
Variable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 64)                18944
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 390
=================================================================
Total params: 19,334
Trainable params: 19,334
Non-trainable params: 0
_________________________________________________________________
```

In [13]:

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

In [14]:

```
# Training the model
model.fit(X_train, Y_train, batch_size=batch_size, validation_data=(X_test, Y_test), epochs=epochs)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow_core/python/ops/math_grad.py:1424: where (from tensorflow.python.op
s.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables is deprec
ated. Please use tf.compat.v1.global_variables instead.

Train on 7352 samples, validate on 2947 samples
Epoch 1/100
7352/7352 [==============================] - 31s 4ms/step - loss: 1.7981 - accuracy: 0.4913 - val_loss: 1.4045 - val_accuracy: 0.5769
Epoch 2/100
7352/7352 [==============================] - 27s 4ms/step - loss: 1.1737 - accuracy: 0.6666 - val_loss: 1.1122 - val_accuracy: 0.6420
Epoch 3/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.9348 - accuracy: 0.7304 - val_loss: 0.9110 - val_accuracy: 0.7180
Epoch 4/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.8217 - accuracy: 0.7598 - val_loss: 0.9339 - val_accuracy: 0.6926
Epoch 5/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.7525 - accuracy: 0.7637 - val_loss: 0.8540 - val_accuracy: 0.6817
Epoch 6/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.7081 - accuracy: 0.7560 - val_loss: 0.7580 - val_accuracy: 0.7316
```

```
Epoch 7/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.6447 - accuracy: 0.7992 - val_loss: 0.6993 - val_accuracy: 0.7699
Epoch 8/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.5607 - accuracy: 0.8345 - val_loss: 0.7088 - val_accuracy: 0.8035
Epoch 9/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.4728 - accuracy: 0.8772 - val_loss: 0.6277 - val_accuracy: 0.8327
Epoch 10/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.5599 - accuracy: 0.8555 - val_loss: 0.6565 - val_accuracy: 0.8174
Epoch 11/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.4231 - accuracy: 0.9055 - val_loss: 0.5969 - val_accuracy: 0.8473
Epoch 12/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.5032 - accuracy: 0.8474 - val_loss: 0.6048 - val_accuracy: 0.8090
Epoch 13/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.4823 - accuracy: 0.8708 - val_loss: 0.6599 - val_accuracy: 0.8371
Epoch 14/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.4543 - accuracy: 0.8946 - val_loss: 0.5582 - val_accuracy: 0.8412
Epoch 15/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.3791 - accuracy: 0.9121 - val_loss: 0.5060 - val_accuracy: 0.8575
Epoch 16/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.4247 - accuracy: 0.8836 - val_loss: 0.5537 - val_accuracy: 0.8341
Epoch 17/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.3631 - accuracy: 0.9211 - val_loss: 0.4467 - val_accuracy: 0.8721
Epoch 18/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.3020 - accuracy: 0.9319 - val_loss: 0.5258 - val_accuracy: 0.8670
Epoch 19/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.3067 - accuracy: 0.9287 - val_loss: 0.3790 - val_accuracy: 0.8999
Epoch 20/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.2761 - accuracy: 0.9361 - val_loss: 0.4268 - val_accuracy: 0.8823
Epoch 21/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.2751 - accuracy: 0.9339 - val_loss: 0.4695 - val_accuracy: 0.8887
Epoch 22/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.3347 - accuracy: 0.9083 - val_loss: 0.5531 - val_accuracy: 0.8738
Epoch 23/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.2896 - accuracy: 0.9289 - val_loss: 0.4332 - val_accuracy: 0.8955
Epoch 24/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2638 - accuracy: 0.9376 - val_loss: 0.4461 - val_accuracy: 0.8924
Epoch 25/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2594 - accuracy: 0.9319 - val_loss: 0.4884 - val_accuracy: 0.8962
Epoch 26/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2626 - accuracy: 0.9378 - val_loss: 0.4177 - val_accuracy: 0.8795
Epoch 27/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2492 - accuracy: 0.9415 - val_loss: 0.3737 - val_accuracy: 0.9036
Epoch 28/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2819 - accuracy: 0.9270 - val_loss: 0.4013 - val_accuracy: 0.8761
Epoch 29/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2453 - accuracy: 0.9422 - val_loss: 0.3830 - val_accuracy: 0.9074
Epoch 30/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2260 - accuracy: 0.9478 - val_loss: 0.4021 - val_accuracy: 0.8975
Epoch 31/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2207 - accuracy: 0.9480 - val_loss: 0.4470 - val_accuracy: 0.8928
Epoch 32/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.2357 - accuracy: 0.9471 - val_loss: 0.3625 - val_accuracy: 0.9013
Epoch 33/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.2149 - accuracy: 0.9471 - val_loss: 0.3322 - val_accuracy: 0.9070
Epoch 34/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2320 - accuracy: 0.9425 - val_loss: 0.3524 - val_accuracy: 0.9016
Epoch 35/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.2738 - accuracy: 0.9327 - val_loss: 0.4189 - val_accuracy: 0.8965
Epoch 36/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.2203 - accuracy: 0.9453 - val_loss: 0.4372 - val_accuracy: 0.8945
Epoch 37/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2225 - accuracy: 0.9448 - val_loss: 0.3704 - val_accuracy: 0.8826
Epoch 38/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2050 - accuracy: 0.9499 - val_loss: 0.3852 - val_accuracy: 0.9080
Epoch 39/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.3395 - accuracy: 0.8998 - val_loss: 0.4365 - val_accuracy: 0.8880
Epoch 40/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.2663 - accuracy: 0.9327 - val_loss: 0.3837 - val_accuracy: 0.8850
Epoch 41/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2307 - accuracy: 0.9410 - val_loss: 0.4023 - val_accuracy: 0.8958
Epoch 42/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2113 - accuracy: 0.9453 - val_loss: 0.3575 - val_accuracy: 0.8979
Epoch 43/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1968 - accuracy: 0.9480 - val_loss: 0.3731 - val_accuracy: 0.9118
Epoch 44/100
7352/7352 [==============================] - 27s 4ms/step - loss: 0.2017 - accuracy: 0.9491 - val_loss: 0.3737 - val_accuracy: 0.9121
Epoch 45/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1896 - accuracy: 0.9501 - val_loss: 0.4282 - val_accuracy: 0.9084
Epoch 46/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1843 - accuracy: 0.9474 - val_loss: 0.3933 - val_accuracy: 0.9111
Epoch 47/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1827 - accuracy: 0.9480 - val_loss: 0.4023 - val_accuracy: 0.9155
Epoch 48/100
```

```
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2158 - accuracy: 0.9411 - val_loss: 0.6683 - val_accuracy: 0.7570
Epoch 49/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2376 - accuracy: 0.9359 - val_loss: 0.3254 - val_accuracy: 0.9070
Epoch 50/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1866 - accuracy: 0.9514 - val_loss: 0.3984 - val_accuracy: 0.9155
Epoch 51/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1888 - accuracy: 0.9487 - val_loss: 0.3609 - val_accuracy: 0.8958
Epoch 52/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1887 - accuracy: 0.9463 - val_loss: 0.4337 - val_accuracy: 0.8931
Epoch 53/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2065 - accuracy: 0.9472 - val_loss: 0.3522 - val_accuracy: 0.9192
Epoch 54/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1846 - accuracy: 0.9517 - val_loss: 0.3786 - val_accuracy: 0.9141
Epoch 55/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1853 - accuracy: 0.9517 - val_loss: 0.3644 - val_accuracy: 0.9182
Epoch 56/100
7352/7352 [==============================] - 29s 4ms/step - loss: 0.1691 - accuracy: 0.9542 - val_loss: 0.3233 - val_accuracy: 0.9189
Epoch 57/100
7352/7352 [==============================] - 29s 4ms/step - loss: 0.1751 - accuracy: 0.9505 - val_loss: 0.3713 - val_accuracy: 0.9148
Epoch 58/100
7352/7352 [==============================] - 29s 4ms/step - loss: 0.1671 - accuracy: 0.9523 - val_loss: 0.3567 - val_accuracy: 0.9158
Epoch 59/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1598 - accuracy: 0.9531 - val_loss: 0.3690 - val_accuracy: 0.9206
Epoch 60/100
7352/7352 [==============================] - 29s 4ms/step - loss: 0.1632 - accuracy: 0.9527 - val_loss: 0.3921 - val_accuracy: 0.9145
Epoch 61/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1784 - accuracy: 0.9489 - val_loss: 0.2798 - val_accuracy: 0.9125
Epoch 62/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1967 - accuracy: 0.9502 - val_loss: 0.3477 - val_accuracy: 0.9036
Epoch 63/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1700 - accuracy: 0.9521 - val_loss: 0.4239 - val_accuracy: 0.9070
Epoch 64/100
7352/7352 [==============================] - 29s 4ms/step - loss: 0.5792 - accuracy: 0.8432 - val_loss: 0.5253 - val_accuracy: 0.8578
Epoch 65/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.4797 - accuracy: 0.8685 - val_loss: 0.3511 - val_accuracy: 0.8894
Epoch 66/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2471 - accuracy: 0.9350 - val_loss: 0.4004 - val_accuracy: 0.8931
Epoch 67/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2100 - accuracy: 0.9419 - val_loss: 0.3443 - val_accuracy: 0.9006
Epoch 68/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1889 - accuracy: 0.9502 - val_loss: 0.3947 - val_accuracy: 0.8975
Epoch 69/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.3049 - accuracy: 0.9072 - val_loss: 0.3735 - val_accuracy: 0.8982
Epoch 70/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2186 - accuracy: 0.9422 - val_loss: 0.3352 - val_accuracy: 0.9036
Epoch 71/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1913 - accuracy: 0.9491 - val_loss: 0.3193 - val_accuracy: 0.9186
Epoch 72/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1766 - accuracy: 0.9536 - val_loss: 0.3318 - val_accuracy: 0.9172
Epoch 73/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1758 - accuracy: 0.9520 - val_loss: 0.3162 - val_accuracy: 0.9226
Epoch 74/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1724 - accuracy: 0.9533 - val_loss: 0.3199 - val_accuracy: 0.9158
Epoch 75/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1992 - accuracy: 0.9431 - val_loss: 0.4346 - val_accuracy: 0.8924
Epoch 76/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1777 - accuracy: 0.9521 - val_loss: 0.3118 - val_accuracy: 0.9172
Epoch 77/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1640 - accuracy: 0.9550 - val_loss: 0.3360 - val_accuracy: 0.9070
Epoch 78/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.7287 - accuracy: 0.8625 - val_loss: 1.1845 - val_accuracy: 0.6902
Epoch 79/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.5488 - accuracy: 0.8415 - val_loss: 0.5476 - val_accuracy: 0.8500
Epoch 80/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2766 - accuracy: 0.9279 - val_loss: 0.3959 - val_accuracy: 0.8880
Epoch 81/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2214 - accuracy: 0.9425 - val_loss: 0.3651 - val_accuracy: 0.8962
Epoch 82/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1994 - accuracy: 0.9455 - val_loss: 0.3329 - val_accuracy: 0.9057
Epoch 83/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1904 - accuracy: 0.9456 - val_loss: 0.3423 - val_accuracy: 0.8992
Epoch 84/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1902 - accuracy: 0.9438 - val_loss: 0.3404 - val_accuracy: 0.9040
Epoch 85/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1920 - accuracy: 0.9446 - val_loss: 0.3265 - val_accuracy: 0.9111
Epoch 86/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1829 - accuracy: 0.9465 - val_loss: 0.3360 - val_accuracy: 0.9057
Epoch 87/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2856 - accuracy: 0.9109 - val_loss: 0.6701 - val_accuracy: 0.7224
Epoch 88/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2913 - accuracy: 0.9059 - val_loss: 0.4094 - val_accuracy: 0.9002
Epoch 89/100
```

```
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2203 - accuracy: 0.9410 - val_loss: 0.3877 - val_accuracy: 0.8958
Epoch 90/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1981 - accuracy: 0.9457 - val_loss: 0.3746 - val_accuracy: 0.8975
Epoch 91/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1880 - accuracy: 0.9445 - val_loss: 0.3982 - val_accuracy: 0.8975
Epoch 92/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1935 - accuracy: 0.9468 - val_loss: 0.3602 - val_accuracy: 0.9009
Epoch 93/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2019 - accuracy: 0.9437 - val_loss: 0.7694 - val_accuracy: 0.7981
Epoch 94/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.3109 - accuracy: 0.9089 - val_loss: 0.3008 - val_accuracy: 0.9216
Epoch 95/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1851 - accuracy: 0.9494 - val_loss: 0.3294 - val_accuracy: 0.9158
Epoch 96/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1682 - accuracy: 0.9505 - val_loss: 0.3314 - val_accuracy: 0.9135
Epoch 97/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.2063 - accuracy: 0.9425 - val_loss: 0.3316 - val_accuracy: 0.9111
Epoch 98/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1708 - accuracy: 0.9504 - val_loss: 0.3345 - val_accuracy: 0.9070
Epoch 99/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1706 - accuracy: 0.9531 - val_loss: 0.4320 - val_accuracy: 0.9138
Epoch 100/100
7352/7352 [==============================] - 28s 4ms/step - loss: 0.1702 - accuracy: 0.9514 - val_loss: 0.3749 - val_accuracy: 0.8965
```

Out[14]:

```
<keras.callbacks.callbacks.History at 0x7f5fba1cfc18>
```

In [15]:

```
# Confusion Matrix
print(confusion_matrix(Y_test, model.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0         0        0                   0
SITTING                  0      414        60        0                   0
STANDING                 0      125       402        1                   0
WALKING                  0        0         0      465                  30
WALKING_DOWNSTAIRS       0        0         0        1                 414
WALKING_UPSTAIRS         0        0         0       21                  13

Pred                WALKING_UPSTAIRS
True
LAYING                            27
SITTING                           17
STANDING                           4
WALKING                            1
WALKING_DOWNSTAIRS                 5
WALKING_UPSTAIRS                 437
```

In [16]:

```
score = model.evaluate(X_test, Y_test)
```

```
2947/2947 [==============================] - 3s 880us/step
```

In [17]:

```
score
```

Out[17]:

```
[0.3748672223941609, 0.8965049386024475]
```

THREE

In [11]:

```
# Initializing parameters
epochs = 50
batch_size = 70
n_hidden1 = 32
n_hidden2 = 64
```

In [12]:

```
# https://machinelearningmastery.com/stacked-long-short-term-memory-networks/
# Initiliazing the sequential model
model = Sequential()

# Configuring the parameters
model.add(LSTM(n_hidden1, kernel_initializer= 'he_normal', kernel_regularizer= regularizers.l2(0.001),
          return_sequences=True, input_shape=(timesteps, input_dim)))
model.add(LSTM(n_hidden2, kernel_initializer= 'he_normal', kernel_regularizer= regularizers.l2(0.001),
          input_shape=(timesteps, input_dim)))

# Adding a dropout layer
model.add(Dropout(0.5))

# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))
model.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling BaseResource
Variable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 128, 32)           5376
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                24832
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 390
=================================================================
Total params: 30,598
Trainable params: 30,598
Non-trainable params: 0
_________________________________________________________________
```

In [13]:

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

In [14]:

```
# Training the model
model.fit(X_train, Y_train, batch_size=batch_size, validation_data=(X_test, Y_test), epochs=epochs)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow_core/python/ops/math_grad.py:1424: where (from tensorflow.python.op
s.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables is deprec
ated. Please use tf.compat.v1.global_variables instead.

