HumanActivityRecognition case-study

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.

1.1 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.

1.2Feature 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

1.3 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**

1.4 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***

1.5 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'

1.6 Data Size :

> The size of the data is only 27 MB which is very small as I am doing this case-study in a No-GPU enviorment and I am also going to apply the deep-learning model over the data

1.7 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.

1.8 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.

1.9 Problem Statement

- Given a new datapoint we have to predict the Activity out of the six given human activites

2 Performing Exploratory-data analysis over the HAR dataset

In [4]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings(action='ignore')
from prettytable import PrettyTable

# get the features from the file features.txt
features = list()
with open('UCI_HAR_Dataset/features.txt') as f:
    features = [line.split()[1] for line in f.readlines()]
print('No of Features: {}'.format(len(features)))
```

```
No of Features: 561
```

2.1 Obtain the train data

In [2]:
```
# get the data from txt files to pandas dataffame
X_train = pd.read_csv('UCI_HAR_dataset/train/X_train.txt', delim_whitespace=True, header=None, names=f
eatures)

# add subject column to the dataframe
X_train['subject'] = pd.read_csv('UCI_HAR_dataset/train/subject_train.txt', header=None, squeeze=True)

y_train = pd.read_csv('UCI_HAR_dataset/train/y_train.txt', names=['Activity'], squeeze=True)
y_train_labels = y_train.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',\
                       4:'SITTING', 5:'STANDING',6:'LAYING'})

# put all columns in a single dataframe
train = X_train
train['Activity'] = y_train
train['ActivityName'] = y_train_labels
train.sample()
```

Out[2]:

	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	...
6303	0.267297	-0.010238	-0.097483	-0.940007	-0.918087	-0.948077	-0.950673	-0.921217	-0.948432	-0.860039	...

1 rows × 564 columns

In [3]:
```
train.shape
```

Out[3]:
```
(7352, 564)
```

2.2 Obtain the test data

In [4]:
```
# get the data from txt files to pandas dataffame
X_test = pd.read_csv('UCI_HAR_dataset/test/X_test.txt', delim_whitespace=True, header=None, names=feat
ures)

# add subject column to the dataframe
X_test['subject'] = pd.read_csv('UCI_HAR_dataset/test/subject_test.txt', header=None, squeeze=True)

# get y labels from the txt file
y_test = pd.read_csv('UCI_HAR_dataset/test/y_test.txt', names=['Activity'], squeeze=True)
y_test_labels = y_test.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',\
                       4:'SITTING', 5:'STANDING',6:'LAYING'})


# put all columns in a single dataframe
test = X_test
test['Activity'] = y_test
test['ActivityName'] = y_test_labels
test.sample()
```

Out[4]:

	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	...
1195	0.280002	-0.021817	-0.105898	-0.268965	-0.130949	-0.015931	-0.30796	-0.134914	-0.047938	-0.044352	...

1 rows × 564 columns

2.3 Data Cleaning

2.3.1. Check for Duplicates

In [5]:
```
print('No of duplicates in train: {}'.format(sum(train.duplicated())))
print('No of duplicates in test : {}'.format(sum(test.duplicated())))
```

```
No of duplicates in train: 0
No of duplicates in test : 0
```

2.3.2 Checking for NaN/null values

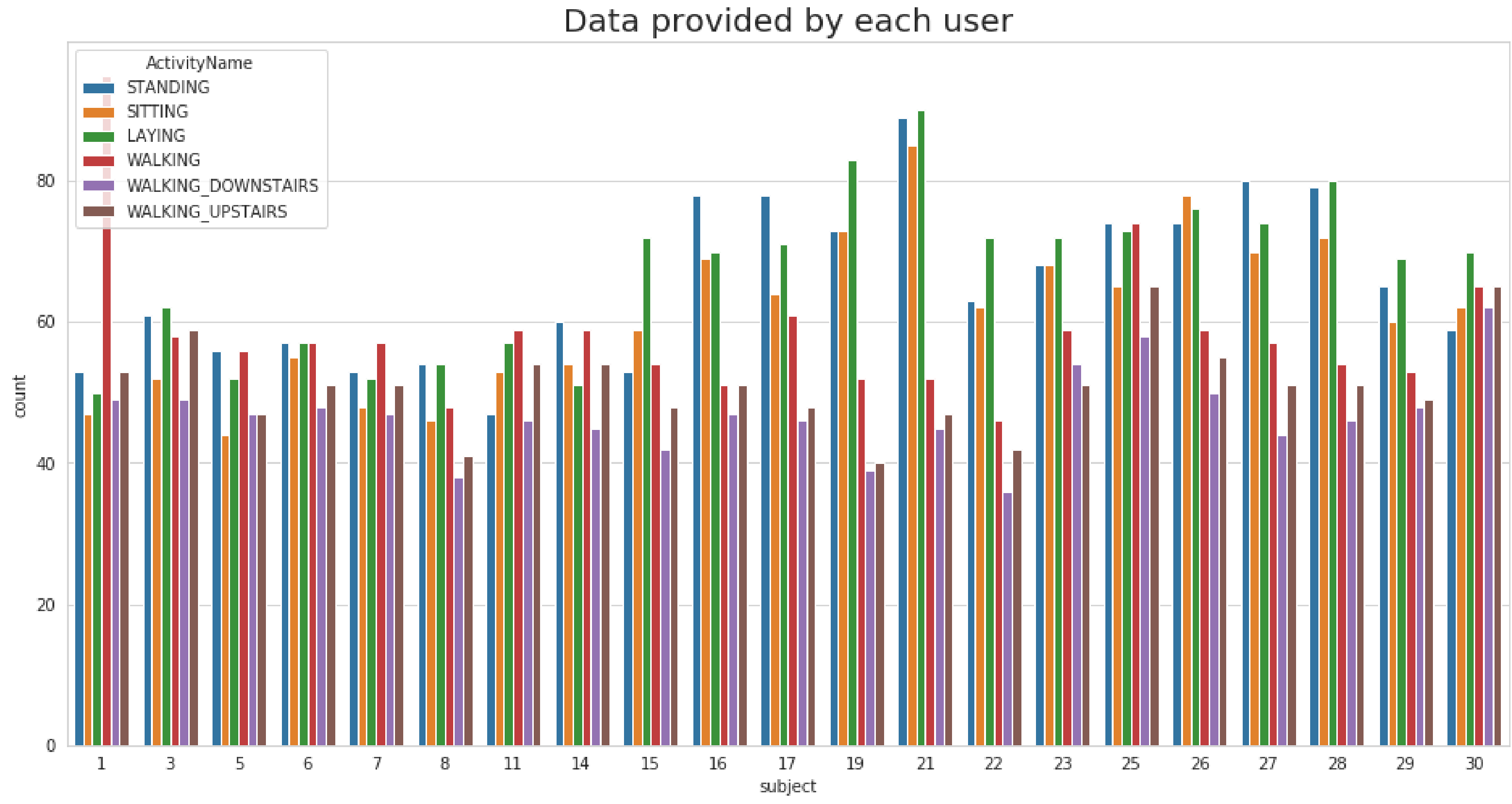
In [6]:
```
print('We have {} NaN/Null values in train'.format(train.isnull().values.sum()))
print('We have {} NaN/Null values in test'.format(test.isnull().values.sum()))
```

```
We have 0 NaN/Null values in train
We have 0 NaN/Null values in test
```

2.3.3. Check for data imbalance

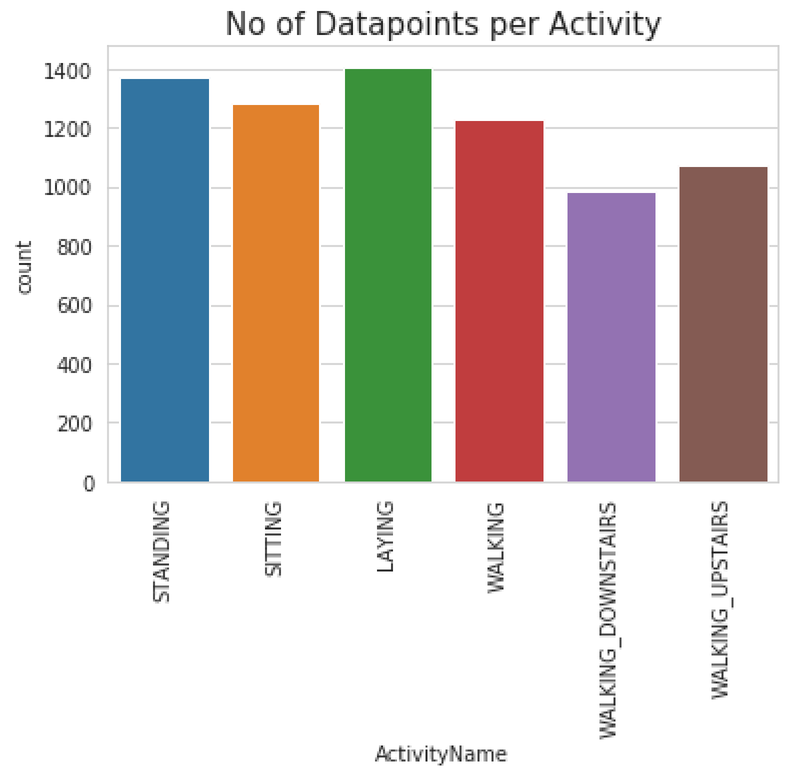
In [7]:
```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style('whitegrid')
plt.rcParams['font.family'] = 'Dejavu Sans'
```

In [8]:
```
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 [9]:
```
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) among all the class labels.

2.4. Changing feature names

In [10]:
```
columns = train.columns

# Removing '()' from column names
columns = columns.str.replace('[()]','')
columns = columns.str.replace('[-]', '')
columns = columns.str.replace('[,]','')

train.columns = columns
test.columns = columns

test.columns
```

Out[10]:
```
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)
```

2.5. Save this dataframe in a csv files

In [11]:
```
train.to_csv('UCI_HAR_Dataset/csv_files/train.csv', index=False)
test.to_csv('UCI_HAR_Dataset/csv_files/test.csv', index=False)
```

3. Performing Univariate Exploratory Data Analysis

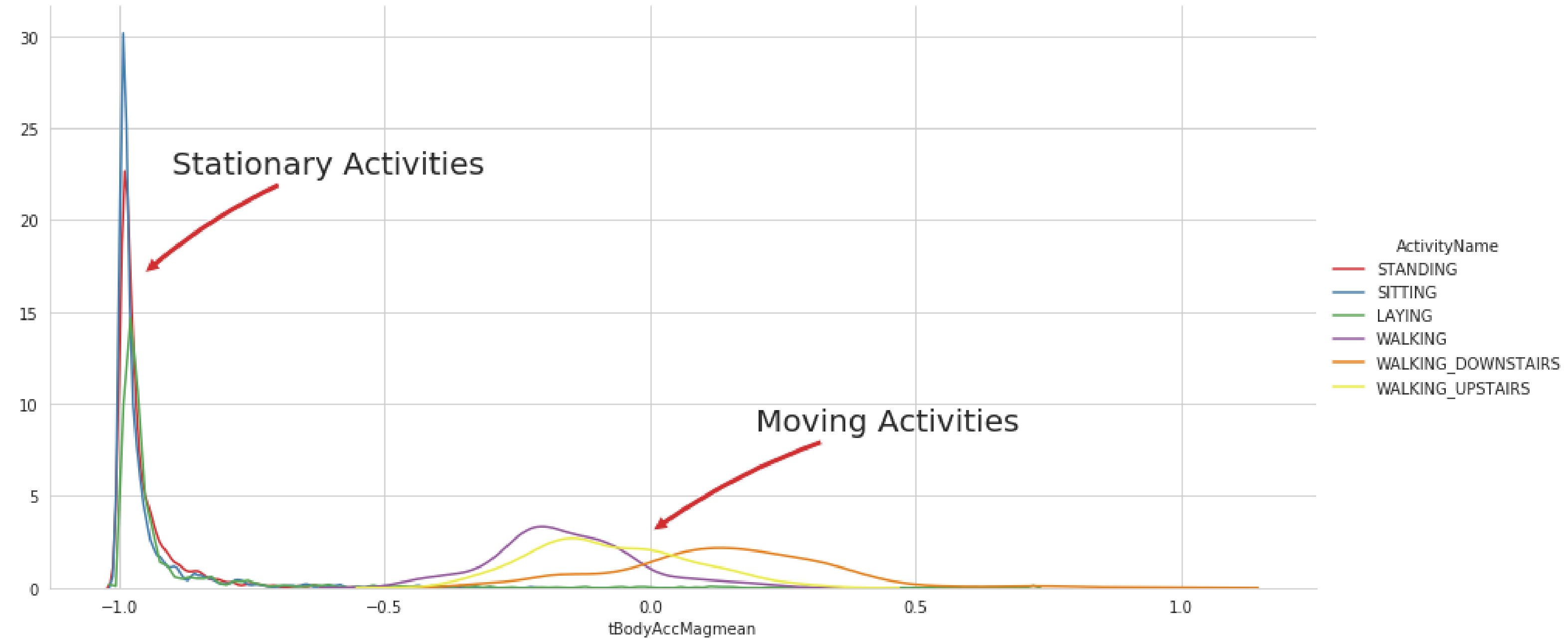
"***Without domain knowledge EDA has no meaning, without EDA a problem has no soul.***"

3.1. Featuring Engineering from Domain Knowledge

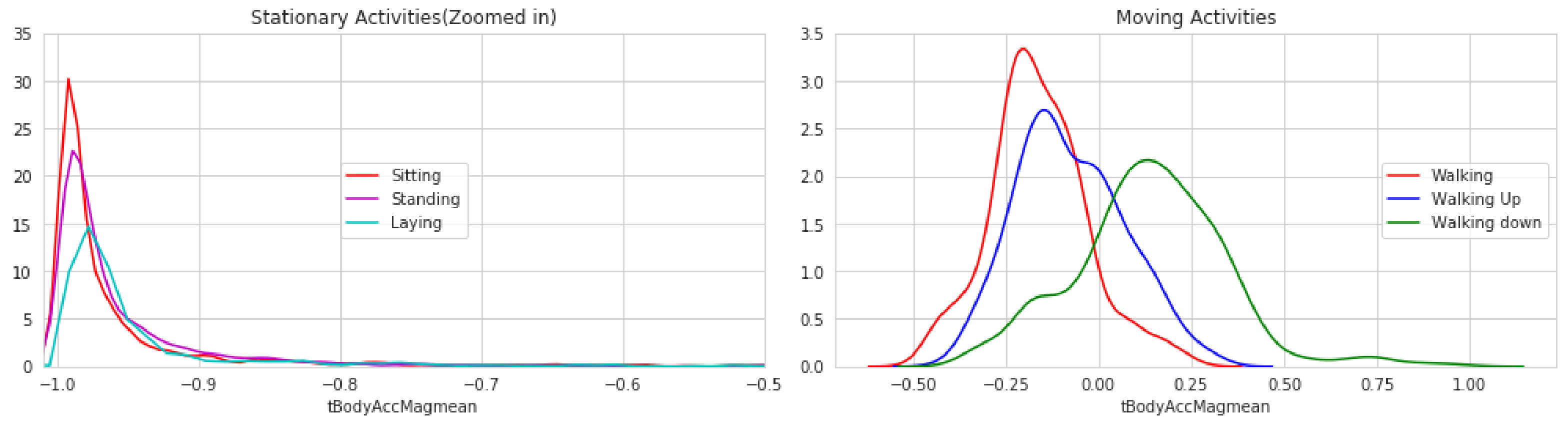
- **Static and Dynamic Activities**
  - In static activities (sit, stand, lie down) motion information will not be very useful.
  - In the dynamic activities (Walking, WalkingUpstairs,WalkingDownstairs) motion info will be significant.

3.1.1. Stationary and Moving activities are completely different

In [37]:
```
sns.set_palette("Set1", desat=0.80)
facetgrid = sns.FacetGrid(train, hue='ActivityName', height=6,aspect=2)
facetgrid.map(sns.distplot,'tBodyAccMagmean', hist=False)\
    .add_legend()
plt.annotate("Stationary Activities", xy=(-0.956,17), xytext=(-0.9, 23), 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()
```

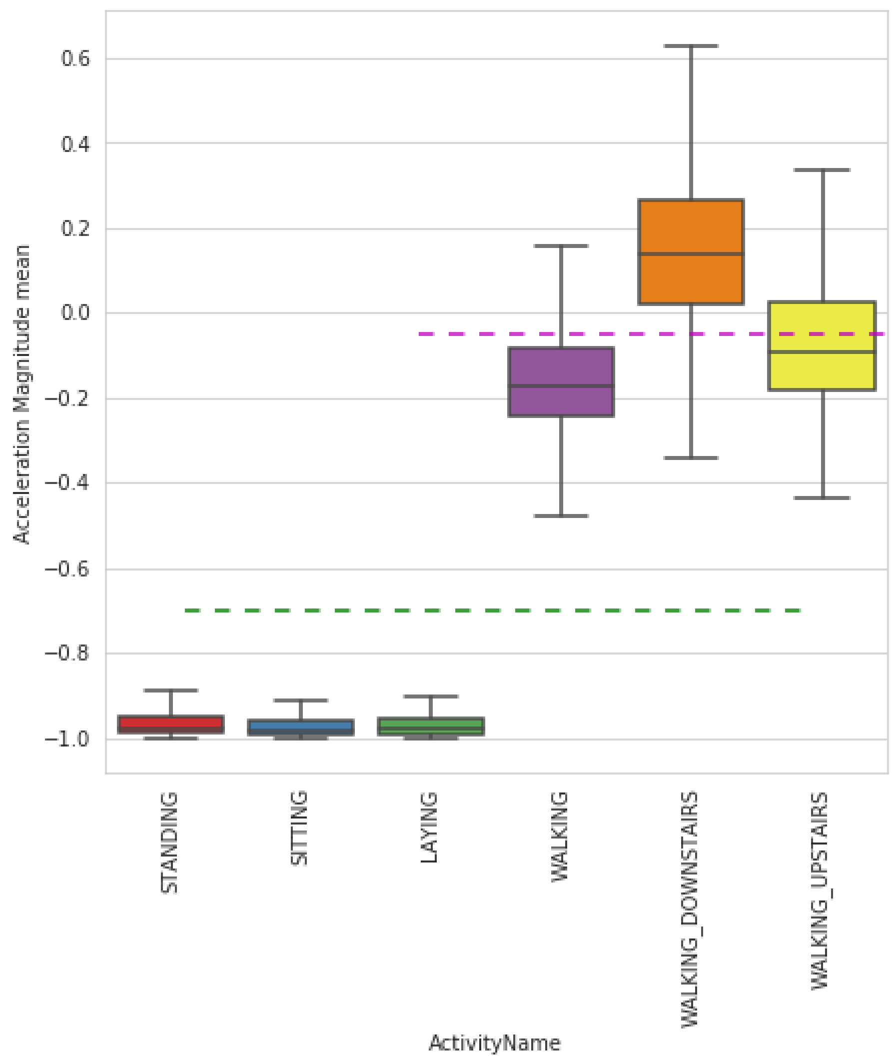