Train on 7352 samples, validate on 2947 samples
Epoch 1/50
7352/7352 [==============================] - 59s 8ms/step - loss: 1.7838 - accuracy: 0.5563 - val_loss: 1.3617 - val_accuracy: 0.6169
Epoch 2/50
7352/7352 [==============================] - 54s 7ms/step - loss: 1.1699 - accuracy: 0.7043 - val_loss: 1.1542 - val_accuracy: 0.6926
Epoch 3/50
7352/7352 [==============================] - 55s 8ms/step - loss: 0.8993 - accuracy: 0.8112 - val_loss: 0.9296 - val_accuracy: 0.7777
Epoch 4/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.6931 - accuracy: 0.8781 - val_loss: 0.7331 - val_accuracy: 0.8656
Epoch 5/50
7352/7352 [==============================] - 55s 8ms/step - loss: 0.5502 - accuracy: 0.9166 - val_loss: 0.7542 - val_accuracy: 0.8507
Epoch 6/50
7352/7352 [==============================] - 56s 8ms/step - loss: 0.4851 - accuracy: 0.9215 - val_loss: 0.7129 - val_accuracy: 0.8568
Epoch 7/50
7352/7352 [==============================] - 56s 8ms/step - loss: 0.4335 - accuracy: 0.9339 - val_loss: 0.6954 - val_accuracy: 0.8585
Epoch 8/50
7352/7352 [==============================] - 56s 8ms/step - loss: 0.4120 - accuracy: 0.9334 - val_loss: 0.6404 - val_accuracy: 0.8554
Epoch 9/50
7352/7352 [==============================] - 55s 8ms/step - loss: 0.3757 - accuracy: 0.9421 - val_loss: 0.6217 - val_accuracy: 0.8697
Epoch 10/50
7352/7352 [==============================] - 55s 8ms/step - loss: 0.3829 - accuracy: 0.9384 - val_loss: 0.7423 - val_accuracy: 0.8588
Epoch 11/50
```

```
7352/7352 [==============================] - 56s 8ms/step - loss: 0.3861 - accuracy: 0.9350 - val_loss: 0.5878 - val_accuracy: 0.8846
Epoch 12/50
7352/7352 [==============================] - 54s 7ms/step - loss: 0.3509 - accuracy: 0.9382 - val_loss: 0.6650 - val_accuracy: 0.8588
Epoch 13/50
7352/7352 [==============================] - 54s 7ms/step - loss: 0.3528 - accuracy: 0.9369 - val_loss: 0.5794 - val_accuracy: 0.8792
Epoch 14/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.3304 - accuracy: 0.9410 - val_loss: 0.6931 - val_accuracy: 0.8663
Epoch 15/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.3231 - accuracy: 0.9411 - val_loss: 0.5938 - val_accuracy: 0.8819
Epoch 16/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.2935 - accuracy: 0.9463 - val_loss: 0.7189 - val_accuracy: 0.8398
Epoch 17/50
7352/7352 [==============================] - 55s 8ms/step - loss: 0.3111 - accuracy: 0.9421 - val_loss: 0.5724 - val_accuracy: 0.8833
Epoch 18/50
7352/7352 [==============================] - 56s 8ms/step - loss: 0.2794 - accuracy: 0.9493 - val_loss: 0.5634 - val_accuracy: 0.8758
Epoch 19/50
7352/7352 [==============================] - 56s 8ms/step - loss: 0.2794 - accuracy: 0.9408 - val_loss: 0.5989 - val_accuracy: 0.8789
Epoch 20/50
7352/7352 [==============================] - 56s 8ms/step - loss: 0.2545 - accuracy: 0.9525 - val_loss: 0.5806 - val_accuracy: 0.8853
Epoch 21/50
7352/7352 [==============================] - 58s 8ms/step - loss: 0.2646 - accuracy: 0.9374 - val_loss: 0.4922 - val_accuracy: 0.8982
Epoch 22/50
7352/7352 [==============================] - 57s 8ms/step - loss: 0.2489 - accuracy: 0.9504 - val_loss: 0.5275 - val_accuracy: 0.8935
Epoch 23/50
7352/7352 [==============================] - 57s 8ms/step - loss: 0.2489 - accuracy: 0.9498 - val_loss: 0.4629 - val_accuracy: 0.8979
Epoch 24/50
7352/7352 [==============================] - 56s 8ms/step - loss: 0.2352 - accuracy: 0.9539 - val_loss: 0.4702 - val_accuracy: 0.8948
Epoch 25/50
7352/7352 [==============================] - 57s 8ms/step - loss: 0.2627 - accuracy: 0.9510 - val_loss: 0.4822 - val_accuracy: 0.9060
Epoch 26/50
7352/7352 [==============================] - 56s 8ms/step - loss: 0.2685 - accuracy: 0.9321 - val_loss: 0.5105 - val_accuracy: 0.8877
Epoch 27/50
7352/7352 [==============================] - 55s 8ms/step - loss: 0.2975 - accuracy: 0.9334 - val_loss: 0.5275 - val_accuracy: 0.8914
Epoch 28/50
7352/7352 [==============================] - 56s 8ms/step - loss: 0.2584 - accuracy: 0.9446 - val_loss: 0.6764 - val_accuracy: 0.8534
Epoch 29/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.2348 - accuracy: 0.9504 - val_loss: 0.4996 - val_accuracy: 0.8989
Epoch 30/50
7352/7352 [==============================] - 55s 8ms/step - loss: 0.2196 - accuracy: 0.9516 - val_loss: 0.5189 - val_accuracy: 0.8962
Epoch 31/50
7352/7352 [==============================] - 55s 8ms/step - loss: 0.2180 - accuracy: 0.9476 - val_loss: 0.5321 - val_accuracy: 0.8979
Epoch 32/50
7352/7352 [==============================] - 55s 8ms/step - loss: 0.2076 - accuracy: 0.9565 - val_loss: 0.5686 - val_accuracy: 0.8958
Epoch 33/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.2066 - accuracy: 0.9553 - val_loss: 0.5395 - val_accuracy: 0.8931
Epoch 34/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.1986 - accuracy: 0.9584 - val_loss: 0.5370 - val_accuracy: 0.8941
Epoch 35/50
7352/7352 [==============================] - 54s 7ms/step - loss: 0.2041 - accuracy: 0.9517 - val_loss: 0.5081 - val_accuracy: 0.8972
Epoch 36/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.2072 - accuracy: 0.9512 - val_loss: 0.5495 - val_accuracy: 0.8870
Epoch 37/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.3030 - accuracy: 0.9248 - val_loss: 0.7116 - val_accuracy: 0.8755
Epoch 38/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.2253 - accuracy: 0.9514 - val_loss: 0.6346 - val_accuracy: 0.8853
Epoch 39/50
7352/7352 [==============================] - 55s 8ms/step - loss: 0.2072 - accuracy: 0.9528 - val_loss: 0.6635 - val_accuracy: 0.8863
Epoch 40/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.2014 - accuracy: 0.9536 - val_loss: 0.5990 - val_accuracy: 0.8962
Epoch 41/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.1889 - accuracy: 0.9563 - val_loss: 0.5606 - val_accuracy: 0.9080
Epoch 42/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.1948 - accuracy: 0.9557 - val_loss: 0.7855 - val_accuracy: 0.8653
Epoch 43/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.2077 - accuracy: 0.9509 - val_loss: 0.6252 - val_accuracy: 0.8918
Epoch 44/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.2072 - accuracy: 0.9510 - val_loss: 0.5330 - val_accuracy: 0.8962
Epoch 45/50
7352/7352 [==============================] - 56s 8ms/step - loss: 0.1860 - accuracy: 0.9573 - val_loss: 0.5225 - val_accuracy: 0.9016
Epoch 46/50
7352/7352 [==============================] - 55s 8ms/step - loss: 0.1825 - accuracy: 0.9525 - val_loss: 0.5684 - val_accuracy: 0.8989
Epoch 47/50
7352/7352 [==============================] - 55s 8ms/step - loss: 0.1745 - accuracy: 0.9580 - val_loss: 0.5409 - val_accuracy: 0.9046
Epoch 48/50
7352/7352 [==============================] - 55s 8ms/step - loss: 0.2347 - accuracy: 0.9453 - val_loss: 1.7685 - val_accuracy: 0.6352
Epoch 49/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.3437 - accuracy: 0.9136 - val_loss: 0.4971 - val_accuracy: 0.8965
Epoch 50/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.2704 - accuracy: 0.9316 - val_loss: 0.5486 - val_accuracy: 0.8938
```

Out[14]:

```
<keras.callbacks.callbacks.History at 0x7f12ea9e8470>
```

In [15]:

```python
# Confusion Matrix
print(confusion_matrix(Y_test, model.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0         0        0                   0
SITTING                  0      349       119        0                   0
STANDING                 0       46       480        4                   0
WALKING                  0        0         0      460                  20
WALKING_DOWNSTAIRS       0        0         0        4                 404
WALKING_UPSTAIRS         0        2         0        5                  33

Pred                WALKING_UPSTAIRS
True
LAYING                            27
SITTING                           23
STANDING                           2
WALKING                           16
WALKING_DOWNSTAIRS                12
WALKING_UPSTAIRS                 431
```

In [16]:

```python
score = model.evaluate(X_test, Y_test)
```

```
2947/2947 [==============================] - 5s 2ms/step
```

In [17]:

```python
score
```

Out[17]:

```
[0.5485815121676908, 0.8937903046607971]
```

FOUR

In [11]:

```python
# Initializing parameters
epochs = 50
batch_size = 70
n_hidden1 = 32
n_hidden2 = 64
```

In [12]:

```python
# https://machinelearningmastery.com/stacked-long-short-term-memory-networks/
# Initiliazing the sequential model
model = Sequential()

# Configuring the parameters
model.add(LSTM(n_hidden1, kernel_regularizer= regularizers.l2(0.01), return_sequences=True,
          input_shape=(timesteps, input_dim)))
model.add(LSTM(n_hidden2, kernel_regularizer= regularizers.l2(0.01), input_shape=(timesteps, input_dim)))

# Adding a dropout layer
model.add(Dropout(0.5))

# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))

model.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling BaseResource
Variable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 128, 32)           5376
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                24832
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 390
=================================================================
Total params: 30,598
Trainable params: 30,598
Non-trainable params: 0
_________________________________________________________________
```

In [13]:

```python
# Compiling the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [14]:

```python
# Training the model
model.fit(X_train, Y_train, batch_size=batch_size, validation_data=(X_test, Y_test), epochs=epochs)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow_core/python/ops/math_grad.py:1424: where (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 7352 samples, validate on 2947 samples
Epoch 1/50
7352/7352 [==============================] - 54s 7ms/step - loss: 1.8588 - accuracy: 0.3343 - val_loss: 1.5992 - val_accuracy: 0.3271
Epoch 2/50
7352/7352 [==============================] - 50s 7ms/step - loss: 1.4525 - accuracy: 0.3625 - val_loss: 1.4467 - val_accuracy: 0.4075
Epoch 3/50
7352/7352 [==============================] - 50s 7ms/step - loss: 1.2425 - accuracy: 0.4854 - val_loss: 1.2867 - val_accuracy: 0.4388
Epoch 4/50
7352/7352 [==============================] - 51s 7ms/step - loss: 1.2264 - accuracy: 0.4887 - val_loss: 1.3395 - val_accuracy: 0.5215
Epoch 5/50
7352/7352 [==============================] - 51s 7ms/step - loss: 1.2145 - accuracy: 0.5014 - val_loss: 1.2318 - val_accuracy: 0.5531
Epoch 6/50
7352/7352 [==============================] - 51s 7ms/step - loss: 1.2475 - accuracy: 0.4803 - val_loss: 1.2367 - val_accuracy: 0.5066
Epoch 7/50
7352/7352 [==============================] - 51s 7ms/step - loss: 1.3889 - accuracy: 0.4229 - val_loss: 1.4138 - val_accuracy: 0.3529
Epoch 8/50
7352/7352 [==============================] - 51s 7ms/step - loss: 1.2888 - accuracy: 0.4340 - val_loss: 1.2542 - val_accuracy: 0.3811
Epoch 9/50
7352/7352 [==============================] - 51s 7ms/step - loss: 1.3222 - accuracy: 0.4162 - val_loss: 1.3625 - val_accuracy: 0.3722
Epoch 10/50
7352/7352 [==============================] - 51s 7ms/step - loss: 1.4575 - accuracy: 0.4638 - val_loss: 1.7101 - val_accuracy: 0.3268
Epoch 11/50
7352/7352 [==============================] - 51s 7ms/step - loss: 1.4903 - accuracy: 0.3388 - val_loss: 1.4940 - val_accuracy: 0.3241
Epoch 12/50
7352/7352 [==============================] - 52s 7ms/step - loss: 1.4166 - accuracy: 0.3566 - val_loss: 1.4560 - val_accuracy: 0.3213
Epoch 13/50
7352/7352 [==============================] - 53s 7ms/step - loss: 1.3790 - accuracy: 0.4068 - val_loss: 1.4648 - val_accuracy: 0.3393
Epoch 14/50
7352/7352 [==============================] - 52s 7ms/step - loss: 1.4227 - accuracy: 0.3698 - val_loss: 1.4850 - val_accuracy: 0.3397
Epoch 15/50
7352/7352 [==============================] - 52s 7ms/step - loss: 1.4087 - accuracy: 0.3603 - val_loss: 1.4672 - val_accuracy: 0.3393
Epoch 16/50
7352/7352 [==============================] - 51s 7ms/step - loss: 1.3886 - accuracy: 0.3674 - val_loss: 1.4361 - val_accuracy: 0.3434
Epoch 17/50
7352/7352 [==============================] - 52s 7ms/step - loss: 1.3878 - accuracy: 0.3911 - val_loss: 1.4566 - val_accuracy: 0.3512
Epoch 18/50
7352/7352 [==============================] - 52s 7ms/step - loss: 1.3872 - accuracy: 0.3617 - val_loss: 1.4407 - val_accuracy: 0.3397
Epoch 19/50
7352/7352 [==============================] - 52s 7ms/step - loss: 1.3586 - accuracy: 0.4038 - val_loss: 1.3980 - val_accuracy: 0.4255
Epoch 20/50
7352/7352 [==============================] - 53s 7ms/step - loss: 1.3112 - accuracy: 0.4374 - val_loss: 1.2918 - val_accuracy: 0.4795
Epoch 21/50
7352/7352 [==============================] - 54s 7ms/step - loss: 1.2098 - accuracy: 0.4850 - val_loss: 1.5007 - val_accuracy: 0.3828
Epoch 22/50
7352/7352 [==============================] - 53s 7ms/step - loss: 1.2107 - accuracy: 0.4893 - val_loss: 1.1678 - val_accuracy: 0.5103
Epoch 23/50
7352/7352 [==============================] - 53s 7ms/step - loss: 0.9867 - accuracy: 0.5890 - val_loss: 1.0594 - val_accuracy: 0.5677
Epoch 24/50
```

```
7352/7352 [==============================] - 53s 7ms/step - loss: 1.0233 - accuracy: 0.5918 - val_loss: 1.0620 - val_accuracy: 0.5541
Epoch 25/50
7352/7352 [==============================] - 53s 7ms/step - loss: 0.9400 - accuracy: 0.5676 - val_loss: 1.0485 - val_accuracy: 0.5477
Epoch 26/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.9180 - accuracy: 0.6054 - val_loss: 1.0236 - val_accuracy: 0.5843
Epoch 27/50
7352/7352 [==============================] - 53s 7ms/step - loss: 1.1483 - accuracy: 0.5166 - val_loss: 1.2465 - val_accuracy: 0.4520
Epoch 28/50
7352/7352 [==============================] - 52s 7ms/step - loss: 1.0679 - accuracy: 0.5462 - val_loss: 1.1707 - val_accuracy: 0.4846
Epoch 29/50
7352/7352 [==============================] - 52s 7ms/step - loss: 0.8892 - accuracy: 0.6313 - val_loss: 0.9455 - val_accuracy: 0.5965
Epoch 30/50
7352/7352 [==============================] - 52s 7ms/step - loss: 0.7927 - accuracy: 0.6400 - val_loss: 0.9363 - val_accuracy: 0.5948
Epoch 31/50
7352/7352 [==============================] - 53s 7ms/step - loss: 0.7813 - accuracy: 0.6483 - val_loss: 0.9761 - val_accuracy: 0.5887
Epoch 32/50
7352/7352 [==============================] - 53s 7ms/step - loss: 0.7843 - accuracy: 0.6424 - val_loss: 0.9392 - val_accuracy: 0.5748
Epoch 33/50
7352/7352 [==============================] - 53s 7ms/step - loss: 0.7660 - accuracy: 0.6439 - val_loss: 0.9583 - val_accuracy: 0.5877
Epoch 34/50
7352/7352 [==============================] - 52s 7ms/step - loss: 0.7673 - accuracy: 0.6477 - val_loss: 2.2394 - val_accuracy: 0.3750
Epoch 35/50
7352/7352 [==============================] - 53s 7ms/step - loss: 1.3035 - accuracy: 0.4856 - val_loss: 1.2020 - val_accuracy: 0.5175
Epoch 36/50
7352/7352 [==============================] - 53s 7ms/step - loss: 0.8683 - accuracy: 0.6294 - val_loss: 0.8537 - val_accuracy: 0.5965
Epoch 37/50
7352/7352 [==============================] - 54s 7ms/step - loss: 0.9738 - accuracy: 0.6017 - val_loss: 1.1880 - val_accuracy: 0.4839
Epoch 38/50
7352/7352 [==============================] - 54s 7ms/step - loss: 1.0554 - accuracy: 0.5290 - val_loss: 1.0148 - val_accuracy: 0.5266
Epoch 39/50
7352/7352 [==============================] - 55s 7ms/step - loss: 1.1817 - accuracy: 0.5322 - val_loss: 1.3822 - val_accuracy: 0.4486
Epoch 40/50
7352/7352 [==============================] - 54s 7ms/step - loss: 1.1810 - accuracy: 0.5095 - val_loss: 1.1416 - val_accuracy: 0.5209
Epoch 41/50
7352/7352 [==============================] - 55s 8ms/step - loss: 1.0262 - accuracy: 0.5775 - val_loss: 1.1379 - val_accuracy: 0.5151
Epoch 42/50
7352/7352 [==============================] - 55s 8ms/step - loss: 0.9318 - accuracy: 0.6028 - val_loss: 1.2614 - val_accuracy: 0.4469
Epoch 43/50
7352/7352 [==============================] - 54s 7ms/step - loss: 1.1441 - accuracy: 0.4942 - val_loss: 1.0252 - val_accuracy: 0.4730
Epoch 44/50
7352/7352 [==============================] - 53s 7ms/step - loss: 0.8884 - accuracy: 0.5850 - val_loss: 0.8912 - val_accuracy: 0.5952
Epoch 45/50
7352/7352 [==============================] - 54s 7ms/step - loss: 0.8063 - accuracy: 0.6364 - val_loss: 0.9072 - val_accuracy: 0.5840
Epoch 46/50
7352/7352 [==============================] - 55s 7ms/step - loss: 0.8026 - accuracy: 0.6287 - val_loss: 0.8434 - val_accuracy: 0.6071
Epoch 47/50
7352/7352 [==============================] - 54s 7ms/step - loss: 0.7563 - accuracy: 0.6498 - val_loss: 0.9151 - val_accuracy: 0.5976
Epoch 48/50
7352/7352 [==============================] - 54s 7ms/step - loss: 0.7479 - accuracy: 0.6459 - val_loss: 0.8487 - val_accuracy: 0.6071
Epoch 49/50
7352/7352 [==============================] - 52s 7ms/step - loss: 0.7967 - accuracy: 0.6333 - val_loss: 0.8786 - val_accuracy: 0.5718
Epoch 50/50
7352/7352 [==============================] - 51s 7ms/step - loss: 0.7630 - accuracy: 0.6435 - val_loss: 0.8606 - val_accuracy: 0.5847
```

Out[14]:

```
<keras.callbacks.callbacks.History at 0x7f49d61ea7f0>
```

In [15]:

```python
# Confusion Matrix
print(confusion_matrix(Y_test, model.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING
True
LAYING                 534        2         0        1
SITTING                  0      209       255       27
STANDING                 0       17       488       27
WALKING                  0        0         4      492
WALKING_DOWNSTAIRS       0        0         0      420
WALKING_UPSTAIRS         0        0         2      469
```

In [16]:

```python
score = model.evaluate(X_test, Y_test)
```

```
2947/2947 [==============================] - 5s 2ms/step
```

In [17]:

```
score
```

Out[17]:

[0.8606069580271642, 0.5846623778343201]

Five

In [11]:

```
# Initializing parameters
epochs = 50
batch_size = 70
n_hidden1 = 32
n_hidden2 = 64
```

In [12]:

```
# https://machinelearningmastery.com/stacked-long-short-term-memory-networks/
# Initiliazing the sequential model
model = Sequential()

# Configuring the parameters
model.add(LSTM(n_hidden1, kernel_initializer= 'glorot_normal', kernel_regularizer= regularizers.l2(0.001), return_sequences=True,
          input_shape=(timesteps, input_dim)))
model.add(LSTM(n_hidden2, kernel_initializer= 'glorot_normal', kernel_regularizer= regularizers.l2(0.001), input_shape=(timesteps, input_dim)))

# Adding a dropout layer
model.add(Dropout(0.5))

# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))

model.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling BaseResource
Variable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 128, 32)           5376
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                24832
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 390
=================================================================
Total params: 30,598
Trainable params: 30,598
Non-trainable params: 0
_________________________________________________________________
```

In [13]:

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

In [14]:

```
# Training the model
model.fit(X_train, Y_train, batch_size=batch_size, validation_data=(X_test, Y_test), epochs=epochs)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow_core/python/ops/math_grad.py:1424: where (from tensorflow.python.op
s.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables is deprec
ated. Please use tf.compat.v1.global_variables instead.

Train on 7352 samples, validate on 2947 samples
```

```
Epoch 1/50
7352/7352 [==============================] - 52s 7ms/step - loss: 1.3953 - accuracy: 0.4257 - val_loss: 1.1349 - val_accuracy: 0.5497
Epoch 2/50
7352/7352 [==============================] - 48s 7ms/step - loss: 0.9486 - accuracy: 0.5951 - val_loss: 0.9072 - val_accuracy: 0.5952
Epoch 3/50
7352/7352 [==============================] - 48s 7ms/step - loss: 0.8026 - accuracy: 0.6359 - val_loss: 0.8386 - val_accuracy: 0.6359
Epoch 4/50
7352/7352 [==============================] - 48s 7ms/step - loss: 0.7337 - accuracy: 0.6488 - val_loss: 0.7468 - val_accuracy: 0.5908
Epoch 5/50
7352/7352 [==============================] - 48s 7ms/step - loss: 0.7845 - accuracy: 0.6292 - val_loss: 0.7871 - val_accuracy: 0.6383
Epoch 6/50
7352/7352 [==============================] - 48s 7ms/step - loss: 0.7039 - accuracy: 0.6635 - val_loss: 0.8010 - val_accuracy: 0.6189
Epoch 7/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.7142 - accuracy: 0.6581 - val_loss: 0.7637 - val_accuracy: 0.6257
Epoch 8/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.6716 - accuracy: 0.6778 - val_loss: 0.7288 - val_accuracy: 0.6502
Epoch 9/50
7352/7352 [==============================] - 48s 7ms/step - loss: 0.7734 - accuracy: 0.6649 - val_loss: 1.0778 - val_accuracy: 0.5599
Epoch 10/50
7352/7352 [==============================] - 48s 7ms/step - loss: 0.7873 - accuracy: 0.6439 - val_loss: 0.7871 - val_accuracy: 0.6637
Epoch 11/50
7352/7352 [==============================] - 48s 7ms/step - loss: 0.6874 - accuracy: 0.6851 - val_loss: 0.7605 - val_accuracy: 0.7106
Epoch 12/50
7352/7352 [==============================] - 48s 7ms/step - loss: 0.7393 - accuracy: 0.6800 - val_loss: 0.7302 - val_accuracy: 0.7194
Epoch 13/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.6638 - accuracy: 0.7297 - val_loss: 1.1069 - val_accuracy: 0.5674
Epoch 14/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.6627 - accuracy: 0.7142 - val_loss: 0.6769 - val_accuracy: 0.7241
Epoch 15/50
7352/7352 [==============================] - 48s 7ms/step - loss: 0.6563 - accuracy: 0.7443 - val_loss: 0.6882 - val_accuracy: 0.7231
Epoch 16/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.5484 - accuracy: 0.7714 - val_loss: 0.6459 - val_accuracy: 0.7469
Epoch 17/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.5123 - accuracy: 0.7839 - val_loss: 0.6197 - val_accuracy: 0.7340
Epoch 18/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.5474 - accuracy: 0.7499 - val_loss: 0.5692 - val_accuracy: 0.7462
Epoch 19/50
7352/7352 [==============================] - 48s 7ms/step - loss: 0.4462 - accuracy: 0.7965 - val_loss: 0.5722 - val_accuracy: 0.7540
Epoch 20/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.4076 - accuracy: 0.8075 - val_loss: 0.5824 - val_accuracy: 0.7608
Epoch 21/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.4161 - accuracy: 0.8021 - val_loss: 0.5877 - val_accuracy: 0.7662
Epoch 22/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.3862 - accuracy: 0.8278 - val_loss: 0.5155 - val_accuracy: 0.8252
Epoch 23/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.3609 - accuracy: 0.8700 - val_loss: 0.5932 - val_accuracy: 0.7896
Epoch 24/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.3025 - accuracy: 0.9110 - val_loss: 0.5983 - val_accuracy: 0.8161
Epoch 25/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.2850 - accuracy: 0.9159 - val_loss: 0.5426 - val_accuracy: 0.8751
Epoch 26/50
7352/7352 [==============================] - 48s 7ms/step - loss: 0.3649 - accuracy: 0.8788 - val_loss: 0.5147 - val_accuracy: 0.8588
Epoch 27/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.2225 - accuracy: 0.9399 - val_loss: 0.4701 - val_accuracy: 0.8887
Epoch 28/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.2010 - accuracy: 0.9427 - val_loss: 0.5142 - val_accuracy: 0.8690
Epoch 29/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.1920 - accuracy: 0.9429 - val_loss: 0.5133 - val_accuracy: 0.8707
Epoch 30/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.2169 - accuracy: 0.9377 - val_loss: 0.6523 - val_accuracy: 0.8470
Epoch 31/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.2435 - accuracy: 0.9324 - val_loss: 0.4705 - val_accuracy: 0.8792
Epoch 32/50
7352/7352 [==============================] - 50s 7ms/step - loss: 0.2236 - accuracy: 0.9369 - val_loss: 0.5066 - val_accuracy: 0.8707
Epoch 33/50
7352/7352 [==============================] - 50s 7ms/step - loss: 0.2049 - accuracy: 0.9381 - val_loss: 0.5001 - val_accuracy: 0.8856
Epoch 34/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.1861 - accuracy: 0.9471 - val_loss: 0.4935 - val_accuracy: 0.8894
Epoch 35/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.1848 - accuracy: 0.9441 - val_loss: 0.4807 - val_accuracy: 0.8938
Epoch 36/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.1653 - accuracy: 0.9513 - val_loss: 0.5227 - val_accuracy: 0.8911
Epoch 37/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.1736 - accuracy: 0.9491 - val_loss: 0.4926 - val_accuracy: 0.8938
Epoch 38/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.1664 - accuracy: 0.9510 - val_loss: 0.5152 - val_accuracy: 0.8945
Epoch 39/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.1796 - accuracy: 0.9429 - val_loss: 0.4962 - val_accuracy: 0.9009
Epoch 40/50
7352/7352 [==============================] - 50s 7ms/step - loss: 0.1963 - accuracy: 0.9441 - val_loss: 0.4551 - val_accuracy: 0.9019
Epoch 41/50
7352/7352 [==============================] - 50s 7ms/step - loss: 0.1619 - accuracy: 0.9533 - val_loss: 0.5558 - val_accuracy: 0.8809
```

```
Epoch 42/50
7352/7352 [==============================] - 50s 7ms/step - loss: 0.1655 - accuracy: 0.9482 - val_loss: 0.4693 - val_accuracy: 0.8992
Epoch 43/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.1849 - accuracy: 0.9406 - val_loss: 0.4572 - val_accuracy: 0.8924
Epoch 44/50
7352/7352 [==============================] - 50s 7ms/step - loss: 0.1555 - accuracy: 0.9512 - val_loss: 0.5316 - val_accuracy: 0.8863
Epoch 45/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.1474 - accuracy: 0.9555 - val_loss: 0.4766 - val_accuracy: 0.8999
Epoch 46/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.1418 - accuracy: 0.9563 - val_loss: 0.5834 - val_accuracy: 0.8802
Epoch 47/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.1997 - accuracy: 0.9407 - val_loss: 0.6672 - val_accuracy: 0.8592
Epoch 48/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.4829 - accuracy: 0.8659 - val_loss: 0.3533 - val_accuracy: 0.9019
Epoch 49/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.1978 - accuracy: 0.9441 - val_loss: 0.4056 - val_accuracy: 0.8958
Epoch 50/50
7352/7352 [==============================] - 49s 7ms/step - loss: 0.1755 - accuracy: 0.9489 - val_loss: 0.4329 - val_accuracy: 0.8860
```

Out[14]:

```
<keras.callbacks.callbacks.History at 0x7fd85b15c8d0>
```

In [15]:

```python
# Confusion Matrix
print(confusion_matrix(Y_test, model.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0         0        0                   0
SITTING                  5      383        97        6                   0
STANDING                 0      101       430        1                   0
WALKING                  0        0         0      470                   2
WALKING_DOWNSTAIRS       0        0         0        0                 407
WALKING_UPSTAIRS         0        0         0       49                  11

Pred                WALKING_UPSTAIRS
True
LAYING                            27
SITTING                            0
STANDING                           0
WALKING                           24
WALKING_DOWNSTAIRS                13
WALKING_UPSTAIRS                 411
```

In [16]:

```python
score = model.evaluate(X_test, Y_test)
```

```
2947/2947 [==============================] - 4s 1ms/step
```

In [17]:

```python
score
```

Out[17]:

```
[0.4328820135433026, 0.8859857320785522]
```

CNN

Divide and Conquer-Based 1D CNN

In [5]:

```python
from keras import regularizers
```

In [23]:

```python
# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
```

```
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
# Triaxial acceleration from the accelerometer is total acceleration

SIGNALS = ["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"]

# Labelling the classes in y.
label = {0:'WALKING', 1:'WALKING_UPSTAIRS', 2:'WALKING_DOWNSTAIRS', 3:'SITTING', 4:'STANDING', 5:'LAYING'}
```

In [4]:

```
def confusion_matrix(Y_true, Y_pred):
    Y_true = pd.Series([label[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([label[y] for y in np.argmax(Y_pred, axis=1)])

    return pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])
```

In [6]:

```
# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the load
def load_signals(subset):
    signals_data = []
    filename = 'body_acc_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_acc_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_acc_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())


    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))


def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = 'y_{}.txt'.format(subset)
    y = _read_csv(filename)[0]

    # Dynamic activities
    y[y<=3] = 0

    # Static activities
    y[y>3] = 1

    return pd.get_dummies(y).as_matrix()

def load_data():
    """
    Obtain the dataset from multiple files.
```

```
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_test = load_y('train'), load_y('test')

    return X_train, X_test, y_train, y_test

# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error

def plt_dynamic(x, vy, ty, ax, color = 'b'):
    ax.plot(x, vy, 'b', label = 'Validation Loss')
    ax.plot(x, vy, 'b', label = 'Validation Loss')
    ax.plot(x, ty, 'r', label = 'Train Loss')
    plt.grid()
    plt.legend()
    fig.canvas.draw()
```

In [7]:

```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()

print('X_train shape is: ',X_train.shape)
print('Y_train shape is: ',Y_train.shape)
print('X_test shape is: ',X_test.shape)
print('Y_test shape is: ',Y_test.shape)
```

```
X_train shape is:  (7352, 128, 9)
Y_train shape is:  (7352, 2)
X_test shape is:  (2947, 128, 9)
Y_test shape is:  (2947, 2)
```

In [8]:

```
# Importing tensorflow
np.random.seed(42)
import tensorflow as tf
tf.set_random_seed(42)
```

In [9]:

```
# Configuring a session
session_conf = tf.ConfigProto(intra_op_parallelism_threads= 1,
                              inter_op_parallelism_threads= 1)
```

In [10]:

```
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

In [11]:

```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [12]:

```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)

print(timesteps)
print(input_dim)
print(len(X_train))
```

```
128
9
7352
```

In [13]:

```
model1= Sequential()
model1.add(Conv1D(filters= 32, kernel_size= 3, activation= 'relu', kernel_initializer= 'he_normal',
            kernel_regularizer = regularizers.l2(0.1), input_shape=(timesteps, input_dim)))
model1.add(Dropout(0.5))
model1.add(MaxPooling1D(pool_size= 2))
model1.add(Flatten())
model1.add(Dense(units= 2, activation= 'softmax'))
model1.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling BaseResource
Variable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:4070: The name tf.nn.max_pool is deprecat
ed. Please use tf.nn.max_pool2d instead.

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 126, 32)           896
_________________________________________________________________
dropout_1 (Dropout)          (None, 126, 32)           0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 63, 32)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2016)              0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 4034
=================================================================
Total params: 4,930
Trainable params: 4,930
Non-trainable params: 0
_________________________________________________________________
```

In [14]:

```
# Compiling the model
model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [15]:

```
# Initializing parameters
epochs = 10
batch_size = 70

# Training the model
history1= model1.fit(X_train, Y_train, batch_size=batch_size, validation_data=(X_test, Y_test), epochs=epochs)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables is deprec
ated. Please use tf.compat.v1.global_variables instead.

Train on 7352 samples, validate on 2947 samples
Epoch 1/10
7352/7352 [==============================] - 5s 647us/step - loss: 4.8197 - accuracy: 0.9234 - val_loss: 3.3904 - val_accuracy: 0.9817
Epoch 2/10
7352/7352 [==============================] - 2s 251us/step - loss: 2.4643 - accuracy: 0.9959 - val_loss: 1.7946 - val_accuracy: 0.9915
Epoch 3/10
7352/7352 [==============================] - 2s 250us/step - loss: 1.2884 - accuracy: 0.9978 - val_loss: 0.9574 - val_accuracy: 0.9939
Epoch 4/10
7352/7352 [==============================] - 2s 257us/step - loss: 0.6708 - accuracy: 0.9984 - val_loss: 0.5187 - val_accuracy: 0.9966
Epoch 5/10
7352/7352 [==============================] - 2s 258us/step - loss: 0.3508 - accuracy: 0.9986 - val_loss: 0.2916 - val_accuracy: 0.9949
Epoch 6/10
7352/7352 [==============================] - 2s 255us/step - loss: 0.1885 - accuracy: 0.9990 - val_loss: 0.1809 - val_accuracy: 0.9929
Epoch 7/10
7352/7352 [==============================] - 2s 250us/step - loss: 0.1093 - accuracy: 0.9980 - val_loss: 0.1173 - val_accuracy: 0.9976
Epoch 8/10
7352/7352 [==============================] - 2s 245us/step - loss: 0.0687 - accuracy: 0.9988 - val_loss: 0.0886 - val_accuracy: 0.9963
Epoch 9/10
7352/7352 [==============================] - 2s 252us/step - loss: 0.0511 - accuracy: 0.9981 - val_loss: 0.0737 - val_accuracy: 0.9966
Epoch 10/10
7352/7352 [==============================] - 2s 257us/step - loss: 0.0399 - accuracy: 0.9988 - val_loss: 0.0673 - val_accuracy: 0.9983
```

In [17]:

```
score1 = model1.evaluate(X_test, Y_test)
```

2947/2947 [==============================] - 0s 112us/step

In [18]:

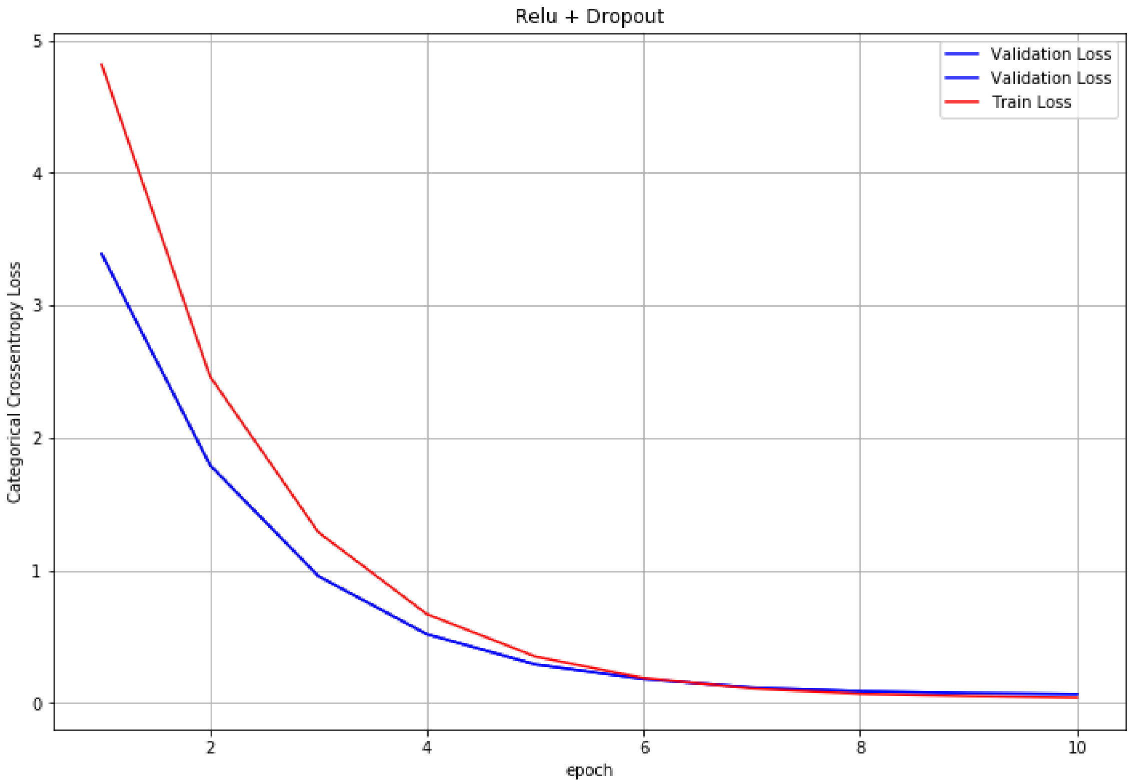
```
score1
```

Out[18]:

[0.06726545751762261, 0.9983033537864685]

In [19]:

```
fig, ax = plt.subplots(1,1, figsize = (12, 8))
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')
plt.title('Relu + Dropout')

# list of epoch numbers: epoch = 10
x = list(range(1,10+1))
vy = history1.history['val_loss']
ty = history1.history['loss']
plt_dynamic(x, vy, ty, ax)
```



In [20]:

```
import pickle
model1.save('model1')
```

Classification of Static Activities

In [62]:

```
SIGNALS = ["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"]

# Labelling the classes in 'y' before OHE {4:'SITTING', 5:'STANDING', 6:'LAYING'}
# Labelling the classes in 'y' after OHE
label = {0:'SITTING', 1:'STANDING', 2:'LAYING'}
```

```
# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the load
def load_signals(subset):
    signals_data = []
    filename = 'body_acc_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_acc_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_acc_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())


    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))

def confusion_matrix(Y_true, Y_pred):
    Y_true = pd.Series([label[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([label[y] for y in np.argmax(Y_pred, axis=1)])

    c_m = pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])

    plt.figure(figsize= (8, 6))
    c_m = sns.heatmap(c_m, annot=True, cmap= sns.light_palette("blue"), fmt=".3f", xticklabels=label.values(),
                      yticklabels= label.values())
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")


    return c_m

# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error

def plt_dynamic(x, vy, ty, ax, color = 'b'):
    ax.plot(x, vy, 'b', label = 'Validation Loss')
    ax.plot(x, vy, 'b', label = 'Validation Loss')
    ax.plot(x, ty, 'r', label = 'Train Loss')
    plt.grid()
    plt.legend()
    fig.canvas.draw()
```

In [15]:

```
def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = 'y_{}.txt'.format(subset)
    y = _read_csv(filename)[0]

    # Static activities
    y_subset = y>3
    y = y[y_subset]

    return pd.get_dummies(y).as_matrix(), y_subset

def load_data():
    """
    Obtain the dataset from multiple files.
```

```
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_train1 = load_y('train')
    y_test, y_test1 =  load_y('test')
    # collecting those datapoints where y > 3

    X_train = X_train[y_train1]
    X_test = X_test[y_test1]

    return X_train, X_test, y_train, y_test
```

In [16]:

```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()

print('X_train shape is: ',X_train.shape)
print('Y_train shape is: ',Y_train.shape)
print('X_test shape is: ',X_test.shape)
print('Y_test shape is: ',Y_test.shape)
```

```
X_train shape is:  (4067, 128, 9)
Y_train shape is:  (4067, 3)
X_test shape is:  (1560, 128, 9)
Y_test shape is:  (1560, 3)
```

In [24]:

```
# checking the y label and middle 1 is 'standing'
Y_train[:5]
```

Out[24]:

```
array([[0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0]], dtype=uint8)
```

In [25]:

```
# Importing tensorflow
np.random.seed(42)
import tensorflow as tf
tf.set_random_seed(42)
```

In [26]:

```
# Configuring a session
session_conf = tf.ConfigProto(intra_op_parallelism_threads= 1,
                              inter_op_parallelism_threads= 1)
```

In [27]:

```
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

In [28]:

```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [29]:

```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)

print('Timesteps:', timesteps)
print('Input Dim:', input_dim)
print('No. of Train datapoints:', len(X_train))
print('No of classes:',n_classes)
```

```
Timesteps: 128
Input Dim: 9
No. of Train datapoints: 4067
No of classes: 3
```

Arrived at this architecture after 50+ trails of architectures.

In [71]:

```
m= Sequential()
m.add(Conv1D(filters= 64, kernel_size= 5, activation= 'relu', kernel_initializer= 'he_uniform',
            input_shape=(timesteps, input_dim)))
m.add(Conv1D(filters= 64, kernel_size= 5, activation= 'relu', kernel_initializer= 'he_uniform'))
m.add(MaxPooling1D(pool_size= 2, padding= 'same'))
m.add(Dropout(0.40))
m.add(Conv1D(filters= 32, kernel_size= 5, activation= 'relu', kernel_initializer= 'he_uniform'))
m.add(Conv1D(filters= 32, kernel_size= 5, activation= 'relu', kernel_initializer= 'he_uniform',))

# https://stackoverflow.com/a/49089027/10219869
# https://stackoverflow.com/a/58498450/10219869
m.add(MaxPooling1D(pool_size= 2, padding= 'same'))
m.add(BatchNormalization())
m.add(Dropout(0.40))
m.add(Flatten())
m.add(Dense(units= 100, activation= 'relu'))
m.add(BatchNormalization())
m.add(Dropout(0.40))
m.add(Dense(units= 3, activation= 'softmax'))
m.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_25 (Conv1D)           (None, 124, 64)           2944      
_________________________________________________________________
conv1d_26 (Conv1D)           (None, 120, 64)           20544     
_________________________________________________________________
max_pooling1d_13 (MaxPooling (None, 60, 64)            0         
_________________________________________________________________
dropout_19 (Dropout)         (None, 60, 64)            0         
_________________________________________________________________
conv1d_27 (Conv1D)           (None, 56, 32)            10272     
_________________________________________________________________
conv1d_28 (Conv1D)           (None, 52, 32)            5152      
_________________________________________________________________
max_pooling1d_14 (MaxPooling (None, 26, 32)            0         
_________________________________________________________________
batch_normalization_14 (Batc (None, 26, 32)            128       
_________________________________________________________________
dropout_20 (Dropout)         (None, 26, 32)            0         
_________________________________________________________________
flatten_7 (Flatten)          (None, 832)               0         
_________________________________________________________________
dense_13 (Dense)             (None, 100)               83300     
_________________________________________________________________
batch_normalization_15 (Batc (None, 100)               400       
_________________________________________________________________
dropout_21 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_14 (Dense)             (None, 3)                 303       
=================================================================
Total params: 123,043
Trainable params: 122,779
Non-trainable params: 264
_________________________________________________________________
```

In [82]:

```
# https://keras.io/optimizers/
# Compiling the model
adam= keras.optimizers.Adam(learning_rate= 0.001)
rmsprop = optimizers.RMSprop(learning_rate= 0.001)
m.compile(loss='categorical_crossentropy', optimizer= 'adam', metrics=['accuracy'])
```

In [83]:

```
# Initializing parameters
epochs = 100
batch_size = 20

# Training the model
h= m.fit(X_train, Y_train, batch_size=batch_size, validation_data=(X_test, Y_test), epochs=epochs)
```

```
Train on 4067 samples, validate on 1560 samples
Epoch 1/100
4067/4067 [==============================] - 9s 2ms/step - loss: 0.0424 - accuracy: 0.9894 - val_loss: 0.5055 - val_accuracy: 0.9026
Epoch 2/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0421 - accuracy: 0.9899 - val_loss: 0.4361 - val_accuracy: 0.9218
Epoch 3/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0270 - accuracy: 0.9914 - val_loss: 0.4946 - val_accuracy: 0.9128
Epoch 4/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0668 - accuracy: 0.9865 - val_loss: 0.3612 - val_accuracy: 0.9224
Epoch 5/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0144 - accuracy: 0.9951 - val_loss: 0.3314 - val_accuracy: 0.9218
Epoch 6/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0186 - accuracy: 0.9948 - val_loss: 0.3938 - val_accuracy: 0.9205
Epoch 7/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0203 - accuracy: 0.9919 - val_loss: 0.3816 - val_accuracy: 0.9205
Epoch 8/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0297 - accuracy: 0.9914 - val_loss: 0.5033 - val_accuracy: 0.9000
Epoch 9/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0254 - accuracy: 0.9931 - val_loss: 0.4036 - val_accuracy: 0.9064
Epoch 10/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0099 - accuracy: 0.9963 - val_loss: 0.4205 - val_accuracy: 0.9122
Epoch 11/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0093 - accuracy: 0.9975 - val_loss: 0.4745 - val_accuracy: 0.9083
Epoch 12/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0250 - accuracy: 0.9911 - val_loss: 0.4930 - val_accuracy: 0.9122
Epoch 13/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0236 - accuracy: 0.9953 - val_loss: 0.4222 - val_accuracy: 0.9160
Epoch 14/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0336 - accuracy: 0.9929 - val_loss: 0.4701 - val_accuracy: 0.9096
Epoch 15/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0139 - accuracy: 0.9953 - val_loss: 0.4549 - val_accuracy: 0.9090
Epoch 16/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0302 - accuracy: 0.9926 - val_loss: 0.4305 - val_accuracy: 0.9160
Epoch 17/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0133 - accuracy: 0.9953 - val_loss: 0.3496 - val_accuracy: 0.9237
Epoch 18/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0323 - accuracy: 0.9892 - val_loss: 0.3102 - val_accuracy: 0.9308
Epoch 19/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0176 - accuracy: 0.9948 - val_loss: 0.4172 - val_accuracy: 0.9179
Epoch 20/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0203 - accuracy: 0.9939 - val_loss: 0.5099 - val_accuracy: 0.9058
Epoch 21/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0141 - accuracy: 0.9961 - val_loss: 0.4907 - val_accuracy: 0.9096
Epoch 22/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0155 - accuracy: 0.9951 - val_loss: 0.4426 - val_accuracy: 0.9135
Epoch 23/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0444 - accuracy: 0.9845 - val_loss: 0.3280 - val_accuracy: 0.9199
Epoch 24/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0257 - accuracy: 0.9934 - val_loss: 0.3385 - val_accuracy: 0.9218
Epoch 25/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0439 - accuracy: 0.9892 - val_loss: 0.4213 - val_accuracy: 0.9282
Epoch 26/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0244 - accuracy: 0.9956 - val_loss: 0.3549 - val_accuracy: 0.9224
Epoch 27/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0103 - accuracy: 0.9973 - val_loss: 0.4475 - val_accuracy: 0.9147
Epoch 28/100
4067/4067 [==============================] - 5s 1ms/step - loss: 0.0102 - accuracy: 0.9970 - val_loss: 0.4589 - val_accuracy: 0.9096
Epoch 29/100
4067/4067 [==============================] - 5s 1ms/step - loss: 0.0313 - accuracy: 0.9899 - val_loss: 0.4489 - val_accuracy: 0.9032
Epoch 30/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0347 - accuracy: 0.9889 - val_loss: 0.3273 - val_accuracy: 0.9224
Epoch 31/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0283 - accuracy: 0.9914 - val_loss: 0.3912 - val_accuracy: 0.9051
Epoch 32/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0088 - accuracy: 0.9970 - val_loss: 0.3481 - val_accuracy: 0.9231
Epoch 33/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0117 - accuracy: 0.9958 - val_loss: 0.4046 - val_accuracy: 0.9096
Epoch 34/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0171 - accuracy: 0.9951 - val_loss: 0.4028 - val_accuracy: 0.9167
Epoch 35/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0152 - accuracy: 0.9943 - val_loss: 0.5677 - val_accuracy: 0.8904
Epoch 36/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0211 - accuracy: 0.9941 - val_loss: 0.3806 - val_accuracy: 0.9173
Epoch 37/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0104 - accuracy: 0.9966 - val_loss: 0.5702 - val_accuracy: 0.9096
```

```
Epoch 38/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0224 - accuracy: 0.9919 - val_loss: 0.4706 - val_accuracy: 0.9160
Epoch 39/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0148 - accuracy: 0.9946 - val_loss: 0.3842 - val_accuracy: 0.9212
Epoch 40/100
4067/4067 [==============================] - 5s 1ms/step - loss: 0.0172 - accuracy: 0.9961 - val_loss: 0.4092 - val_accuracy: 0.9179
Epoch 41/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0218 - accuracy: 0.9946 - val_loss: 0.4374 - val_accuracy: 0.9096
Epoch 42/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0170 - accuracy: 0.9961 - val_loss: 0.3610 - val_accuracy: 0.9173
Epoch 43/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0147 - accuracy: 0.9951 - val_loss: 0.4006 - val_accuracy: 0.9186
Epoch 44/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0039 - accuracy: 0.9993 - val_loss: 0.3962 - val_accuracy: 0.9186
Epoch 45/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0062 - accuracy: 0.9983 - val_loss: 0.4297 - val_accuracy: 0.9141
Epoch 46/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0267 - accuracy: 0.9931 - val_loss: 0.4436 - val_accuracy: 0.9212
Epoch 47/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0163 - accuracy: 0.9946 - val_loss: 0.4261 - val_accuracy: 0.9231
Epoch 48/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0088 - accuracy: 0.9968 - val_loss: 0.6246 - val_accuracy: 0.9103
Epoch 49/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0110 - accuracy: 0.9961 - val_loss: 0.4827 - val_accuracy: 0.9128
Epoch 50/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0282 - accuracy: 0.9897 - val_loss: 0.4029 - val_accuracy: 0.9128
Epoch 51/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0274 - accuracy: 0.9924 - val_loss: 0.3961 - val_accuracy: 0.9019
Epoch 52/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0079 - accuracy: 0.9975 - val_loss: 0.4422 - val_accuracy: 0.9128
Epoch 53/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0159 - accuracy: 0.9966 - val_loss: 0.4013 - val_accuracy: 0.9173
Epoch 54/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0151 - accuracy: 0.9961 - val_loss: 0.3933 - val_accuracy: 0.9160
Epoch 55/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0268 - accuracy: 0.9929 - val_loss: 0.4657 - val_accuracy: 0.9173
Epoch 56/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0057 - accuracy: 0.9988 - val_loss: 0.4454 - val_accuracy: 0.9147
Epoch 57/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0025 - accuracy: 0.9993 - val_loss: 0.4845 - val_accuracy: 0.9128
Epoch 58/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0103 - accuracy: 0.9958 - val_loss: 0.4864 - val_accuracy: 0.9135
Epoch 59/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0228 - accuracy: 0.9936 - val_loss: 0.5151 - val_accuracy: 0.9147
Epoch 60/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0142 - accuracy: 0.9948 - val_loss: 0.4454 - val_accuracy: 0.9135
Epoch 61/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0161 - accuracy: 0.9956 - val_loss: 0.6594 - val_accuracy: 0.9122
Epoch 62/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0241 - accuracy: 0.9921 - val_loss: 0.4121 - val_accuracy: 0.9179
Epoch 63/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0132 - accuracy: 0.9961 - val_loss: 0.5043 - val_accuracy: 0.9122
Epoch 64/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0037 - accuracy: 0.9993 - val_loss: 0.4394 - val_accuracy: 0.9244
Epoch 65/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0227 - accuracy: 0.9948 - val_loss: 0.3911 - val_accuracy: 0.9269
Epoch 66/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0371 - accuracy: 0.9924 - val_loss: 0.4304 - val_accuracy: 0.9179
Epoch 67/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0091 - accuracy: 0.9970 - val_loss: 0.4226 - val_accuracy: 0.9135
Epoch 68/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0066 - accuracy: 0.9980 - val_loss: 0.4135 - val_accuracy: 0.9135
Epoch 69/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0354 - accuracy: 0.9877 - val_loss: 0.4512 - val_accuracy: 0.9154
Epoch 70/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0188 - accuracy: 0.9936 - val_loss: 0.5428 - val_accuracy: 0.9115
Epoch 71/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0203 - accuracy: 0.9936 - val_loss: 0.5003 - val_accuracy: 0.9154
Epoch 72/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0108 - accuracy: 0.9958 - val_loss: 0.5079 - val_accuracy: 0.9083
Epoch 73/100
4067/4067 [==============================] - 6s 2ms/step - loss: 0.0068 - accuracy: 0.9975 - val_loss: 0.5569 - val_accuracy: 0.9058
Epoch 74/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0195 - accuracy: 0.9939 - val_loss: 0.4941 - val_accuracy: 0.9231
Epoch 75/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0091 - accuracy: 0.9975 - val_loss: 0.4488 - val_accuracy: 0.9096
Epoch 76/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0092 - accuracy: 0.9975 - val_loss: 0.3869 - val_accuracy: 0.9186
Epoch 77/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0103 - accuracy: 0.9973 - val_loss: 0.3627 - val_accuracy: 0.9224
Epoch 78/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0065 - accuracy: 0.9980 - val_loss: 0.4152 - val_accuracy: 0.9244
Epoch 79/100
```