```python
In [18]: # for plotting purposes taking datapoints of each activity to a different dataframe
df1 = train[train['Activity']==1]
df2 = train[train['Activity']==2]
df3 = train[train['Activity']==3]
df4 = train[train['Activity']==4]
df5 = train[train['Activity']==5]
df6 = train[train['Activity']==6]

plt.figure(figsize=(14,7))
plt.subplot(2,2,1)
plt.title('Stationary Activities(Zoomed in)')
sns.distplot(df4['tBodyAccMagmean'],color = 'r',hist = False, label = 'Sitting')
sns.distplot(df5['tBodyAccMagmean'],color = 'm',hist = False,label = 'Standing')
sns.distplot(df6['tBodyAccMagmean'],color = 'c',hist = False, label = 'Laying')
plt.axis([-1.01, -0.5, 0, 35])
plt.legend(loc='center')

plt.subplot(2,2,2)
plt.title('Moving Activities')
sns.distplot(df1['tBodyAccMagmean'],color = 'red',hist = False, label = 'Walking')
sns.distplot(df2['tBodyAccMagmean'],color = 'blue',hist = False,label = 'Walking Up')
sns.distplot(df3['tBodyAccMagmean'],color = 'green',hist = False, label = 'Walking down')
plt.legend(loc='center right')


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



3.1.2. Magnitude of an acceleration can saperate it well

```python
In [19]: 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()
```

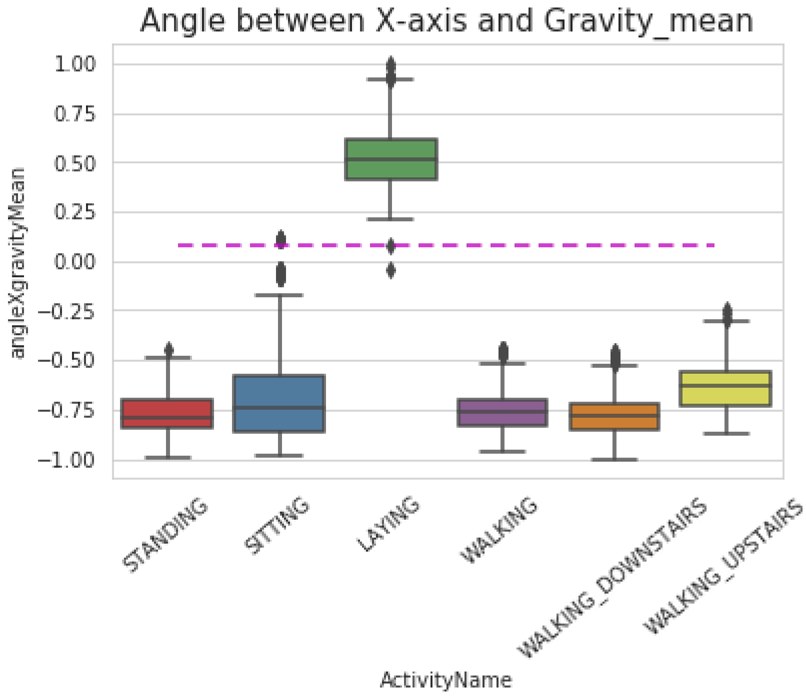


Observations:

- 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 > 0.0 then the Activity is WalkingDownstairs.
- We can classify 75% the Acitivity labels with some errors.|

3.1.3. Position of GravityAccelerationComponants also matters

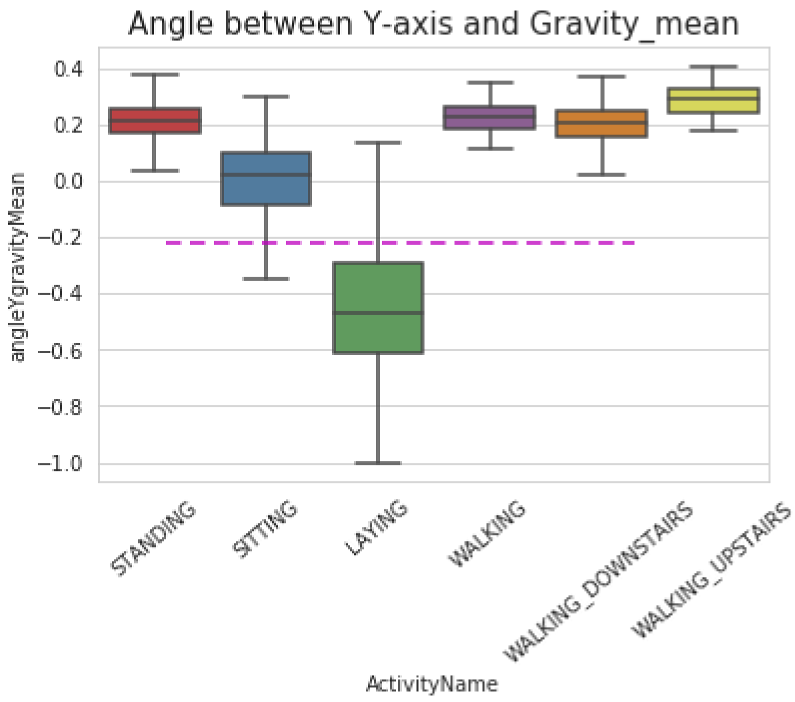
In [20]:
```
sns.boxplot(x='ActivityName', y='angleXgravityMean', data=train)
plt.axhline(y=0.08, xmin=0.1, xmax=0.9,c='m',dashes=(5,3))
plt.title('Angle between X-axis and Gravity_mean', fontsize=15)
plt.xticks(rotation = 40)
plt.show()
```



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 [21]:
```
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()
```



Obsevations

- Using the simple engineered features and doing simple univariate analysis we got very good results.
- All the classes are well separated even by doing a simple univariate analysis except sitting and standing classes.
- so the human engineered features are very usefull in classiffying the model properly and good machine-learning models can be made if all the features are used properly.

4. Apply t-sne on the data

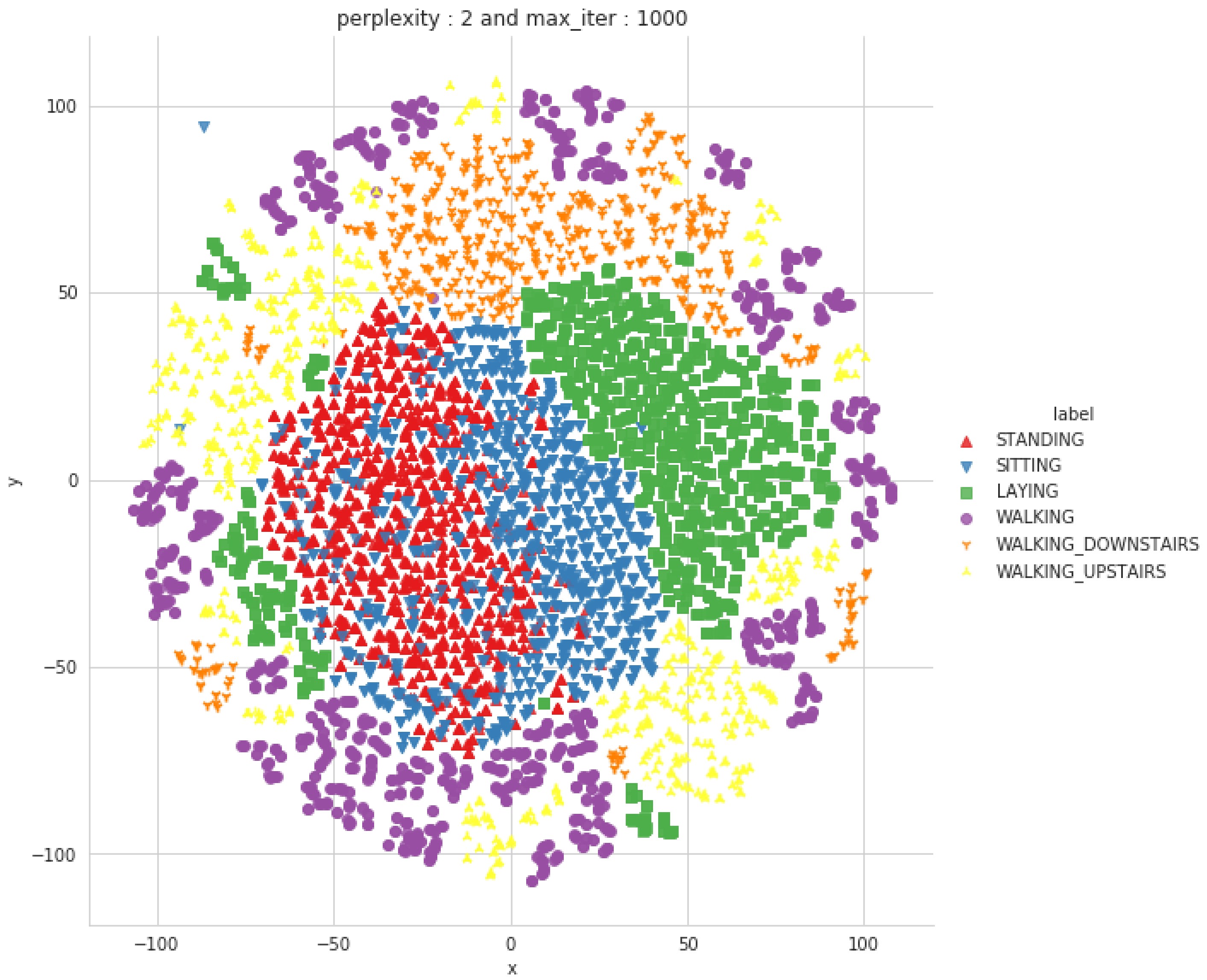
In [22]:
```python
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns
```

In [23]:
```python
# performs t-sne with different perplexity values and their repective plots..

def perform_tsne(X_data, y_data, 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'.format(perplexity,
n_iter))
        X_reduced = TSNE(verbose=2, perplexity=perplexity).fit_transform(X_data)
        print('Done..')

        # prepare the data for seaborn
        print('Creating plot for this t-sne visualization..')
        df = pd.DataFrame({'x':X_reduced[:,0], 'y':X_reduced[:,1] ,'label':y_data})

        # draw the plot in appropriate place in the grid
        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 + '_perp_{}_iter_{}.png'.format(perplexity, n_iter)
        print('saving this plot as image in present working directory...')
        plt.savefig(img_name)
        plt.show()
        print('Done')
```

In [23]:
```python
X_pre_tsne = train.drop(['subject', 'Activity','ActivityName'], axis=1)
y_pre_tsne = train['ActivityName']
perform_tsne(X_data = X_pre_tsne,y_data=y_pre_tsne, perplexities =[2,5,10,20,50])
```

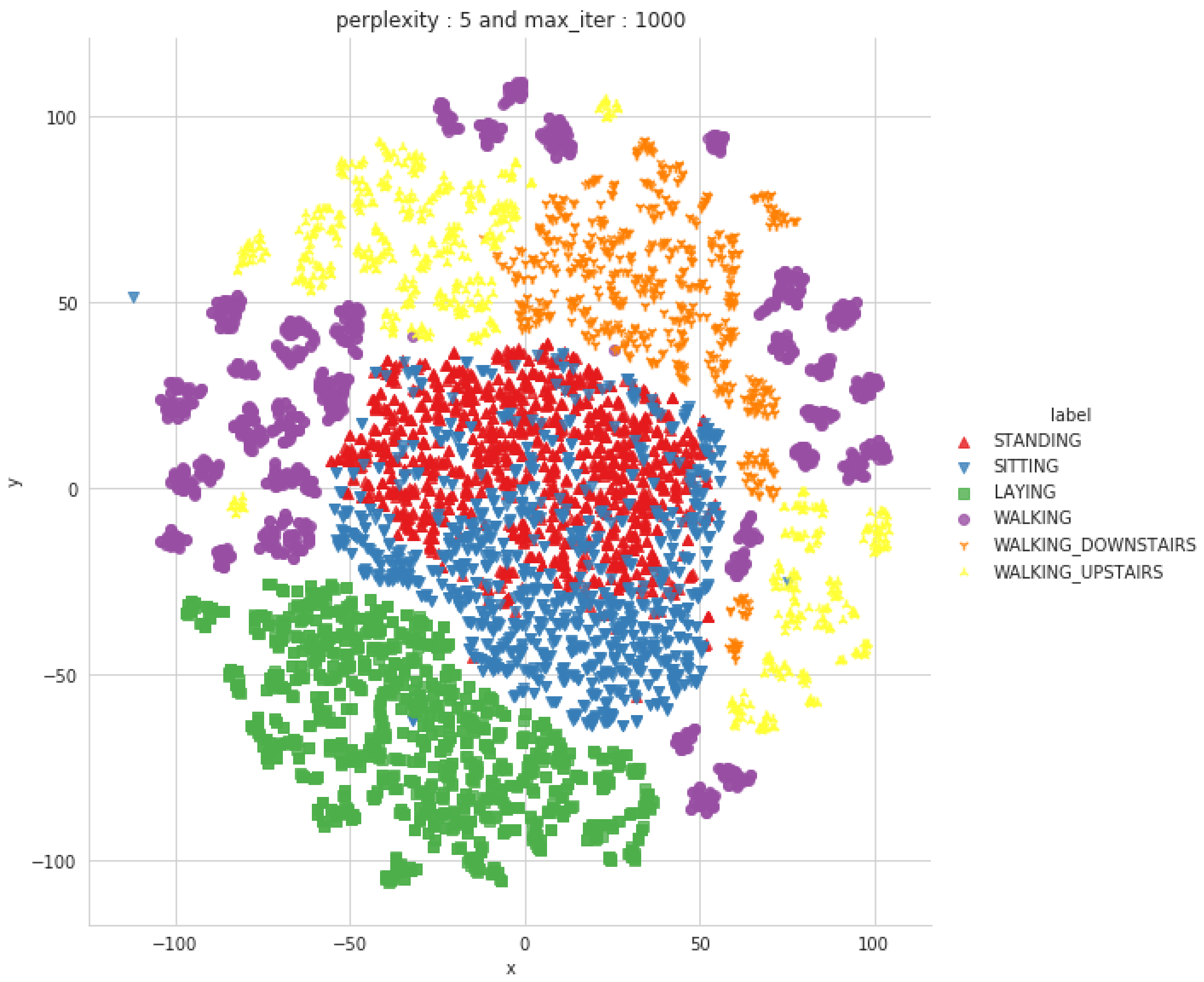
```
performing tsne with perplexity 2 and with 1000 iterations at max
[t-SNE] Computing 7 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.184s...
[t-SNE] Computed neighbors for 7352 samples in 30.912s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 0.635855
[t-SNE] Computed conditional probabilities in 0.024s
[t-SNE] Iteration 50: error = 124.6122284, gradient norm = 0.0269497 (50 iterations in 6.319s)
[t-SNE] Iteration 100: error = 106.9878693, gradient norm = 0.0296072 (50 iterations in 4.518s)
[t-SNE] Iteration 150: error = 100.7272568, gradient norm = 0.0178829 (50 iterations in 3.627s)
[t-SNE] Iteration 200: error = 97.3601303, gradient norm = 0.0203624 (50 iterations in 3.509s)
[t-SNE] Iteration 250: error = 95.0759125, gradient norm = 0.0154472 (50 iterations in 3.528s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 95.075912
[t-SNE] Iteration 300: error = 4.1169338, gradient norm = 0.0015607 (50 iterations in 3.463s)
[t-SNE] Iteration 350: error = 3.2088828, gradient norm = 0.0009879 (50 iterations in 3.603s)
[t-SNE] Iteration 400: error = 2.7790430, gradient norm = 0.0007265 (50 iterations in 3.701s)
[t-SNE] Iteration 450: error = 2.5150440, gradient norm = 0.0005654 (50 iterations in 3.797s)
[t-SNE] Iteration 500: error = 2.3317556, gradient norm = 0.0004760 (50 iterations in 3.762s)
[t-SNE] Iteration 550: error = 2.1935050, gradient norm = 0.0004162 (50 iterations in 3.783s)
[t-SNE] Iteration 600: error = 2.0841215, gradient norm = 0.0003710 (50 iterations in 3.937s)
[t-SNE] Iteration 650: error = 1.9943038, gradient norm = 0.0003338 (50 iterations in 3.938s)
[t-SNE] Iteration 700: error = 1.9188459, gradient norm = 0.0003047 (50 iterations in 3.783s)
[t-SNE] Iteration 750: error = 1.8538190, gradient norm = 0.0002738 (50 iterations in 3.753s)
[t-SNE] Iteration 800: error = 1.7973059, gradient norm = 0.0002579 (50 iterations in 3.748s)
[t-SNE] Iteration 850: error = 1.7473668, gradient norm = 0.0002378 (50 iterations in 3.798s)
[t-SNE] Iteration 900: error = 1.7029767, gradient norm = 0.0002243 (50 iterations in 3.802s)
[t-SNE] Iteration 950: error = 1.6628928, gradient norm = 0.0002128 (50 iterations in 3.740s)
[t-SNE] Iteration 1000: error = 1.6266612, gradient norm = 0.0002001 (50 iterations in 3.821s)
[t-SNE] Error after 1000 iterations: 1.626661
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```

```
c:\users\admin\appdata\local\programs\python\python36\lib\site-packages\seaborn\regression.py:546: Use
rWarning: The `size` paramter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



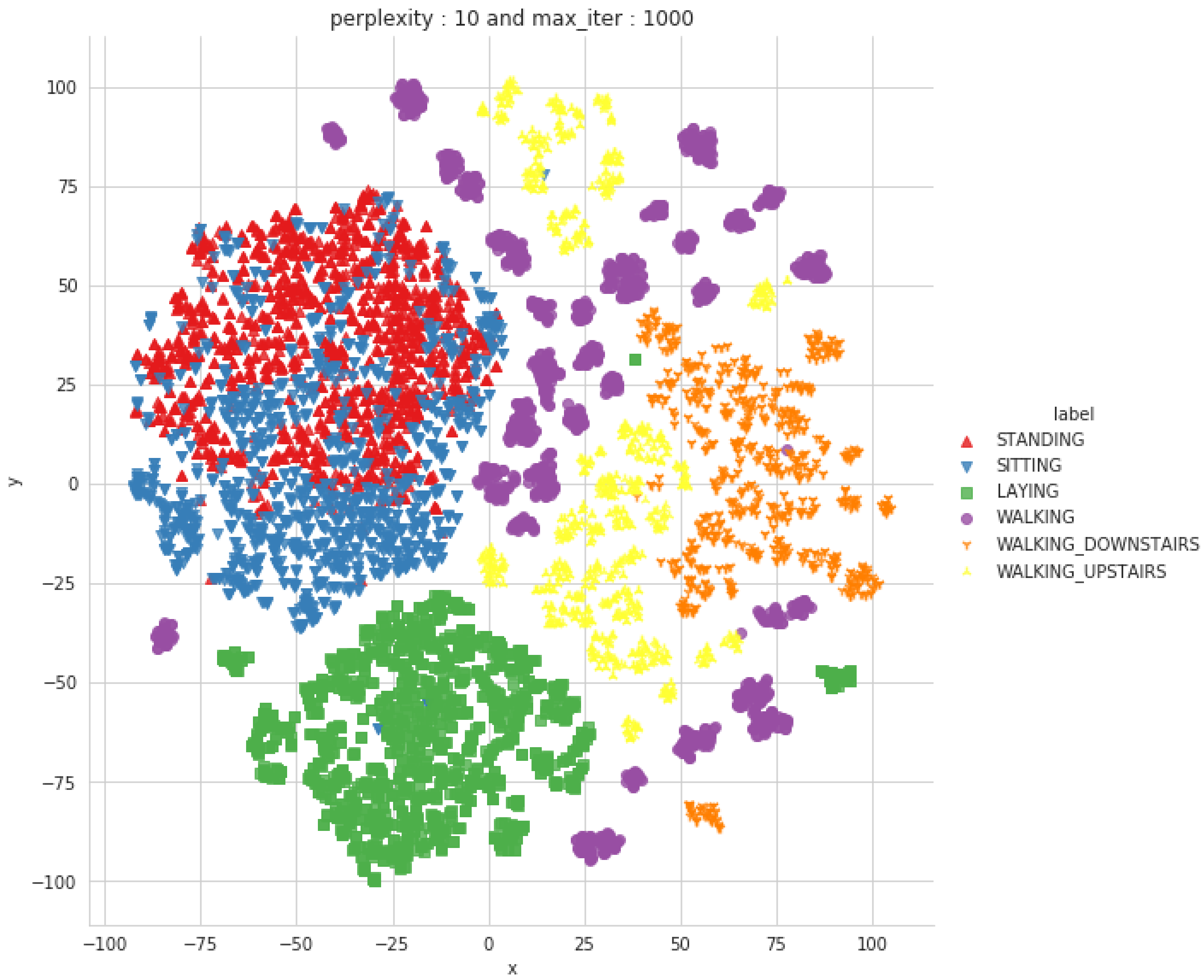

```
Done

performing tsne with perplexity 5 and with 1000 iterations at max
[t-SNE] Computing 16 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.187s...
[t-SNE] Computed neighbors for 7352 samples in 31.523s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 0.961265
[t-SNE] Computed conditional probabilities in 0.039s
[t-SNE] Iteration 50: error = 114.0459518, gradient norm = 0.0192646 (50 iterations in 24.402s)
[t-SNE] Iteration 100: error = 98.0626297, gradient norm = 0.0180720 (50 iterations in 4.764s)
[t-SNE] Iteration 150: error = 93.2986221, gradient norm = 0.0083993 (50 iterations in 4.179s)
[t-SNE] Iteration 200: error = 91.2765656, gradient norm = 0.0070595 (50 iterations in 4.166s)
[t-SNE] Iteration 250: error = 90.0960846, gradient norm = 0.0049592 (50 iterations in 4.121s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 90.096085
[t-SNE] Iteration 300: error = 3.5756156, gradient norm = 0.0014665 (50 iterations in 4.159s)
[t-SNE] Iteration 350: error = 2.8165112, gradient norm = 0.0007453 (50 iterations in 4.111s)
[t-SNE] Iteration 400: error = 2.4358370, gradient norm = 0.0005287 (50 iterations in 4.186s)
[t-SNE] Iteration 450: error = 2.2188416, gradient norm = 0.0004047 (50 iterations in 4.206s)
[t-SNE] Iteration 500: error = 2.0746720, gradient norm = 0.0003342 (50 iterations in 4.148s)
[t-SNE] Iteration 550: error = 1.9696110, gradient norm = 0.0002836 (50 iterations in 4.113s)
[t-SNE] Iteration 600: error = 1.8888149, gradient norm = 0.0002481 (50 iterations in 4.174s)
[t-SNE] Iteration 650: error = 1.8240607, gradient norm = 0.0002170 (50 iterations in 4.129s)
[t-SNE] Iteration 700: error = 1.7704110, gradient norm = 0.0001996 (50 iterations in 4.130s)
[t-SNE] Iteration 750: error = 1.7253617, gradient norm = 0.0001803 (50 iterations in 4.167s)
[t-SNE] Iteration 800: error = 1.6865208, gradient norm = 0.0001677 (50 iterations in 4.190s)
[t-SNE] Iteration 850: error = 1.6533352, gradient norm = 0.0001516 (50 iterations in 4.178s)
[t-SNE] Iteration 900: error = 1.6237185, gradient norm = 0.0001432 (50 iterations in 4.214s)
[t-SNE] Iteration 950: error = 1.5972966, gradient norm = 0.0001336 (50 iterations in 4.211s)
[t-SNE] Iteration 1000: error = 1.5739222, gradient norm = 0.0001264 (50 iterations in 4.178s)
[t-SNE] Error after 1000 iterations: 1.573922
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```

```
c:\users\admin\appdata\local\programs\python\python36\lib\site-packages\seaborn\regression.py:546: UserWarning: The `size` paramter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



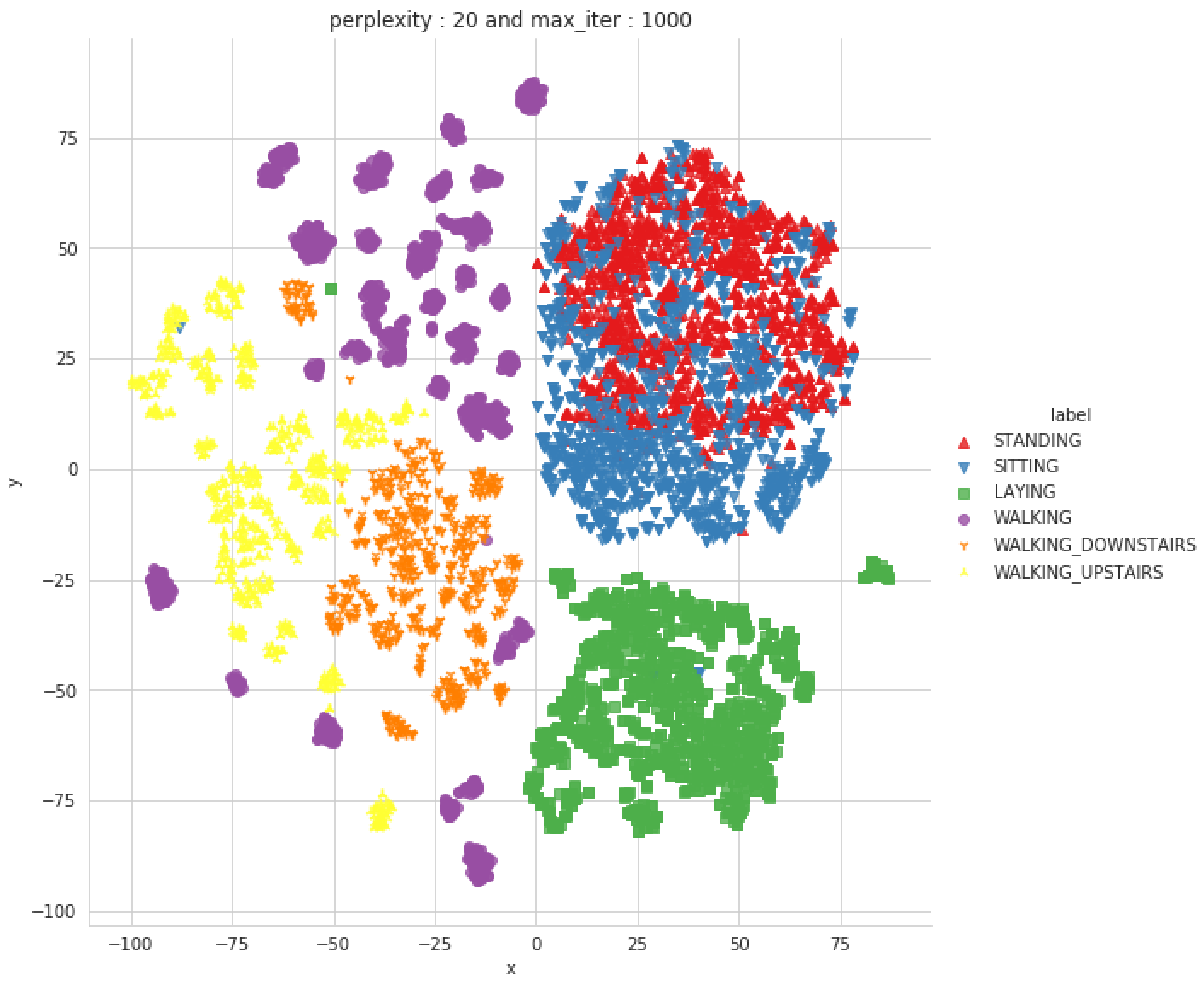

```
Done

performing tsne with perplexity 10 and with 1000 iterations at max
[t-SNE] Computing 31 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.185s...
[t-SNE] Computed neighbors for 7352 samples in 32.420s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.133828
[t-SNE] Computed conditional probabilities in 0.074s
[t-SNE] Iteration 50: error = 105.8530350, gradient norm = 0.0182692 (50 iterations in 7.259s)
[t-SNE] Iteration 100: error = 90.4470215, gradient norm = 0.0119179 (50 iterations in 5.421s)
[t-SNE] Iteration 150: error = 87.2844620, gradient norm = 0.0048722 (50 iterations in 4.867s)
[t-SNE] Iteration 200: error = 86.0391769, gradient norm = 0.0036453 (50 iterations in 4.769s)
[t-SNE] Iteration 250: error = 85.3395996, gradient norm = 0.0037806 (50 iterations in 4.717s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.339600
[t-SNE] Iteration 300: error = 3.1361799, gradient norm = 0.0013883 (50 iterations in 4.598s)
[t-SNE] Iteration 350: error = 2.4927766, gradient norm = 0.0006485 (50 iterations in 4.575s)
[t-SNE] Iteration 400: error = 2.1728568, gradient norm = 0.0004229 (50 iterations in 4.616s)
[t-SNE] Iteration 450: error = 1.9880606, gradient norm = 0.0003154 (50 iterations in 4.692s)
[t-SNE] Iteration 500: error = 1.8694317, gradient norm = 0.0002569 (50 iterations in 4.778s)
[t-SNE] Iteration 550: error = 1.7861626, gradient norm = 0.0002095 (50 iterations in 4.762s)
[t-SNE] Iteration 600: error = 1.7234719, gradient norm = 0.0001812 (50 iterations in 4.803s)
[t-SNE] Iteration 650: error = 1.6743854, gradient norm = 0.0001589 (50 iterations in 4.675s)
[t-SNE] Iteration 700: error = 1.6346445, gradient norm = 0.0001432 (50 iterations in 4.695s)
[t-SNE] Iteration 750: error = 1.6019816, gradient norm = 0.0001288 (50 iterations in 4.773s)
[t-SNE] Iteration 800: error = 1.5744828, gradient norm = 0.0001181 (50 iterations in 4.764s)
[t-SNE] Iteration 850: error = 1.5510921, gradient norm = 0.0001091 (50 iterations in 4.785s)
[t-SNE] Iteration 900: error = 1.5308635, gradient norm = 0.0001028 (50 iterations in 4.799s)
[t-SNE] Iteration 950: error = 1.5134670, gradient norm = 0.0000970 (50 iterations in 4.816s)
[t-SNE] Iteration 1000: error = 1.4981405, gradient norm = 0.0000913 (50 iterations in 4.795s)
[t-SNE] Error after 1000 iterations: 1.498140
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```

```
c:\users\admin\appdata\local\programs\python\python36\lib\site-packages\seaborn\regression.py:546: Use
rWarning: The `size` paramter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



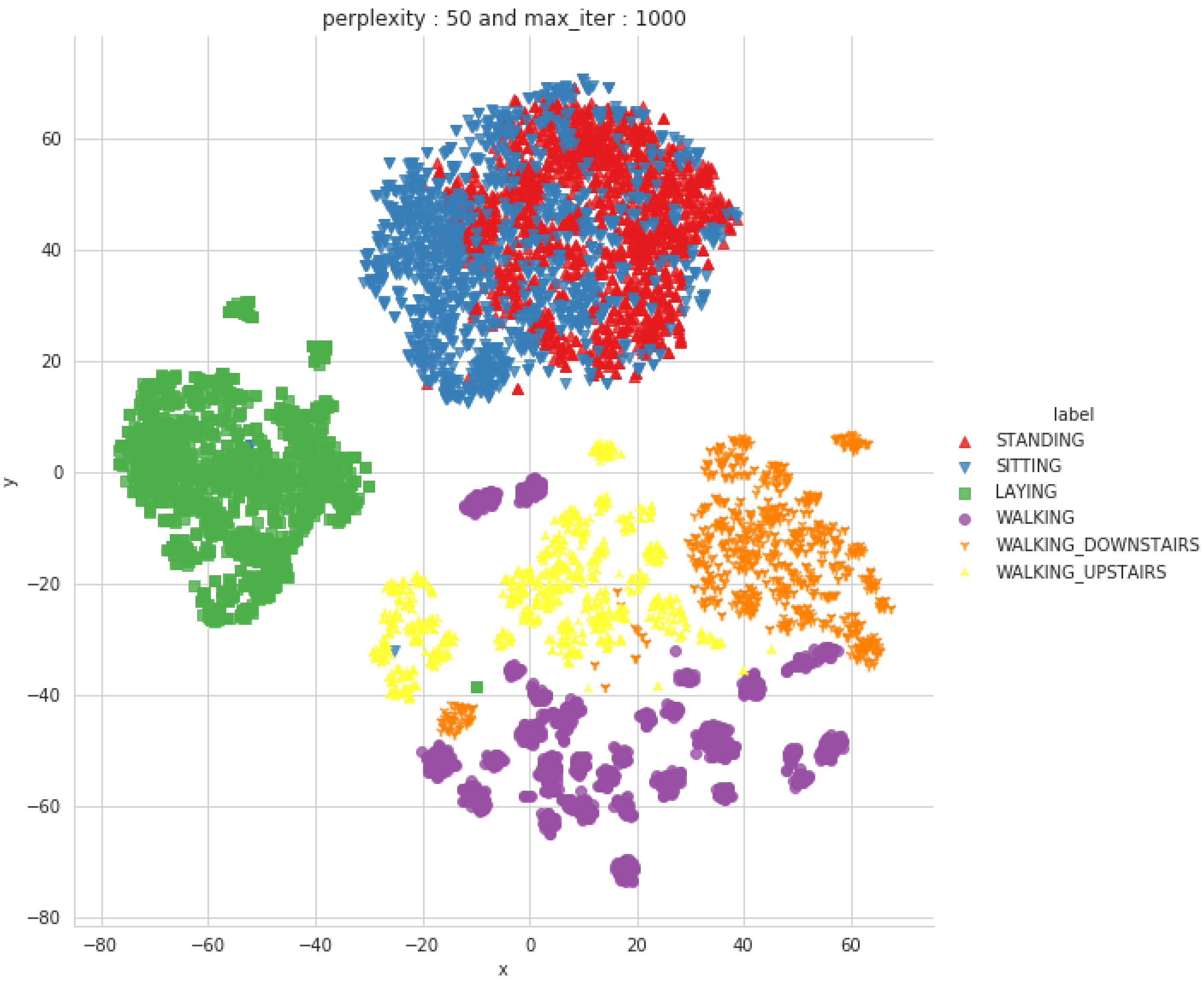

```
Done

performing tsne with perplexity 20 and with 1000 iterations at max
[t-SNE] Computing 61 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.184s...
[t-SNE] Computed neighbors for 7352 samples in 33.179s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.274335
[t-SNE] Computed conditional probabilities in 0.142s
[t-SNE] Iteration 50: error = 97.5793228, gradient norm = 0.0195494 (50 iterations in 8.334s)
[t-SNE] Iteration 100: error = 84.0292892, gradient norm = 0.0071256 (50 iterations in 7.075s)
[t-SNE] Iteration 150: error = 81.9819260, gradient norm = 0.0039321 (50 iterations in 6.712s)
[t-SNE] Iteration 200: error = 81.2209778, gradient norm = 0.0026231 (50 iterations in 6.699s)
[t-SNE] Iteration 250: error = 80.8158112, gradient norm = 0.0018528 (50 iterations in 6.762s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.815811
[t-SNE] Iteration 300: error = 2.6962111, gradient norm = 0.0012949 (50 iterations in 6.471s)
[t-SNE] Iteration 350: error = 2.1630335, gradient norm = 0.0005830 (50 iterations in 5.987s)
[t-SNE] Iteration 400: error = 1.9142398, gradient norm = 0.0003478 (50 iterations in 5.947s)
[t-SNE] Iteration 450: error = 1.7682433, gradient norm = 0.0002490 (50 iterations in 5.986s)
[t-SNE] Iteration 500: error = 1.6742952, gradient norm = 0.0001930 (50 iterations in 6.015s)
[t-SNE] Iteration 550: error = 1.6103836, gradient norm = 0.0001575 (50 iterations in 6.004s)
[t-SNE] Iteration 600: error = 1.5639369, gradient norm = 0.0001382 (50 iterations in 6.003s)
[t-SNE] Iteration 650: error = 1.5287529, gradient norm = 0.0001188 (50 iterations in 6.013s)
[t-SNE] Iteration 700: error = 1.5012516, gradient norm = 0.0001051 (50 iterations in 6.049s)
[t-SNE] Iteration 750: error = 1.4795823, gradient norm = 0.0000959 (50 iterations in 5.998s)
[t-SNE] Iteration 800: error = 1.4621692, gradient norm = 0.0000887 (50 iterations in 6.039s)
[t-SNE] Iteration 850: error = 1.4479774, gradient norm = 0.0000841 (50 iterations in 6.058s)
[t-SNE] Iteration 900: error = 1.4362506, gradient norm = 0.0000768 (50 iterations in 6.078s)
[t-SNE] Iteration 950: error = 1.4260758, gradient norm = 0.0000742 (50 iterations in 6.116s)
[t-SNE] Iteration 1000: error = 1.4174592, gradient norm = 0.0000704 (50 iterations in 6.125s)
[t-SNE] Error after 1000 iterations: 1.417459
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```

```
c:\users\admin\appdata\local\programs\python\python36\lib\site-packages\seaborn\regression.py:546: UserWarning: The `size` paramter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```




```
Done

performing tsne with perplexity 50 and with 1000 iterations at max
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.183s...
[t-SNE] Computed neighbors for 7352 samples in 34.561s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.437672
[t-SNE] Computed conditional probabilities in 0.339s
[t-SNE] Iteration 50: error = 86.7352371, gradient norm = 0.0179428 (50 iterations in 11.872s)
[t-SNE] Iteration 100: error = 75.6326904, gradient norm = 0.0040232 (50 iterations in 10.517s)
[t-SNE] Iteration 150: error = 74.6352463, gradient norm = 0.0024756 (50 iterations in 9.854s)
[t-SNE] Iteration 200: error = 74.2456436, gradient norm = 0.0016277 (50 iterations in 9.865s)
[t-SNE] Iteration 250: error = 74.0648193, gradient norm = 0.0011088 (50 iterations in 9.917s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.064819
[t-SNE] Iteration 300: error = 2.1718478, gradient norm = 0.0012038 (50 iterations in 9.947s)
[t-SNE] Iteration 350: error = 1.7664882, gradient norm = 0.0004879 (50 iterations in 9.804s)
[t-SNE] Iteration 400: error = 1.5953329, gradient norm = 0.0002896 (50 iterations in 9.831s)
[t-SNE] Iteration 450: error = 1.5009161, gradient norm = 0.0001939 (50 iterations in 9.832s)
[t-SNE] Iteration 500: error = 1.4402508, gradient norm = 0.0001424 (50 iterations in 9.824s)
[t-SNE] Iteration 550: error = 1.3984326, gradient norm = 0.0001144 (50 iterations in 9.824s)
[t-SNE] Iteration 600: error = 1.3684371, gradient norm = 0.0000974 (50 iterations in 9.864s)
[t-SNE] Iteration 650: error = 1.3464450, gradient norm = 0.0000854 (50 iterations in 9.856s)
[t-SNE] Iteration 700: error = 1.3304332, gradient norm = 0.0000762 (50 iterations in 9.982s)
[t-SNE] Iteration 750: error = 1.3186313, gradient norm = 0.0000688 (50 iterations in 9.955s)
[t-SNE] Iteration 800: error = 1.3093780, gradient norm = 0.0000650 (50 iterations in 9.890s)
[t-SNE] Iteration 850: error = 1.3019311, gradient norm = 0.0000623 (50 iterations in 9.818s)
[t-SNE] Iteration 900: error = 1.2962193, gradient norm = 0.0000599 (50 iterations in 9.865s)
[t-SNE] Iteration 950: error = 1.2916567, gradient norm = 0.0000568 (50 iterations in 9.472s)
[t-SNE] Iteration 1000: error = 1.2877676, gradient norm = 0.0000549 (50 iterations in 9.422s)
[t-SNE] Error after 1000 iterations: 1.287768
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```

```
c:\users\admin\appdata\local\programs\python\python36\lib\site-packages\seaborn\regression.py:546: Use
rWarning: The `size` paramter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



Done

Observations

1. The above T-sne visualizations are very good and gave a very good 2d-representation of the 564 dimensions vector space.
2. The above t-sne plots are tried over variety of perplexity values and in all these values the visualizations are good to understand.
3. Well globular structures can be seen with a clear separations of the class labels except there is a some overlap in the sitting and standing class.
4. The overlapping of the class-labels such as sitting and standing is persistent with varied perplexity values so the key challenge will be to separate the standing and sitting class labels while the modelling phase.

5. Applying the super-vised machine learning models over HAR

5.1 Obtain the train and test data

In [24]:
```
train = pd.read_csv('UCI_HAR_dataset/csv_files/train.csv')
test = pd.read_csv('UCI_HAR_dataset/csv_files/test.csv')
print(train.shape, test.shape)
```

```
(7352, 564) (2947, 564)
```

In [25]:
```
train.head(3)
```

Out[25]:

	tBodyAccmeanX	tBodyAccmeanY	tBodyAccmeanZ	tBodyAccstdX	tBodyAccstdY	tBodyAccstdZ	tBodyAccmadX	tBodyAcc
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.9
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.9
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.9

3 rows × 564 columns

In [26]:
```
# get X_train and y_train from csv files
X_train = train.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_train = train.ActivityName
```

In [27]:
```
# get X_test and y_test from test csv file
X_test = test.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_test = test.ActivityName
```

In [28]:
```
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 : ((7352, 561),(7352,))
X_test  and y_test  : ((2947, 561),(2947,))
```

5.2 Let's model with our data

Labels that are useful in plotting confusion matrix

In [29]:
```
labels=['LAYING', 'SITTING','STANDING','WALKING','WALKING_DOWNSTAIRS','WALKING_UPSTAIRS']
```

5.2.1Function to plot the confusion matrix

In [30]:
```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
plt.rcParams["font.family"] = 'DejaVu Sans'

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')
```

5.2.2.Generic function to run any model specified

In [31]:
```python
from datetime import datetime
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_time = datetime.now()
    print('training the model..')
    model.fit(X_train, y_train)
    print('Done \n \n')
    train_end_time = datetime.now()
    results['training_time'] =  train_end_time - train_start_time
    print('training_time(HH:MM:SS.ms) - {}\n\n'.format(results['training_time']))


    # predict test data
    print('Predicting test data')
    test_start_time = datetime.now()
    y_pred = model.predict(X_test)
    test_end_time = datetime.now()
    print('Done \n \n')
    results['testing_time'] = test_end_time - test_start_time
    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 matri
x', cmap = cm_cmap)
    plt.show()

    # get classification report
    print('-------------------------')
    print('| Classifiction 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
```

5.2.3. Method to print the gridsearch Attributes

In [32]:
```
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('\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_
))
```

5.3. Logistic Regression with Grid Search

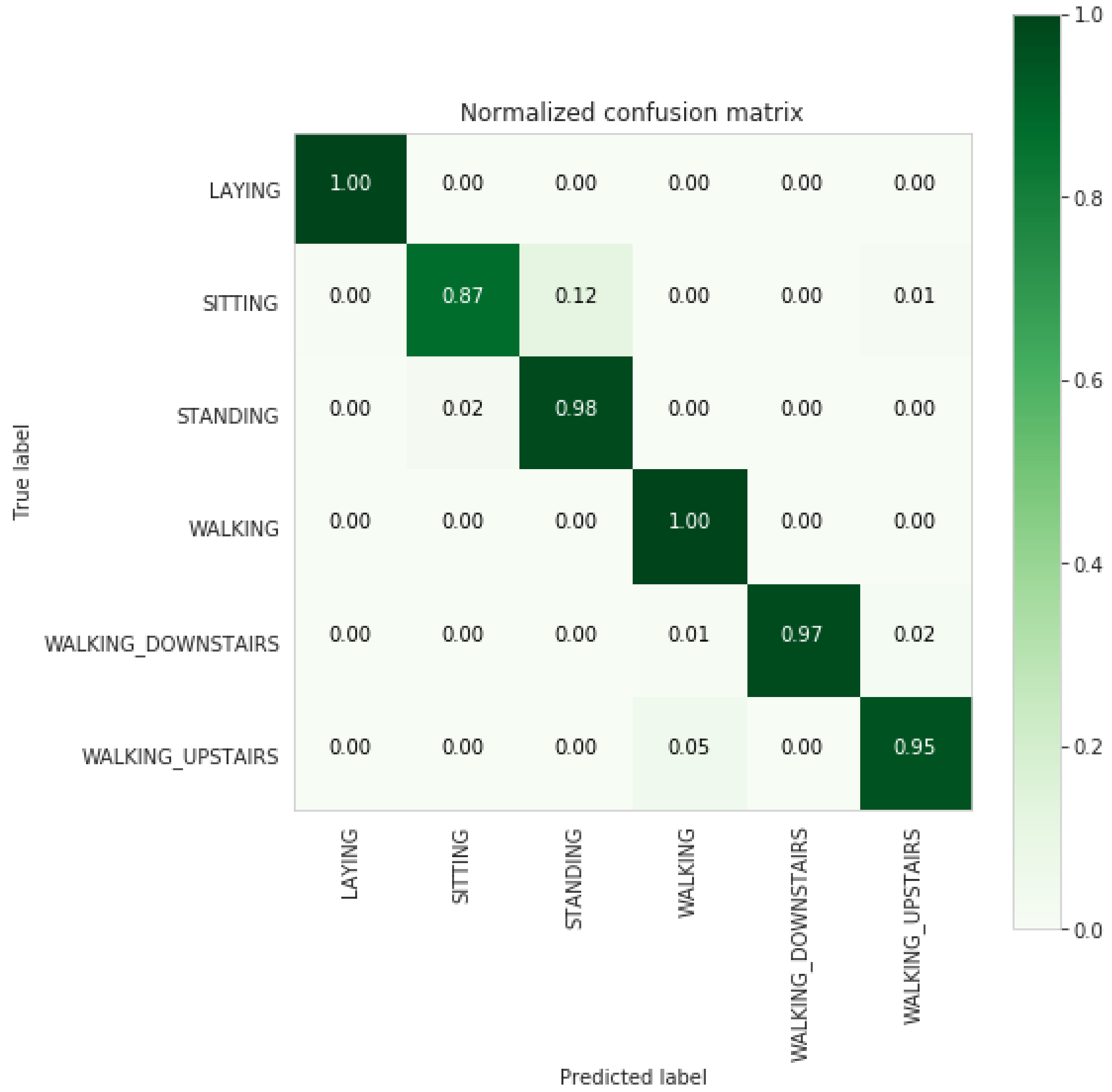
In [33]:
```
from sklearn import linear_model
from sklearn import metrics

from sklearn.model_selection import GridSearchCV
```

In [35]:
```
# start Grid search
parameters = {'C':[0.01, 0.1, 1, 10, 20, 30], 'penalty':['l2','l1']}
log_reg = linear_model.LogisticRegression()
log_reg_grid = GridSearchCV(log_reg, param_grid=parameters, cv=3, verbose=1, n_jobs=-1)
log_reg_grid_results =  perform_model(log_reg_grid, X_train, y_train, X_test, y_test, class_labels=lab
els)
```

```
training the model..
Fitting 3 folds for each of 12 candidates, totalling 36 fits
```

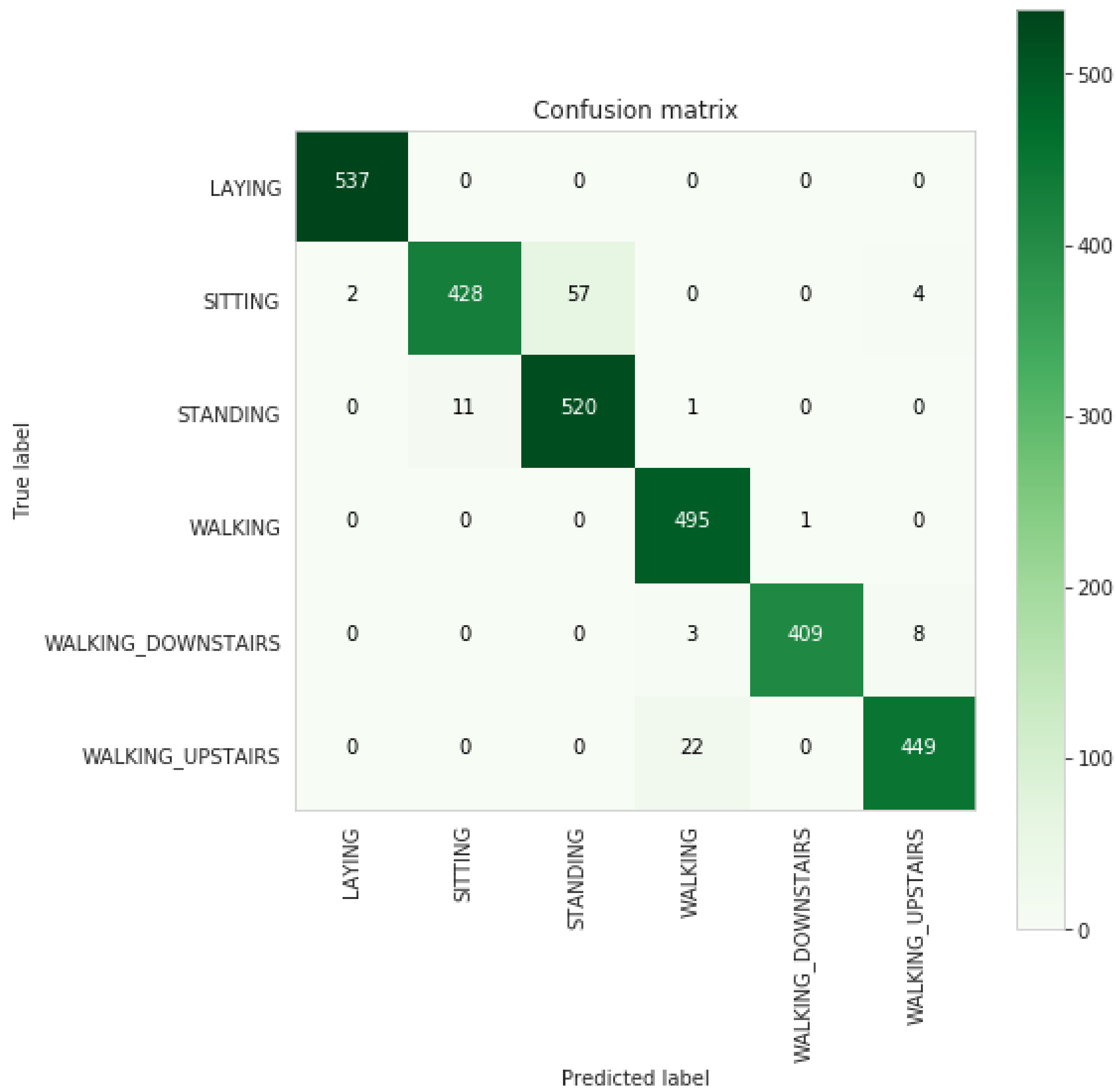
```
[Parallel(n_jobs=-1)]: Done  36 out of  36 | elapsed:   50.3s finished
```

```
Done


training_time(HH:MM:SS.ms) - 0:00:56.617551


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.003962


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

    0.9630132337970818


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

 [[537   0   0   0   0   0]
 [  2 428  57   0   0   4]
 [  0  11 520   1   0   0]
 [  0   0   0 495   1   0]
 [  0   0   0   3 409   8]
 [  0   0   0  22   0 449]]
```




```
------------------------
| Classifiction 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       471

       avg / total       0.96      0.96      0.96      2947
```

In [36]:
```
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 [37]:
```
# 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, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', 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 : 

	0.9458650707290533
```

5.4. Linear SVC with GridSearch

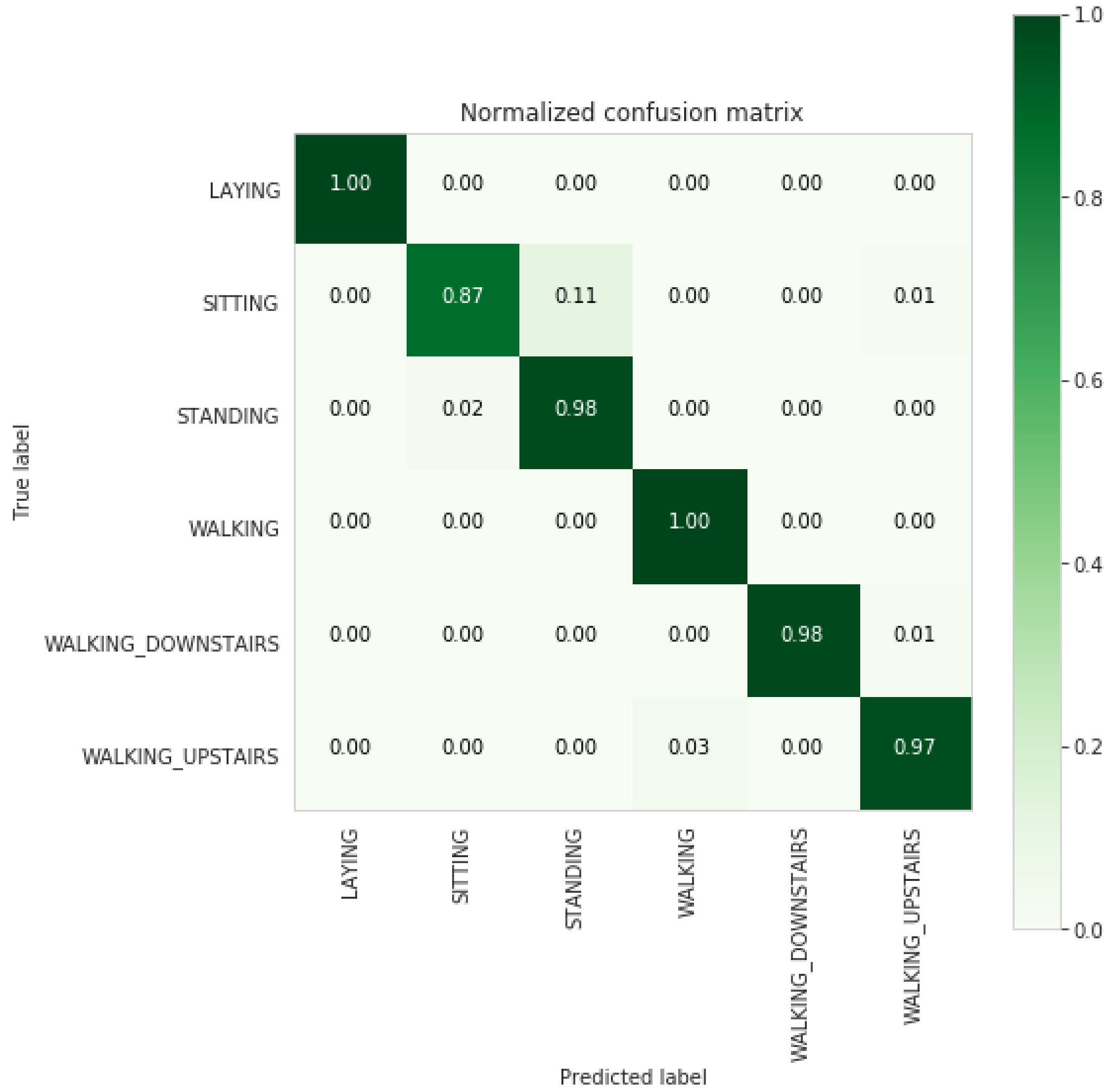
In [38]:
```
from sklearn.svm import LinearSVC
```

In [39]:
```
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)]: Done  18 out of  18 | elapsed:   15.4s finished
```

```
Done 


training_time(HH:MM:SS.ms) - 0:00:19.300331


Predicting test data
Done 


testing time(HH:MM:SS:ms) - 0:00:00.003963


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

    0.9664065151001018


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

 [[537   0   0   0   0   0]
 [  2 429  56   0   0   4]
 [  0  12 519   1   0   0]
 [  0   0   0 496   0   0]
 [  0   0   0   2 412   6]
 [  0   0   0  16   0 455]]
```




```
-------------------------
| Classifiction 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       0.96      1.00      0.98       496
WALKING_DOWNSTAIRS       1.00      0.98      0.99       420
  WALKING_UPSTAIRS       0.98      0.97      0.97       471

       avg / total       0.97      0.97      0.97      2947
```

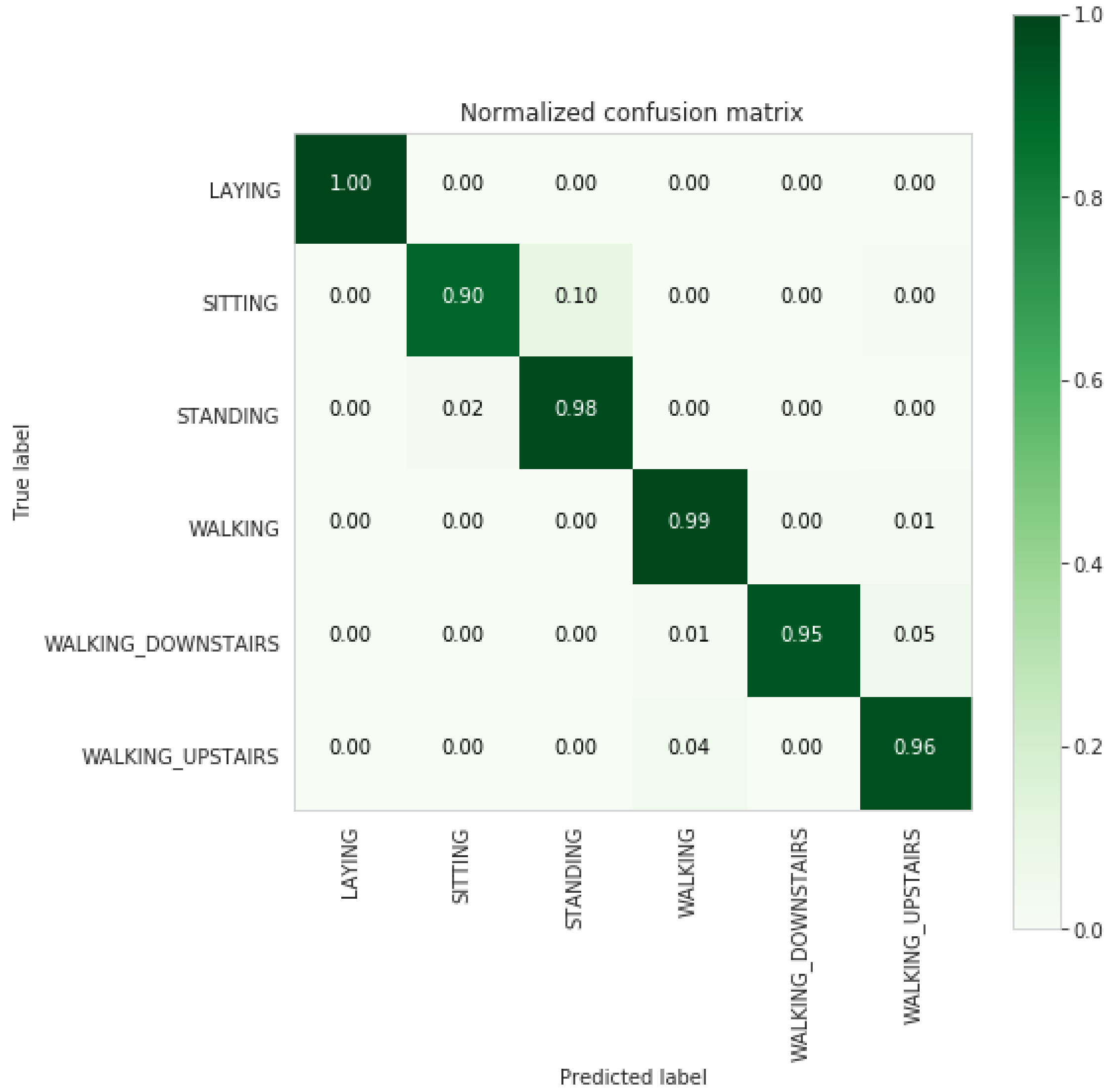
In [40]:
```python
print_grid_search_attributes(lr_svc_grid_results['model'])
```

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

        LinearSVC(C=1, 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': 1}

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.9460010881392819
```

5.5. Kernel SVM with GridSearch

In [41]:
```
from sklearn.svm import SVC
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=labe
ls)
```

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:03:35.902810


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:02.103974


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

    0.9626739056667798


--------------------
| 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   1 453]]
```



```
-------------------------
| Classifiction 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       471

       avg / total       0.96      0.96      0.96      2947
```

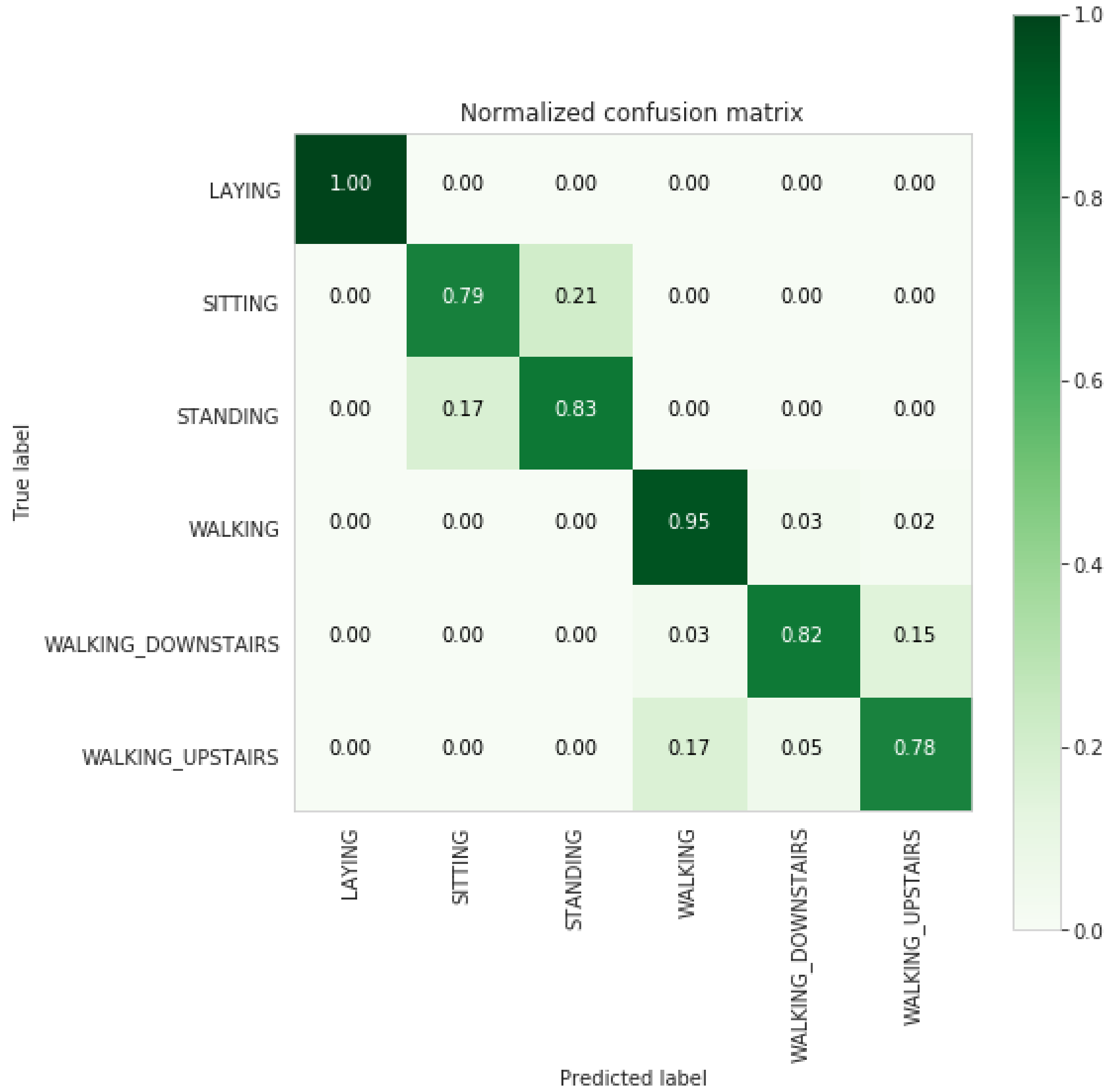
In [42]:
```
print_grid_search_attributes(rbf_svm_grid_results['model'])
```

```
--------------------------
|      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 : 

        0.9440968443960827
```

5.6. Decision Trees with GridSearchCV

In [43]:
```
from sklearn.tree import DecisionTreeClassifier
parameters = {'max_depth':np.arange(3,10,2)}
dt = DecisionTreeClassifier()
dt_grid = GridSearchCV(dt,param_grid=parameters, n_jobs=-1)
dt_grid_results = perform_model(dt_grid, X_train, y_train, X_test, y_test, class_labels=labels)
print_grid_search_attributes(dt_grid_results['model'])
```

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:00:07.845038


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.004990


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

    0.8649474041398032


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

 [[537   0   0   0   0   0]
 [  0 388 103   0   0   0]
 [  0  93 439   0   0   0]
 [  0   0   0 471  17   8]
 [  0   0   0  14 345  61]
 [  0   0   0  78  24 369]]
```



```
-------------------------
| Classifiction 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       0.84      0.95      0.89       496
WALKING_DOWNSTAIRS       0.89      0.82      0.86       420
  WALKING_UPSTAIRS       0.84      0.78      0.81       471

       avg / total       0.87      0.86      0.86      2947

--------------------------
|      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 :

        0.8404515778019587
```

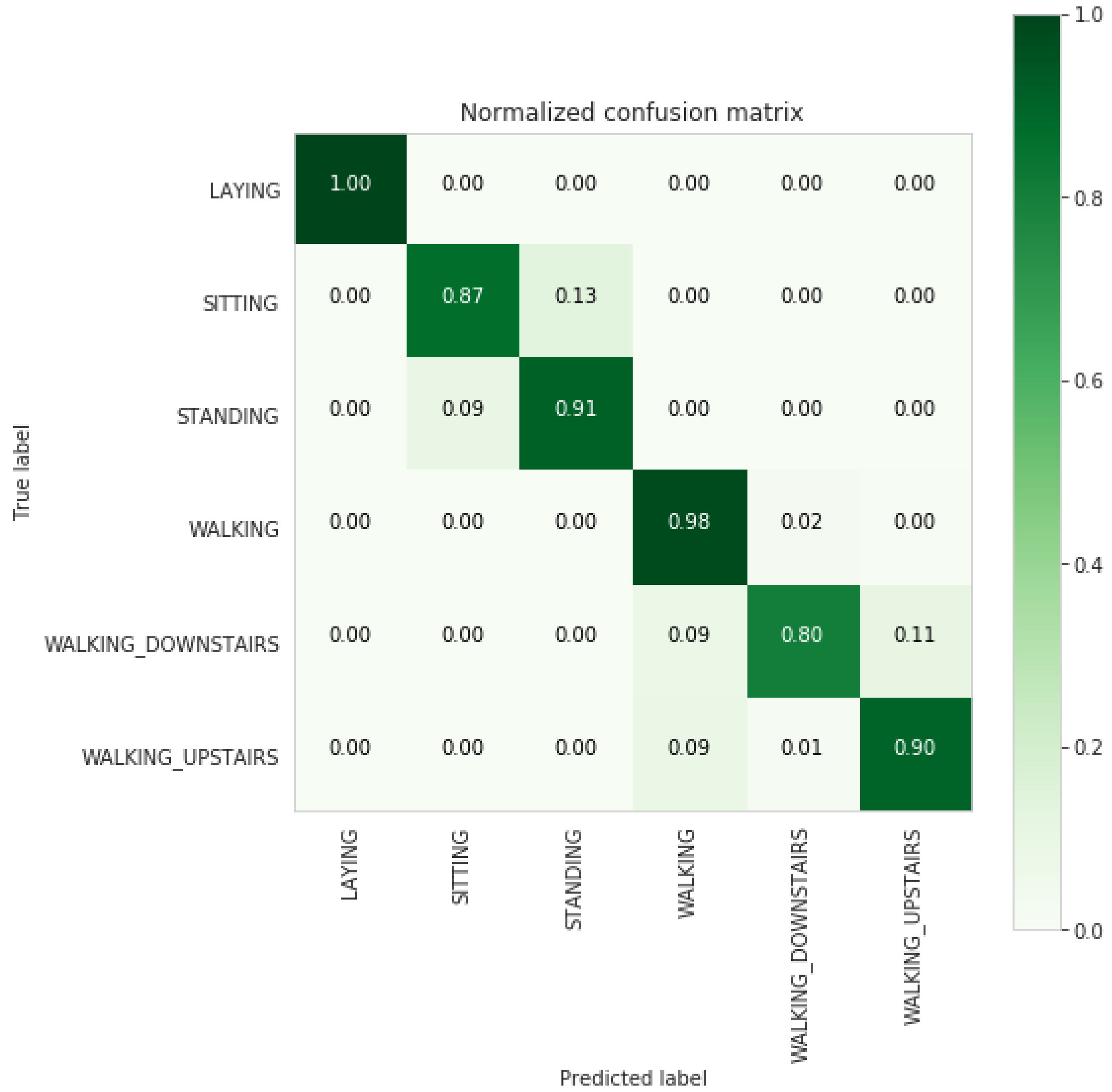
5.7 Random Forest Classifier with GridSearch

In [44]:

```
from sklearn.ensemble import RandomForestClassifier
params = {'n_estimators': np.arange(10,201,20), 'max_depth':np.arange(3,15,2)}
rfc = RandomForestClassifier()
rfc_grid = GridSearchCV(rfc, param_grid=params, n_jobs=-1)
rfc_grid_results = perform_model(rfc_grid, X_train, y_train, X_test, y_test, class_labels=labels)
print_grid_search_attributes(rfc_grid_results['model'])
```

```
c:\users\admin\appdata\local\programs\python\python36\lib\site-packages\sklearn\ensemble\weight_boosti
ng.py:29: DeprecationWarning: numpy.core.umath_tests is an internal NumPy module and should not be imp
orted. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

```
training the model..
Done 
 

training_time(HH:MM:SS.ms) - 0:02:19.537085
 

Predicting test data
Done 
 

testing time(HH:MM:SS:ms) - 0:00:00.049891
 

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

    0.9134713267729895
 

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

 [[537   0   0   0   0   0]
 [  0 427  64   0   0   0]
 [  0  49 483   0   0   0]
 [  0   0   0 484  10   2]
 [  0   0   0  36 337  47]
 [  0   0   0  41   6 424]]
```



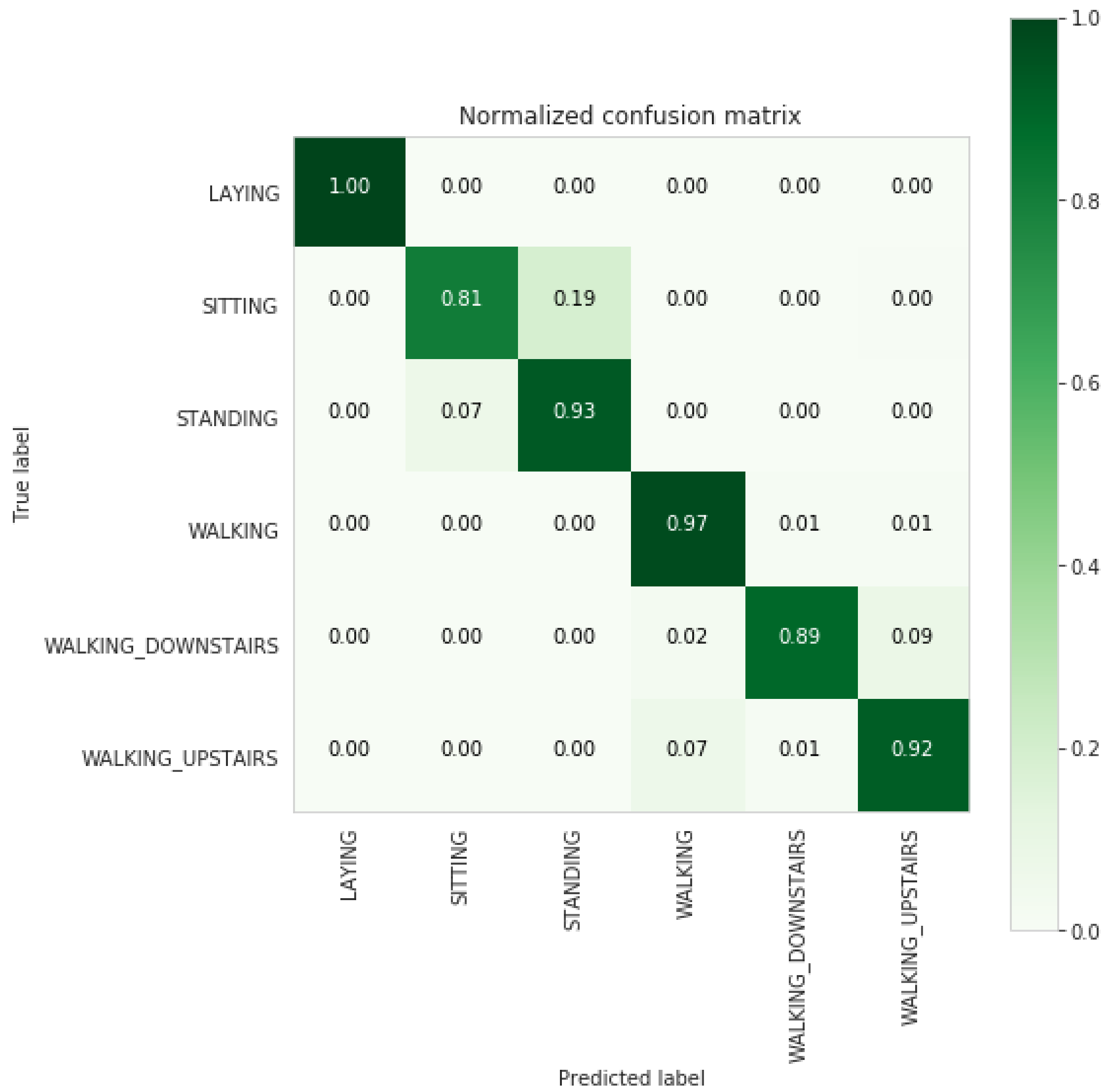
```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.90      0.87      0.88       491
          STANDING       0.88      0.91      0.90       532
           WALKING       0.86      0.98      0.92       496
WALKING_DOWNSTAIRS       0.95      0.80      0.87       420
  WALKING_UPSTAIRS       0.90      0.90      0.90       471

       avg / total       0.92      0.91      0.91      2947

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

        RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=7, 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=170, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)

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

        {'max_depth': 7, 'n_estimators': 170}

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.9181175190424374
```

5.8. Gradient Boosted Decision Trees With GridSearch

In [45]:

```
from sklearn.ensemble import GradientBoostingClassifier
param_grid = {'max_depth': np.arange(5,8,1), \
             'n_estimators':np.arange(130,170,10)}
gbdt = GradientBoostingClassifier()
gbdt_grid = GridSearchCV(gbdt, 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)
print_grid_search_attributes(gbdt_grid_results['model'])
```

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:16:41.633751


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.057247


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

    0.9229725144214456


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

 [[537   0   0   0   0   0]
 [  0 398  91   0   0   2]
 [  0  37 495   0   0   0]
 [  0   0   0 483   7   6]
 [  0   0   0  10 374  36]
 [  0   1   0  31   6 433]]
```



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

            LAYING       1.00      1.00      1.00       537
           SITTING       0.91      0.81      0.86       491
          STANDING       0.84      0.93      0.89       532
           WALKING       0.92      0.97      0.95       496
WALKING_DOWNSTAIRS       0.97      0.89      0.93       420
  WALKING_UPSTAIRS       0.91      0.92      0.91       471

       avg / total       0.92      0.92      0.92      2947

--------------------------
|      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=160,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)

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

        {'max_depth': 5, 'n_estimators': 160}

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.9047878128400435
```

5.9. Comparing all models

In [46]:

```
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 : 96.3%       3.699%
Linear SVC          : 96.64%       3.359%
rbf SVM classifier  : 96.27%      3.733%
DecisionTree        : 86.49%      13.51%
Random Forest       : 91.35%      8.653%
GradientBoosting DT : 91.35%      8.653%
```

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

Conclusion :

- In the real world, domain-knowledge, EDA and feature-engineering matter most, as these type of features adds most value to a model but developing them is pain-staking task to accomplish as it needs good expertise and knowledge of the given problem.
- Among all the supervised machine-learning models the linear models and the kernel SVM models perform exceptionally well as the accuracy values are 96% which is very high as compared to the other models.
- Here the tree-based models are also performing well but not as good as the linear and kernel SVM models.
- The Kernel and the linear SVM separates the sitting and the standing classes quite elegantly with minimal error rates.

6. Implementing the Deep-learning models (LSTM)

6.1 Work-flow of the LSTM model for (HAR)

The overall algorithm has the following workflow:

1. Load the data.
2. Define the hyperparameters.
3. Set up the LSTM model using imperative programming and the hyperparameters.
4. Apply batch-wise training. That is, pick a batch of data, feed it to the model, then, after some iterations, evaluate the model and print the batch loss and the accuracy.
5. Output the chart for the training and test errors.

The above steps can be followed and constructed a pipeline:

6.2 Utility function for plotting the confusion matrix

In [38]:
```
# Activities are the class labels
# It is a 6 class classification
ACTIVITIES = {
    0: 'WALKING',
    1: 'WALKING_UPSTAIRS',
    2: 'WALKING_DOWNSTAIRS',
    3: 'SITTING',
    4: 'STANDING',
    5: 'LAYING',
}

# Utility function to print the confusion matrix
def confusion_matrix(Y_true, Y_pred):
    Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_pred, axis=1)])

    print(metrics.confusion_matrix(Y_true,Y_pred))
    confusion=metrics.confusion_matrix(Y_true,Y_pred)

    plt.figure(figsize=(9,9))
    sns.heatmap(confusion, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
    plt.ylabel('Predicted label');
    plt.xlabel('Actual label');

    plt.show()

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

6.3 Data

```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"
]
```

6.4 Utility functions for reading and loading the data

```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 = []

    for signal in SIGNALS:
        filename = f'UCI_HAR_Dataset/{subset}/Inertial Signals/{signal}_{subset}.txt'
        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))
```

```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 = f'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
    y = _read_csv(filename)[0]

    return pd.get_dummies(y).as_matrix()
```

```python
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
```

6.5 Preparation of the sessions and other parameters for implementing the model

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

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

```python
# Import Keras
from keras import backend as K
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

```
Using TensorFlow backend.
```

In [46]:
```
# Importing libraries
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout
from keras.wrappers.scikit_learn import KerasClassifier
from keras.layers.normalization import BatchNormalization
from keras import regularizers
from keras.regularizers import l1
from keras.regularizers import l2
```

In [51]:
```
# Initializing parameters
epochs = 30
batch_size = 16
n_hidden = 32
```

In [52]:
```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [53]:
```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

In [54]:
```
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
```

6.6. Utility function for plotting the weights and the error plots

In [55]:
```
# Plot train and cross validation loss
def plot_train_cv_loss(trained_model, epochs, colors=['b']):
    fig, ax = plt.subplots(1,1)
    ax.set_xlabel('epoch')
    ax.set_ylabel('Categorical Crossentropy Loss')
    x_axis_values = list(range(1,epochs+1))

    validation_loss = trained_model.history['val_loss']
    train_loss = trained_model.history['loss']

    ax.plot(x_axis_values, validation_loss, 'b', label="Validation Loss")
    ax.plot(x_axis_values, train_loss, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()


# Plot weight distribution using violin plot
def plot_weights(model):
    w_after = model.get_weights()

    o1_w = w_after[0].flatten().reshape(-1,1)
    o2_w = w_after[2].flatten().reshape(-1,1)
    out_w = w_after[4].flatten().reshape(-1,1)


    fig = plt.figure(figsize=(10,7))
    plt.title("Weight matrices after model trained\n")
    plt.subplot(1, 3, 1)
    plt.title("Trained model\n Weights")
    ax = sns.violinplot(y=o1_w,color='b')
    plt.xlabel('Hidden Layer 1')

    plt.subplot(1, 3, 2)
    plt.title("Trained model\n Weights")
    ax = sns.violinplot(y=o2_w, color='r')
    plt.xlabel('Hidden Layer 2 ')

    plt.subplot(1, 3, 3)
    plt.title("Trained model\n Weights")
    ax = sns.violinplot(y=out_w,color='y')
    plt.xlabel('Output Layer ')
    plt.show()
```

6.7 Defining the Architecture of LSTM

In [56]:
```
# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(n_hidden, 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()
```

```
_________________________________________________________________
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
_________________________________________________________________
```

6.7.1 Compiling and training the above LSTM model

In [57]:
```
# Compiling the model
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

In [58]:
```
# Training the model
har_lstm_1=model.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)
```

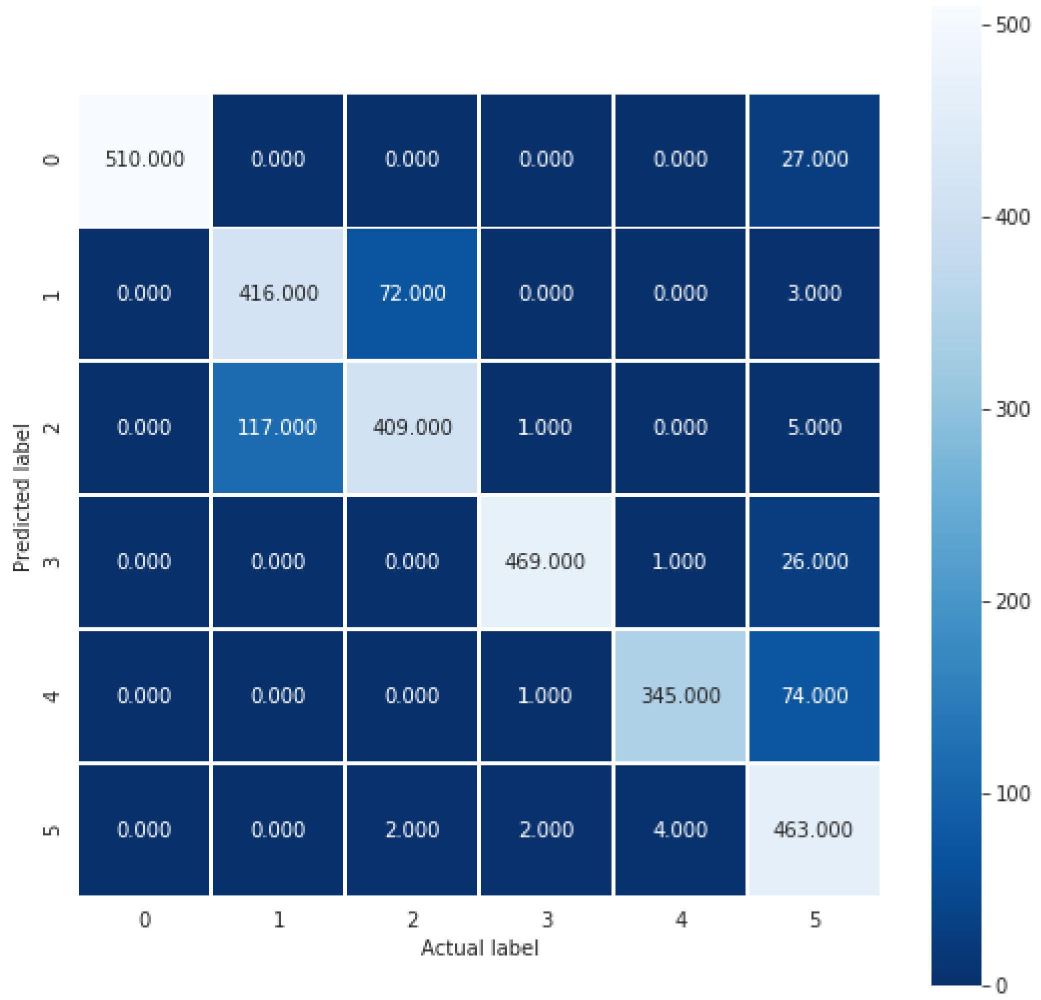
```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 23s 3ms/step - loss: 1.3727 - acc: 0.4106 - val_loss: 1.2
336 - val_acc: 0.4700
Epoch 2/30
7352/7352 [==============================] - 23s 3ms/step - loss: 1.1478 - acc: 0.5038 - val_loss: 1.0
592 - val_acc: 0.5205 acc: 0.492 - ETA: 5s - loss: 1.17
Epoch 3/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.9324 - acc: 0.6026 - val_loss: 0.9
706 - val_acc: 0.5555
Epoch 4/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.9916 - acc: 0.5613 - val_loss: 1.3
933 - val_acc: 0.4058
Epoch 5/30
7352/7352 [==============================] - 23s 3ms/step - loss: 1.2244 - acc: 0.4510 - val_loss: 1.2
554 - val_acc: 0.5239
Epoch 6/30
7352/7352 [==============================] - 24s 3ms/step - loss: 0.9771 - acc: 0.5673 - val_loss: 0.9
204 - val_acc: 0.6125
Epoch 7/30
7352/7352 [==============================] - 23s 3ms/step - loss: 1.0461 - acc: 0.5337 - val_loss: 0.9
159 - val_acc: 0.585766 - ac - ETA: 2s - loss: 1.0766 - acc: 0.52 - ETA: 2
Epoch 8/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.7998 - acc: 0.6114 - val_loss: 0.8
579 - val_acc: 0.6345
Epoch 9/30
7352/7352 [==============================] - 22s 3ms/step - loss: 0.7645 - acc: 0.6419 - val_loss: 0.8
562 - val_acc: 0.6284
Epoch 10/30
7352/7352 [==============================] - 22s 3ms/step - loss: 0.8661 - acc: 0.6230 - val_loss: 0.8
274 - val_acc: 0.6427
Epoch 11/30
7352/7352 [==============================] - 22s 3ms/step - loss: 0.7198 - acc: 0.6706 - val_loss: 0.7
587 - val_acc: 0.6637
Epoch 12/30
7352/7352 [==============================] - 22s 3ms/step - loss: 0.6827 - acc: 0.6970 - val_loss: 0.7
151 - val_acc: 0.7099
Epoch 13/30
7352/7352 [==============================] - 22s 3ms/step - loss: 0.6573 - acc: 0.7186 - val_loss: 0.6
910 - val_acc: 0.7218
Epoch 14/30
7352/7352 [==============================] - 22s 3ms/step - loss: 0.6195 - acc: 0.7454 - val_loss: 1.0
719 - val_acc: 0.6770
Epoch 15/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.5836 - acc: 0.7701 - val_loss: 0.6
398 - val_acc: 0.777468 -
Epoch 16/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.4823 - acc: 0.8264 - val_loss: 0.4
936 - val_acc: 0.8286
Epoch 17/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.4221 - acc: 0.8638 - val_loss: 0.4
761 - val_acc: 0.8476loss
Epoch 18/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.3791 - acc: 0.8856 - val_loss: 0.3
775 - val_acc: 0.8663
Epoch 19/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.3080 - acc: 0.9048 - val_loss: 0.3
957 - val_acc: 0.8721
Epoch 20/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.3101 - acc: 0.8999 - val_loss: 0.5
482 - val_acc: 0.7872
Epoch 21/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.3055 - acc: 0.9076 - val_loss: 0.3
420 - val_acc: 0.8962
Epoch 22/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.3206 - acc: 0.8921 - val_loss: 0.3
685 - val_acc: 0.8870
Epoch 23/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.2401 - acc: 0.9263 - val_loss: 0.3
738 - val_acc: 0.8880
Epoch 24/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.2060 - acc: 0.9324 - val_loss: 0.3
426 - val_acc: 0.9043
Epoch 25/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.1975 - acc: 0.9320 - val_loss: 0.4
369 - val_acc: 0.8714s - loss: 0
Epoch 26/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.2208 - acc: 0.9290 - val_loss: 0.4
025 - val_acc: 0.8979
Epoch 27/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.2527 - acc: 0.9230 - val_loss: 0.3
649 - val_acc: 0.8894
Epoch 28/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.2417 - acc: 0.9159 - val_loss: 0.3
411 - val_acc: 0.8962
Epoch 29/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.2019 - acc: 0.9324 - val_loss: 0.3
```

```
027 - val_acc: 0.9026
Epoch 30/30
7352/7352 [==============================] - 23s 3ms/step - loss: 0.2147 - acc: 0.9317 - val_loss: 0.3
555 - val_acc: 0.8863: 2s -
```

6.7.2. Plotting the confusion and the error plots of the above model

In [62]:
```python
# Confusion Matrix
print(confusion_matrix(Y_test, model.predict(X_test)))
```

```
[[510   0   0   0   0  27]
 [  0 416  72   0   0   3]
 [  0 117 409   1   0   5]
 [  0   0   0 469   1  26]
 [  0   0   0   1 345  74]
 [  0   0   2   2   4 463]]
```

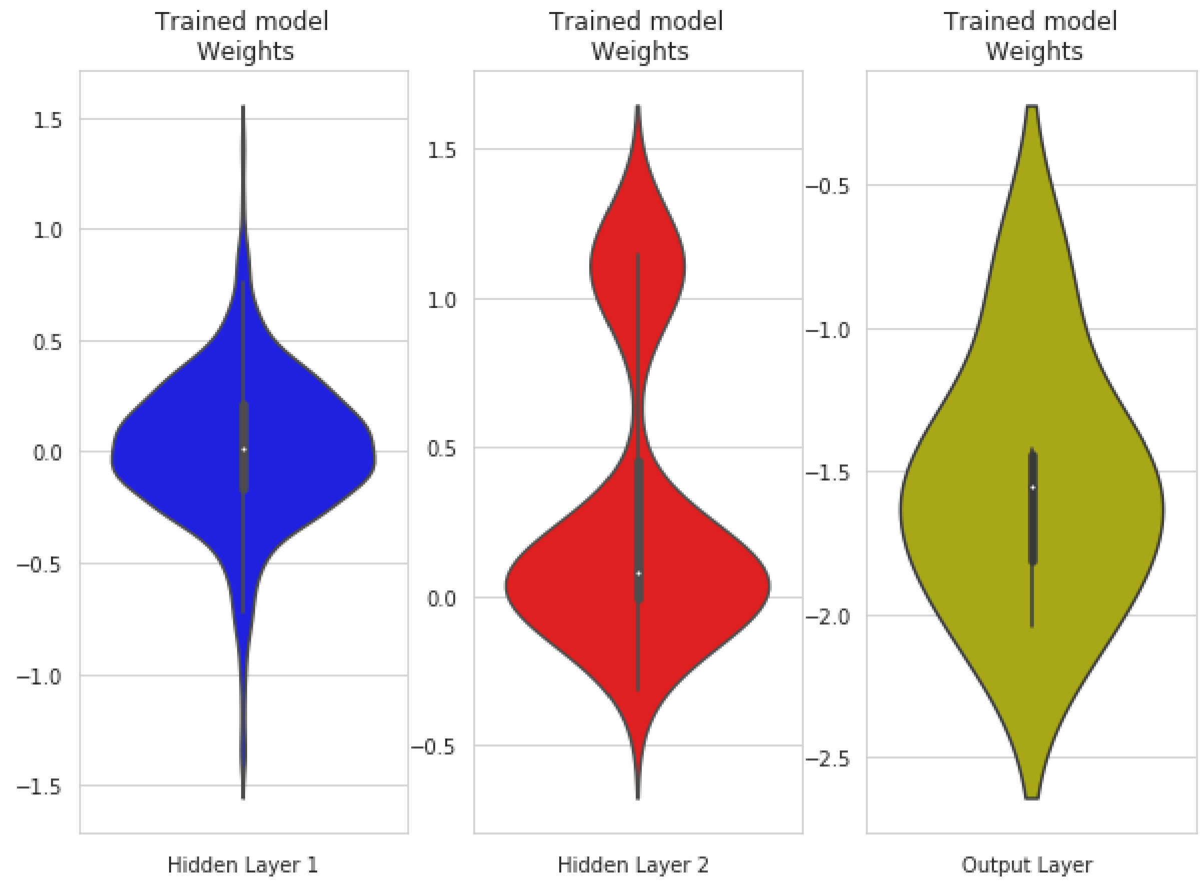


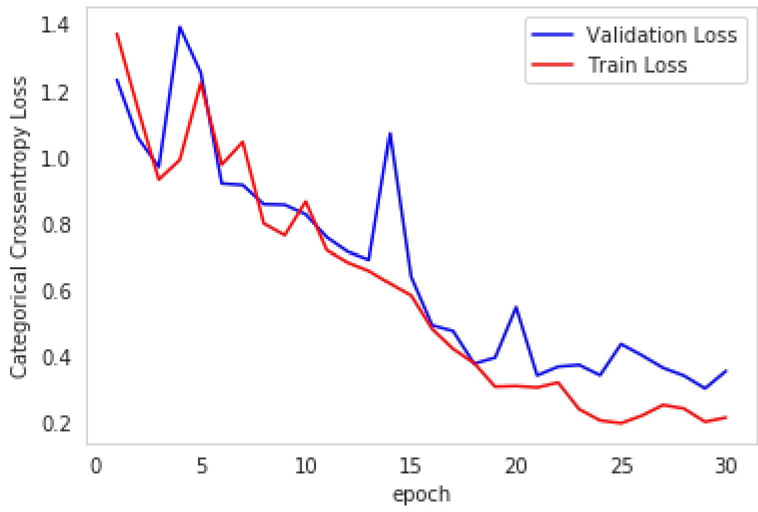

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0         0        0                   0
SITTING                  0      416        72        0                   0
STANDING                 0      117       409        1                   0
WALKING                  0        0         0      469                   1
WALKING_DOWNSTAIRS       0        0         0        1                 345
WALKING_UPSTAIRS         0        0         2        2                   4

Pred                WALKING_UPSTAIRS
True
LAYING                            27
SITTING                            3
STANDING                           5
WALKING                           26
WALKING_DOWNSTAIRS                74
WALKING_UPSTAIRS                 463
```

In [60]:
```python
score = model.evaluate(X_test, Y_test)
print(score)
```

```
2947/2947 [==============================] - 1s 283us/step
[0.3555445054686313, 0.8863250763488293]
```

In [61]:
```python
# Plot weight distribution using violin plot
import warnings
warnings.filterwarnings(action='ignore')
plot_weights(model)
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(har_lstm_1,epochs)
```





Observations:

- Since the data has a sequential behaviour so LSTM model is implemented as it works very well on sequential data.
- The above LSTM model is a single layered model with raw input vectors as inputs to the LSTM model.
- Even if the LSTM model is not using a human engineered features as input the accuracy is very impressive which is around 93.24% at the end of the 30th epoch.
- But still the model is overfitting slightly as the loss is slightly high as compared the previous supervised machine learning models.
- By tuning the hyper-parameter values the performance of the LSTM model can be easily improved so I have tried the next thing as tuning the hyperparameters.

6.7.3. Hyperparameter-tuning the above single-layered LSTM model

Utility function for building the LSTM model

In [63]:
```python
Input=(timesteps, input_dim)
#Implementing the elastic-net regularization
reg=regularizers.l1_l2(l1=0.01, l2=0.01)

def lstm_model(neurons,Dp,Iter):

# Initiliazing the sequential model
    Model = Sequential()
# Configuring the parameters
    for i in range(Iter):
        Model.add(LSTM(neurons, input_shape=(timesteps, input_dim),bias_regularizer=reg))
# Adding a dropout layer
        Model.add(Dropout(Dp))
# Adding a dense output layer with sigmoid activation
    Model.add(Dense(n_classes, activation='sigmoid'))
    Model.summary()
# Compiling the model
    Model.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['accuracy'])
    return Model

batch_size = 192
def train(X_t,train_y,X_te, test_y,batch_size,epoch,model):
    train=model.fit(X_t, train_y, epochs = epoch, batch_size=batch_size, verbose = 2,validation_data=(
X_te, test_y))
    return train
```

6.7.3.1. Taking the number of neurons as 48 and drop-out value as 0.6

In [114]:
```python
model_1=lstm_model(48,0.6,1)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_27 (LSTM)               (None, 48)                11136     
_________________________________________________________________
dropout_24 (Dropout)         (None, 48)                0         
_________________________________________________________________
dense_24 (Dense)             (None, 6)                 294       
=================================================================
Total params: 11,430
Trainable params: 11,430
Non-trainable params: 0
_________________________________________________________________
```