```
Epoch 79/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0187 - accuracy: 0.9958 - val_loss: 0.4311 - val_accuracy: 0.9160
Epoch 80/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0085 - accuracy: 0.9973 - val_loss: 0.4775 - val_accuracy: 0.9141
Epoch 81/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0031 - accuracy: 0.9983 - val_loss: 0.4466 - val_accuracy: 0.9263
Epoch 82/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0090 - accuracy: 0.9978 - val_loss: 0.4933 - val_accuracy: 0.9237
Epoch 83/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0107 - accuracy: 0.9966 - val_loss: 0.4586 - val_accuracy: 0.9192
Epoch 84/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0368 - accuracy: 0.9899 - val_loss: 0.3906 - val_accuracy: 0.9167
Epoch 85/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0115 - accuracy: 0.9961 - val_loss: 0.4547 - val_accuracy: 0.9103
Epoch 86/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0059 - accuracy: 0.9980 - val_loss: 0.5307 - val_accuracy: 0.9115
Epoch 87/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0179 - accuracy: 0.9948 - val_loss: 0.5100 - val_accuracy: 0.8981
Epoch 88/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0298 - accuracy: 0.9921 - val_loss: 0.4624 - val_accuracy: 0.9109
Epoch 89/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0081 - accuracy: 0.9988 - val_loss: 0.3869 - val_accuracy: 0.9186
Epoch 90/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0030 - accuracy: 0.9993 - val_loss: 0.4273 - val_accuracy: 0.9192
Epoch 91/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0195 - accuracy: 0.9946 - val_loss: 0.3703 - val_accuracy: 0.9147
Epoch 92/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0067 - accuracy: 0.9983 - val_loss: 0.4068 - val_accuracy: 0.9122
Epoch 93/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0042 - accuracy: 0.9983 - val_loss: 0.5512 - val_accuracy: 0.9064
Epoch 94/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0109 - accuracy: 0.9980 - val_loss: 0.4673 - val_accuracy: 0.9147
Epoch 95/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0211 - accuracy: 0.9934 - val_loss: 0.4547 - val_accuracy: 0.9218
Epoch 96/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0116 - accuracy: 0.9956 - val_loss: 0.4314 - val_accuracy: 0.9141
Epoch 97/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0034 - accuracy: 0.9993 - val_loss: 0.5583 - val_accuracy: 0.9077
Epoch 98/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0128 - accuracy: 0.9958 - val_loss: 0.4526 - val_accuracy: 0.9141
Epoch 99/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0104 - accuracy: 0.9973 - val_loss: 0.4714 - val_accuracy: 0.9154
Epoch 100/100
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0052 - accuracy: 0.9985 - val_loss: 0.4710 - val_accuracy: 0.9218
```

In [84]:

```
s= m.evaluate(X_test, Y_test)
```

```
1560/1560 [==============================] - 0s 267us/step
```

In [85]:

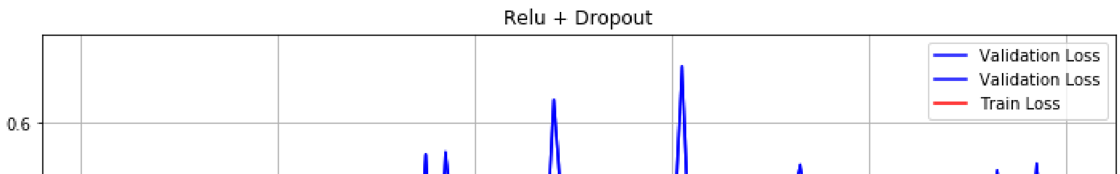
```
s
```

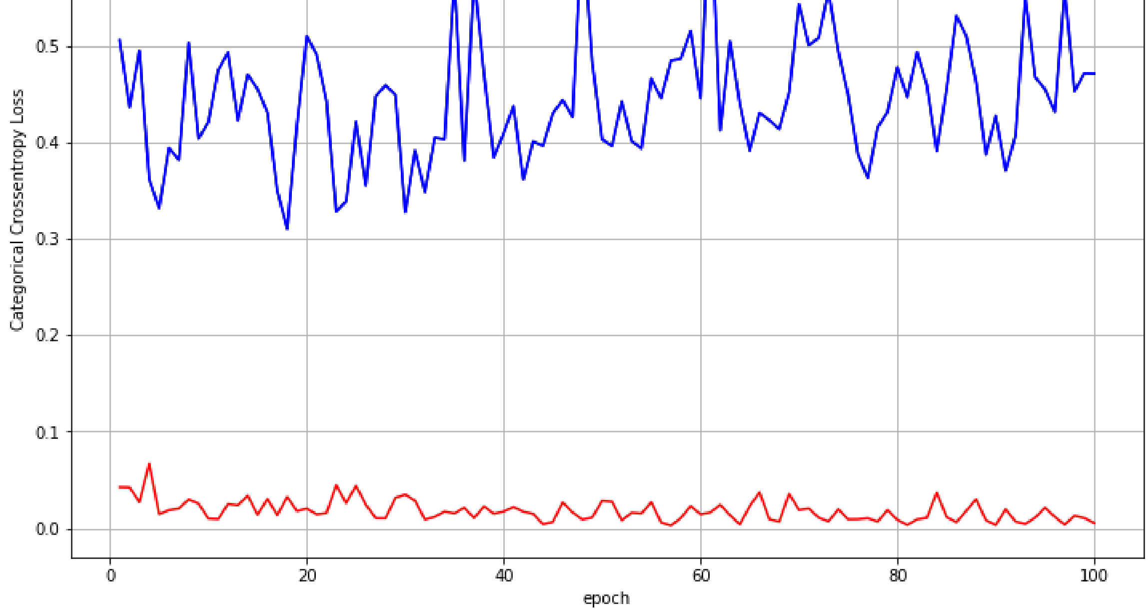
Out[85]:

```
[0.471027467745917, 0.9217948913574219]
```

In [86]:

```
fig, ax = plt.subplots(1,1, figsize = (12, 8))
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')
plt.title('Relu + Dropout')

# list of epoch numbers: epoch = 100
x = list(range(1,100+1))
vy = h.history['val_loss']
ty = h.history['loss']
plt_dynamic(x, vy, ty, ax)
```



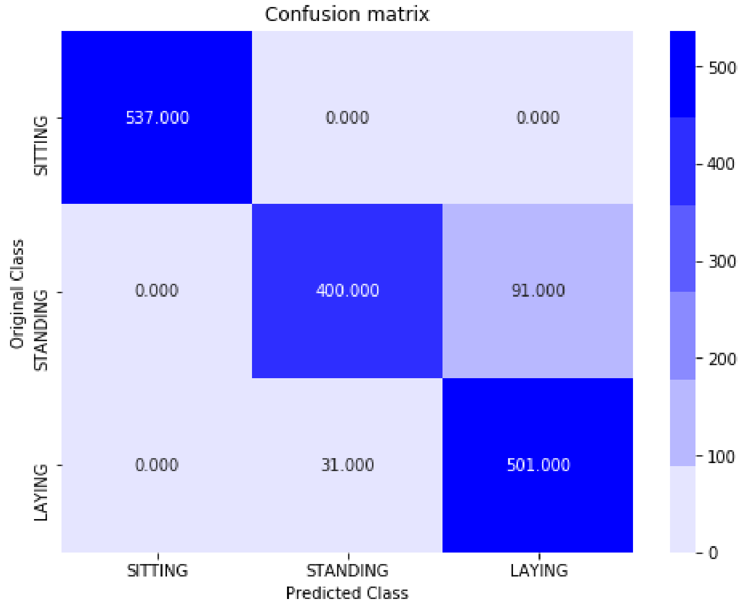


In [111]:

```
confusion_matrix(Y_test, m.predict(X_test))
```

Out[111]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f8df1b0c3c8>



In [79]:

```
model2.save('model2')
```

Classification of Dynamic Activities

In [112]:

```
SIGNALS = ["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"]

# Labelling the classes in 'y' before OHE {1:'WALKING', 2:'WALKING_UPSTAIRS', 3:'WALKING_DOWNSTAIRS'}
# Labelling the classes in 'y' after OHE
label = {0:'WALKING', 1:'WALKING_UPSTAIRS', 2:'WALKING_DOWNSTAIRS'}
```

```python
# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the load
def load_signals(subset):
    signals_data = []
    filename = 'body_acc_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_acc_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_acc_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())


    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))

def confusion_matrix(Y_true, Y_pred):

    Y_true = pd.Series([label[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([label[y] for y in np.argmax(Y_pred, axis=1)])

    c_m = pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])

    plt.figure(figsize= (8, 6))
    c_m = sns.heatmap(c_m, annot=True, cmap= sns.light_palette("blue"), fmt=".3f", xticklabels=label.values(),
                 yticklabels= label.values())
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    return c_m

# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error

def plt_dynamic(x, vy, ty, ax, color = 'b'):
    ax.plot(x, vy, 'b', label = 'Validation Loss')
    ax.plot(x, vy, 'b', label = 'Validation Loss')
    ax.plot(x, ty, 'r', label = 'Train Loss')
    plt.grid()
    plt.legend()
    fig.canvas.draw()
```

In [113]:

```python
def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = 'y_{}.txt'.format(subset)
    y = _read_csv(filename)[0]

    # Static activities
    y_subset = y<=3
    y = y[y_subset]

    return pd.get_dummies(y).as_matrix(), y_subset

def load_data():
    """
    Obtain the dataset from multiple files.
```

```
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_train1 = load_y('train')
    y_test, y_test1 =  load_y('test')
    # collecting those datapoints where y > 3

    X_train = X_train[y_train1]
    X_test = X_test[y_test1]

    return X_train, X_test, y_train, y_test
```

In [114]:

```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()

print('X_train shape is: ',X_train.shape)
print('Y_train shape is: ',Y_train.shape)
print('X_test shape is: ',X_test.shape)
print('Y_test shape is: ',Y_test.shape)
```

```
X_train shape is:  (3285, 128, 9)
Y_train shape is:  (3285, 3)
X_test shape is:  (1387, 128, 9)
Y_test shape is:  (1387, 3)
```

In [115]:

```
# checking the y label and middle 1 is 'standing'
Y_train[:5]
```

Out[115]:

```
array([[1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0]], dtype=uint8)
```

In [116]:

```
# Importing tensorflow
np.random.seed(42)
import tensorflow as tf
tf.set_random_seed(42)
```

In [117]:

```
# Configuring a session
session_conf = tf.ConfigProto(intra_op_parallelism_threads= 1,
                              inter_op_parallelism_threads= 1)
```

In [118]:

```
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

In [119]:

```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [120]:

```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)

print('Timesteps:', timesteps)
print('Input Dim:', input_dim)
print('No. of Train datapoints:', len(X_train))
print('No of classes:',n_classes)
```

Timesteps: 128
Input Dim: 9
No. of Train datapoints: 3285
No of classes: 3

In [121]:

```
m= Sequential()
m.add(Conv1D(filters= 64, kernel_size= 5, activation= 'relu', kernel_initializer= 'he_uniform',
                input_shape=(timesteps, input_dim)))
m.add(Conv1D(filters= 64, kernel_size= 5, activation= 'relu', kernel_initializer= 'he_uniform'))
m.add(MaxPooling1D(pool_size= 2, padding= 'same'))
m.add(Dropout(0.40))
m.add(Conv1D(filters= 32, kernel_size= 5, activation= 'relu', kernel_initializer= 'he_uniform'))
m.add(Conv1D(filters= 32, kernel_size= 5, activation= 'relu', kernel_initializer= 'he_uniform',))

# https://stackoverflow.com/a/49089027/10219869
# https://stackoverflow.com/a/58498450/10219869
m.add(MaxPooling1D(pool_size= 2, padding= 'same'))
m.add(BatchNormalization())
m.add(Dropout(0.40))
m.add(Flatten())
m.add(Dense(units= 100, activation= 'relu'))
m.add(BatchNormalization())
m.add(Dropout(0.40))
m.add(Dense(units= 3, activation= 'softmax'))
m.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_41 (Conv1D)           (None, 124, 64)           2944
_________________________________________________________________
conv1d_42 (Conv1D)           (None, 120, 64)           20544
_________________________________________________________________
max_pooling1d_21 (MaxPooling (None, 60, 64)            0
_________________________________________________________________
dropout_31 (Dropout)         (None, 60, 64)            0
_________________________________________________________________
conv1d_43 (Conv1D)           (None, 56, 32)            10272
_________________________________________________________________
conv1d_44 (Conv1D)           (None, 52, 32)            5152
_________________________________________________________________
max_pooling1d_22 (MaxPooling (None, 26, 32)            0
_________________________________________________________________
batch_normalization_22 (Batc (None, 26, 32)            128
_________________________________________________________________
dropout_32 (Dropout)         (None, 26, 32)            0
_________________________________________________________________
flatten_11 (Flatten)         (None, 832)               0
_________________________________________________________________
dense_21 (Dense)             (None, 100)               83300
_________________________________________________________________
batch_normalization_23 (Batc (None, 100)               400
_________________________________________________________________
dropout_33 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_22 (Dense)             (None, 3)                 303
=================================================================
Total params: 123,043
Trainable params: 122,779
Non-trainable params: 264
_________________________________________________________________
```

In [122]:

```
# https://keras.io/optimizers/
# Compiling the model
adam= keras.optimizers.Adam(learning_rate= 0.001)
rmsprop = optimizers.RMSprop(learning_rate= 0.001)
m.compile(loss='categorical_crossentropy', optimizer= 'adam', metrics=['accuracy'])
```

In [123]:

```
# Initializing parameters
epochs = 100
batch_size = 20
```

```python
# Training the model
h= m.fit(X_train, Y_train, batch_size=batch_size, validation_data=(X_test, Y_test), epochs=epochs)
```

```
Train on 3285 samples, validate on 1387 samples
Epoch 1/100
3285/3285 [==============================] - 8s 3ms/step - loss: 1.0449 - accuracy: 0.5945 - val_loss: 0.7546 - val_accuracy: 0.7195
Epoch 2/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.2208 - accuracy: 0.9184 - val_loss: 1.3485 - val_accuracy: 0.6936
Epoch 3/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0742 - accuracy: 0.9735 - val_loss: 0.2296 - val_accuracy: 0.9120
Epoch 4/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0553 - accuracy: 0.9820 - val_loss: 0.2610 - val_accuracy: 0.8940
Epoch 5/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0637 - accuracy: 0.9769 - val_loss: 0.1835 - val_accuracy: 0.9351
Epoch 6/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0389 - accuracy: 0.9863 - val_loss: 0.1121 - val_accuracy: 0.9474
Epoch 7/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0278 - accuracy: 0.9912 - val_loss: 0.1713 - val_accuracy: 0.9229
Epoch 8/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0457 - accuracy: 0.9857 - val_loss: 0.2314 - val_accuracy: 0.9164
Epoch 9/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0231 - accuracy: 0.9927 - val_loss: 0.1160 - val_accuracy: 0.9531
Epoch 10/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0264 - accuracy: 0.9900 - val_loss: 0.0255 - val_accuracy: 0.9913
Epoch 11/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0198 - accuracy: 0.9939 - val_loss: 0.0318 - val_accuracy: 0.9870
Epoch 12/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0192 - accuracy: 0.9927 - val_loss: 0.0582 - val_accuracy: 0.9755
Epoch 13/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0076 - accuracy: 0.9979 - val_loss: 0.1272 - val_accuracy: 0.9603
Epoch 14/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0100 - accuracy: 0.9973 - val_loss: 0.3314 - val_accuracy: 0.9221
Epoch 15/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0117 - accuracy: 0.9960 - val_loss: 0.0371 - val_accuracy: 0.9849
Epoch 16/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0208 - accuracy: 0.9924 - val_loss: 0.2115 - val_accuracy: 0.9200
Epoch 17/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0088 - accuracy: 0.9979 - val_loss: 0.1105 - val_accuracy: 0.9820
Epoch 18/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0084 - accuracy: 0.9973 - val_loss: 0.1597 - val_accuracy: 0.9466
Epoch 19/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0157 - accuracy: 0.9948 - val_loss: 0.1106 - val_accuracy: 0.9676
Epoch 20/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0131 - accuracy: 0.9960 - val_loss: 0.0897 - val_accuracy: 0.9704
Epoch 21/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0348 - accuracy: 0.9900 - val_loss: 0.1953 - val_accuracy: 0.9603
Epoch 22/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0137 - accuracy: 0.9970 - val_loss: 0.1159 - val_accuracy: 0.9683
Epoch 23/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0036 - accuracy: 0.9994 - val_loss: 0.0925 - val_accuracy: 0.9834
Epoch 24/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0072 - accuracy: 0.9976 - val_loss: 0.2563 - val_accuracy: 0.9452
Epoch 25/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0171 - accuracy: 0.9942 - val_loss: 0.4279 - val_accuracy: 0.9048
Epoch 26/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0228 - accuracy: 0.9921 - val_loss: 0.0756 - val_accuracy: 0.9740
Epoch 27/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0094 - accuracy: 0.9982 - val_loss: 0.1103 - val_accuracy: 0.9676
Epoch 28/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0082 - accuracy: 0.9979 - val_loss: 0.7601 - val_accuracy: 0.7960
Epoch 29/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0054 - accuracy: 0.9982 - val_loss: 0.0901 - val_accuracy: 0.9683
Epoch 30/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0118 - accuracy: 0.9954 - val_loss: 0.0485 - val_accuracy: 0.9885
Epoch 31/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0059 - accuracy: 0.9976 - val_loss: 0.2297 - val_accuracy: 0.9438
Epoch 32/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0127 - accuracy: 0.9960 - val_loss: 0.4673 - val_accuracy: 0.8789
Epoch 33/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0048 - accuracy: 0.9985 - val_loss: 0.0374 - val_accuracy: 0.9834
Epoch 34/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0120 - accuracy: 0.9963 - val_loss: 0.1456 - val_accuracy: 0.9676
Epoch 35/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0176 - accuracy: 0.9954 - val_loss: 0.0827 - val_accuracy: 0.9813
Epoch 36/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0069 - accuracy: 0.9985 - val_loss: 0.0694 - val_accuracy: 0.9841
Epoch 37/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0038 - accuracy: 0.9991 - val_loss: 0.0574 - val_accuracy: 0.9856
Epoch 38/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0251 - accuracy: 0.9942 - val_loss: 0.0454 - val_accuracy: 0.9820
Epoch 39/100
```

```
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0124 - accuracy: 0.9957 - val_loss: 0.1291 - val_accuracy: 0.9640
Epoch 40/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0057 - accuracy: 0.9985 - val_loss: 0.0878 - val_accuracy: 0.9726
Epoch 41/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0069 - accuracy: 0.9982 - val_loss: 0.2877 - val_accuracy: 0.9135
Epoch 42/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0079 - accuracy: 0.9970 - val_loss: 0.0291 - val_accuracy: 0.9877
Epoch 43/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0066 - accuracy: 0.9976 - val_loss: 0.0390 - val_accuracy: 0.9877
Epoch 44/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0017 - accuracy: 0.9997 - val_loss: 0.0382 - val_accuracy: 0.9863
Epoch 45/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0094 - accuracy: 0.9979 - val_loss: 0.0448 - val_accuracy: 0.9877
Epoch 46/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0338 - accuracy: 0.9903 - val_loss: 0.0528 - val_accuracy: 0.9834
Epoch 47/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0142 - accuracy: 0.9957 - val_loss: 0.0384 - val_accuracy: 0.9849
Epoch 48/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0089 - accuracy: 0.9973 - val_loss: 0.0308 - val_accuracy: 0.9899
Epoch 49/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0087 - accuracy: 0.9973 - val_loss: 0.0918 - val_accuracy: 0.9755
Epoch 50/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0053 - accuracy: 0.9985 - val_loss: 0.0269 - val_accuracy: 0.9906
Epoch 51/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0014 - accuracy: 0.9994 - val_loss: 0.0243 - val_accuracy: 0.9928
Epoch 52/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0012 - accuracy: 0.9997 - val_loss: 0.0216 - val_accuracy: 0.9928
Epoch 53/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0018 - accuracy: 0.9994 - val_loss: 0.0346 - val_accuracy: 0.9885
Epoch 54/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0023 - accuracy: 0.9994 - val_loss: 0.0400 - val_accuracy: 0.9863
Epoch 55/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0113 - accuracy: 0.9991 - val_loss: 0.0365 - val_accuracy: 0.9892
Epoch 56/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0130 - accuracy: 0.9982 - val_loss: 0.0220 - val_accuracy: 0.9921
Epoch 57/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0059 - accuracy: 0.9979 - val_loss: 0.0529 - val_accuracy: 0.9784
Epoch 58/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0050 - accuracy: 0.9988 - val_loss: 0.0476 - val_accuracy: 0.9885
Epoch 59/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0089 - accuracy: 0.9963 - val_loss: 0.0445 - val_accuracy: 0.9827
Epoch 60/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0030 - accuracy: 0.9988 - val_loss: 0.1116 - val_accuracy: 0.9776
Epoch 61/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0056 - accuracy: 0.9982 - val_loss: 0.0857 - val_accuracy: 0.9784
Epoch 62/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0039 - accuracy: 0.9985 - val_loss: 0.0352 - val_accuracy: 0.9870
Epoch 63/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0011 - accuracy: 0.9997 - val_loss: 0.0419 - val_accuracy: 0.9892
Epoch 64/100
3285/3285 [==============================] - 5s 1ms/step - loss: 5.7950e-04 - accuracy: 1.0000 - val_loss: 0.0306 - val_accuracy: 0.9913
Epoch 65/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0044 - accuracy: 0.9985 - val_loss: 0.0535 - val_accuracy: 0.9899
Epoch 66/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0018 - accuracy: 0.9997 - val_loss: 0.0362 - val_accuracy: 0.9921
Epoch 67/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0032 - accuracy: 0.9991 - val_loss: 0.1680 - val_accuracy: 0.9762
Epoch 68/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0041 - accuracy: 0.9985 - val_loss: 0.1089 - val_accuracy: 0.9827
Epoch 69/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0182 - accuracy: 0.9951 - val_loss: 0.7204 - val_accuracy: 0.8392
Epoch 70/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0052 - accuracy: 0.9982 - val_loss: 0.0862 - val_accuracy: 0.9704
Epoch 71/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0017 - accuracy: 0.9997 - val_loss: 0.0861 - val_accuracy: 0.9805
Epoch 72/100
3285/3285 [==============================] - 5s 1ms/step - loss: 7.8277e-04 - accuracy: 1.0000 - val_loss: 0.1057 - val_accuracy: 0.9805
Epoch 73/100
3285/3285 [==============================] - 5s 1ms/step - loss: 3.9598e-04 - accuracy: 1.0000 - val_loss: 0.0818 - val_accuracy: 0.9820
Epoch 74/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0136 - accuracy: 0.9970 - val_loss: 0.0733 - val_accuracy: 0.9827
Epoch 75/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0118 - accuracy: 0.9976 - val_loss: 0.0454 - val_accuracy: 0.9849
Epoch 76/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0030 - accuracy: 0.9988 - val_loss: 0.0172 - val_accuracy: 0.9921
Epoch 77/100
3285/3285 [==============================] - 5s 1ms/step - loss: 8.7805e-04 - accuracy: 0.9997 - val_loss: 0.0196 - val_accuracy: 0.9928
Epoch 78/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0101 - accuracy: 0.9967 - val_loss: 0.0472 - val_accuracy: 0.9885
Epoch 79/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0019 - accuracy: 0.9994 - val_loss: 0.0209 - val_accuracy: 0.9906
Epoch 80/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0017 - accuracy: 0.9997 - val_loss: 0.0297 - val_accuracy: 0.9870
```

```
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0017 - accuracy: 0.9997 - val_loss: 0.0397 - val_accuracy: 0.9870
Epoch 81/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0035 - accuracy: 0.9991 - val_loss: 0.0209 - val_accuracy: 0.9913
Epoch 82/100
3285/3285 [==============================] - 5s 1ms/step - loss: 9.1677e-04 - accuracy: 0.9997 - val_loss: 0.0757 - val_accuracy: 0.9834
Epoch 83/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0017 - accuracy: 0.9997 - val_loss: 0.0792 - val_accuracy: 0.9849
Epoch 84/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0028 - accuracy: 0.9991 - val_loss: 0.0174 - val_accuracy: 0.9935
Epoch 85/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0047 - accuracy: 0.9985 - val_loss: 0.0373 - val_accuracy: 0.9906
Epoch 86/100
3285/3285 [==============================] - 5s 1ms/step - loss: 7.0720e-04 - accuracy: 1.0000 - val_loss: 0.0218 - val_accuracy: 0.9928
Epoch 87/100
3285/3285 [==============================] - 5s 1ms/step - loss: 5.0616e-04 - accuracy: 1.0000 - val_loss: 0.0496 - val_accuracy: 0.9899
Epoch 88/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0020 - accuracy: 0.9991 - val_loss: 0.0031 - val_accuracy: 0.9986
Epoch 89/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0016 - accuracy: 0.9997 - val_loss: 0.0456 - val_accuracy: 0.9877
Epoch 90/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0044 - accuracy: 0.9985 - val_loss: 0.2035 - val_accuracy: 0.9575
Epoch 91/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0015 - accuracy: 0.9994 - val_loss: 0.0421 - val_accuracy: 0.9870
Epoch 92/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0045 - accuracy: 0.9982 - val_loss: 0.0310 - val_accuracy: 0.9892
Epoch 93/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0049 - accuracy: 0.9988 - val_loss: 0.0092 - val_accuracy: 0.9964
Epoch 94/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0076 - accuracy: 0.9982 - val_loss: 0.0144 - val_accuracy: 0.9942
Epoch 95/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0046 - accuracy: 0.9982 - val_loss: 0.0080 - val_accuracy: 0.9986
Epoch 96/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0069 - accuracy: 0.9982 - val_loss: 0.0536 - val_accuracy: 0.9798
Epoch 97/100
3285/3285 [==============================] - 5s 2ms/step - loss: 7.1112e-04 - accuracy: 0.9997 - val_loss: 0.0519 - val_accuracy: 0.9849
Epoch 98/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0011 - accuracy: 0.9994 - val_loss: 0.0513 - val_accuracy: 0.9849
Epoch 99/100
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0020 - accuracy: 0.9991 - val_loss: 0.0767 - val_accuracy: 0.9762
Epoch 100/100
3285/3285 [==============================] - 5s 2ms/step - loss: 0.0018 - accuracy: 0.9997 - val_loss: 0.0441 - val_accuracy: 0.9856
```

In [124]:

```python
s= m.evaluate(X_test, Y_test)
```

```
1387/1387 [==============================] - 0s 307us/step
```

In [125]:

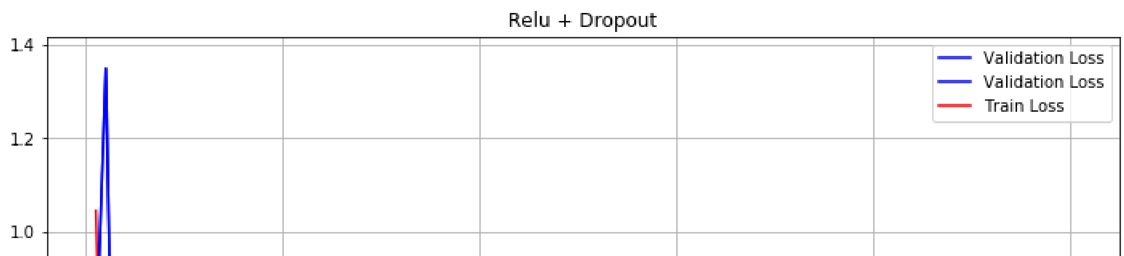
```python
s
```

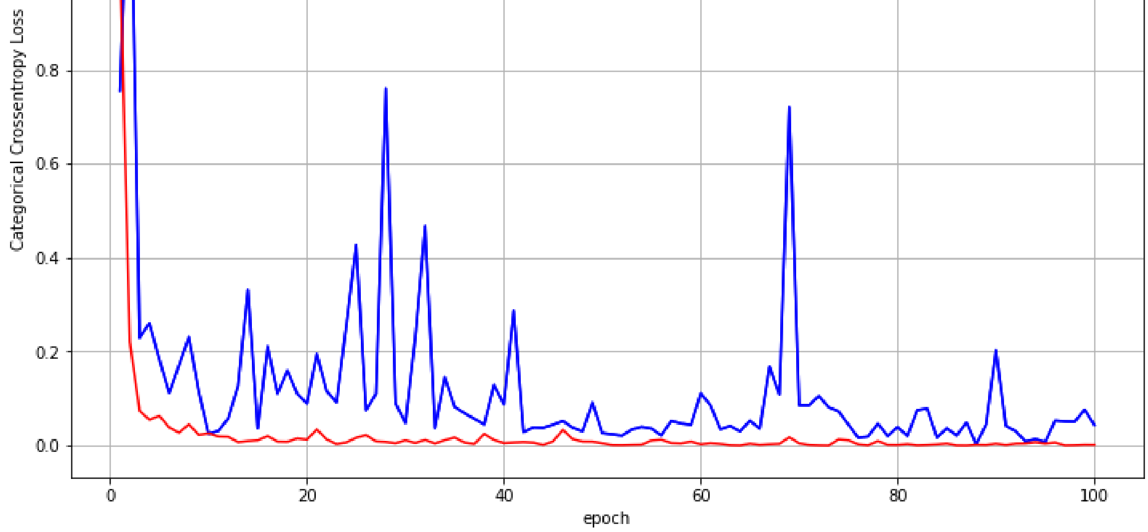
Out[125]:

```
[0.04413533313056151, 0.9855803847312927]
```

In [126]:

```python
fig, ax = plt.subplots(1,1, figsize = (12, 8))
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')
plt.title('Relu + Dropout')

# list of epoch numbers: epoch = 100
x = list(range(1,100+1))
vy = h.history['val_loss']
ty = h.history['loss']
plt_dynamic(x, vy, ty, ax)
```



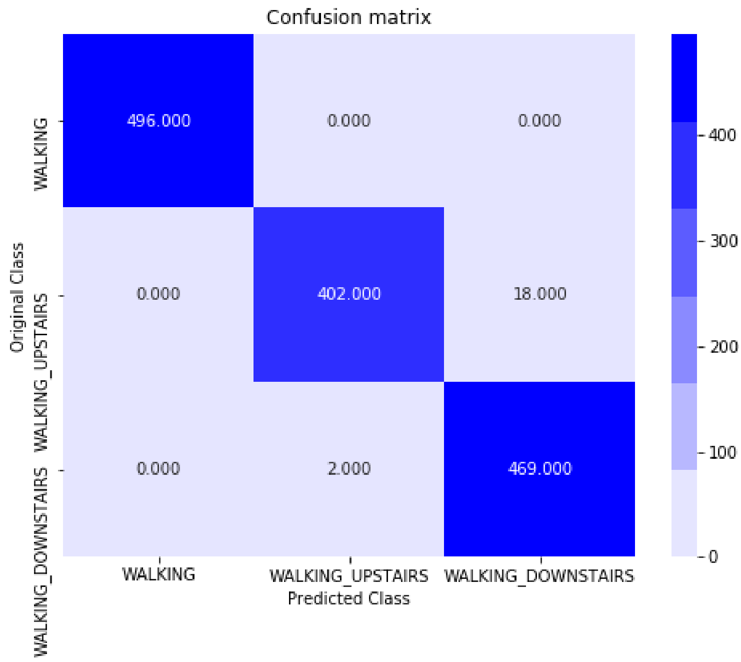


In [127]:

```
confusion_matrix(Y_test, m.predict(X_test))
```

Out[127]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f8e45026c50>



In [128]:

```
m.save('model3')
```

Divide + Conquer Prediction

In [129]:

```
# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
# Triaxial acceleration from the accelerometer is total acceleration

SIGNALS = ["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"]

# Labelling the classes in y.
label = {0:'WALKING', 1:'WALKING_UPSTAIRS', 2:'WALKING_DOWNSTAIRS', 3:'SITTING', 4:'STANDING', 5:'LAYING'}
```

In [146]:

```
# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the load
def load_signals(subset):
    signals_data = []
    filename = 'body_acc_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_acc_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_acc_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'body_gyro_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_x_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_y_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())
    filename = 'total_acc_z_{}.txt'.format(subset)
    signals_data.append(_read_csv(filename).as_matrix())


    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))


def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = 'y_{}.txt'.format(subset)
    y = _read_csv(filename)[0]

    return pd.get_dummies(y).as_matrix()

def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_test = load_y('train'), load_y('test')

    return X_train, X_test, y_train, y_test

# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error

def plt_dynamic(x, vy, ty, ax, color = 'b'):
    ax.plot(x, vy, 'b', label = 'Validation Loss')
    ax.plot(x, vy, 'b', label = 'Validation Loss')
    ax.plot(x, ty, 'r', label = 'Train Loss')
    plt.grid()
    plt.legend()
    fig.canvas.draw()
```

In [135]:

```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()

print('X_train shape is: ',X_train.shape)
```

```
print('Y_train shape is: ',Y_train.shape)
print('X_test shape is: ',X_test.shape)
print('Y_test shape is: ',Y_test.shape)
```

```
X_train shape is:  (7352, 128, 9)
Y_train shape is:  (7352, 6)
X_test shape is:  (2947, 128, 9)
Y_test shape is:  (2947, 6)
```

In [136]:

```
# Importing tensorflow
np.random.seed(42)
import tensorflow as tf
tf.set_random_seed(42)
```

In [137]:

```
# Configuring a session
session_conf = tf.ConfigProto(intra_op_parallelism_threads= 1,
                              inter_op_parallelism_threads= 1)
```

In [138]:

```
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

In [139]:

```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [140]:

```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)

print(timesteps)
print(input_dim)
print(len(X_train))
```

```
128
9
7352
```

In [143]:

```
from keras.models import load_model

model1 = load_model('model1')
model2 = load_model('model2')
model3 = load_model('model3')
```

In [147]:

```
# https://github.com/UdiBhaskar/Human-Activity-Recognition--Using-Deep-NN
#predicting output activity

def predict(X):
    ##predicting whether dynamic or static
    predict_binary = model1.predict(X)
    f_predict_binary = np.argmax(predict_binary, axis=1)

    #static data filter
    X_static = X[f_predict_binary==1]

    #dynamic data filter
    X_dynamic = X[f_predict_binary==0]

    #predicting static activities
    predict_static = model2.predict(X_static)
    f_predict_static = np.argmax(predict_static,axis=1)
```

```python
    #adding 3 because need to get inital prediction lable as output
    f_predict_static = f_predict_static + 3

    #predicting dynamic activites
    predict_dynamic = model3.predict(X_dynamic)
    f_predict_dynamic = np.argmax(predict_dynamic,axis=1)

    #adding 1 because need to get inal prediction lable as output
    f_predict_dynamic = f_predict_dynamic

    ##appending final output to one list in the same sequence of input data
    i,j = 0,0
    final_predict = []

    for q_p in f_predict_binary:
        if q_p == 1:
            final_predict.append(f_predict_static[i])
            i = i + 1
        else:
            final_predict.append(f_predict_dynamic[j])
            j = j + 1

    return final_predict
```

In [149]:

```python
from sklearn.metrics import accuracy_score

train_pred = predict(X_train)
test_pred = predict(X_test)

print('Accuracy of train data',accuracy_score(np.argmax(Y_train,axis=1),train_pred))
print('Accuracy of validation data',accuracy_score(np.argmax(Y_test,axis=1),test_pred))
```

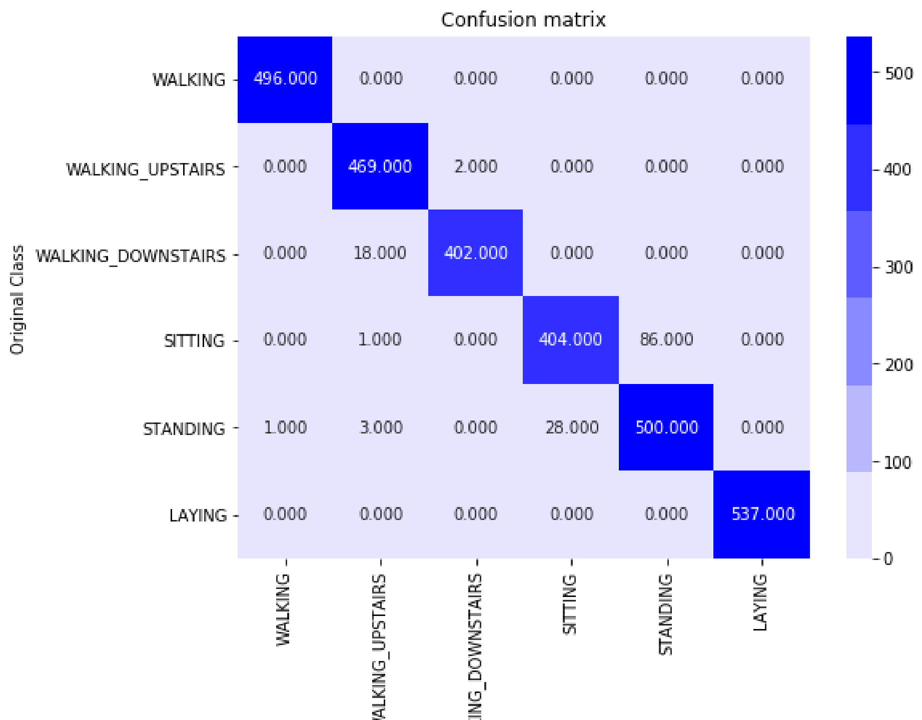
```
Accuracy of train data 0.9895266594124048
Accuracy of validation data 0.9528333898880217
```

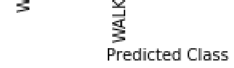
In [183]:

```python
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(np.argmax(Y_test,axis=1), test_pred)
plt.figure(figsize= (8, 6))
sns.heatmap(cm, annot=True, cmap= sns.light_palette("blue"), fmt=".3f", xticklabels=label.values(),
            yticklabels= label.values())
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.title("Confusion matrix")
```

Out[183]:

```
Text(0.5, 1.0, 'Confusion matrix')
```





Conclusions

Due to low computation power (Colab not supported on windows vista), hyperparameter tuning via GridSearchCV was not possible, hence tried different architectures.

In [186]:

```
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ['Rank', 'Model', "Test Accuracy"]

x.add_row([1, "Model-3 Dynamic", "98.60%"])
x.add_row([2, "Model-2 Static", "93.20%"])
x.add_row([3, "Divide + Conquer Model", "95.30%"])
x.add_row([4, "32 LSTM Base Model", "90.20%"])

print(x)
```

```
+------+------------------------+---------------+
| Rank |         Model          | Test Accuracy |
+------+------------------------+---------------+
|  1   |    Model-3 Dynamic     |     98.60%    |
|  2   |     Model-2 Static     |     93.20%    |
|  3   | Divide + Conquer Model |     95.30%    |
|  4   |   32 LSTM Base Model   |     90.20%    |
+------+------------------------+---------------+
```

In [ ]: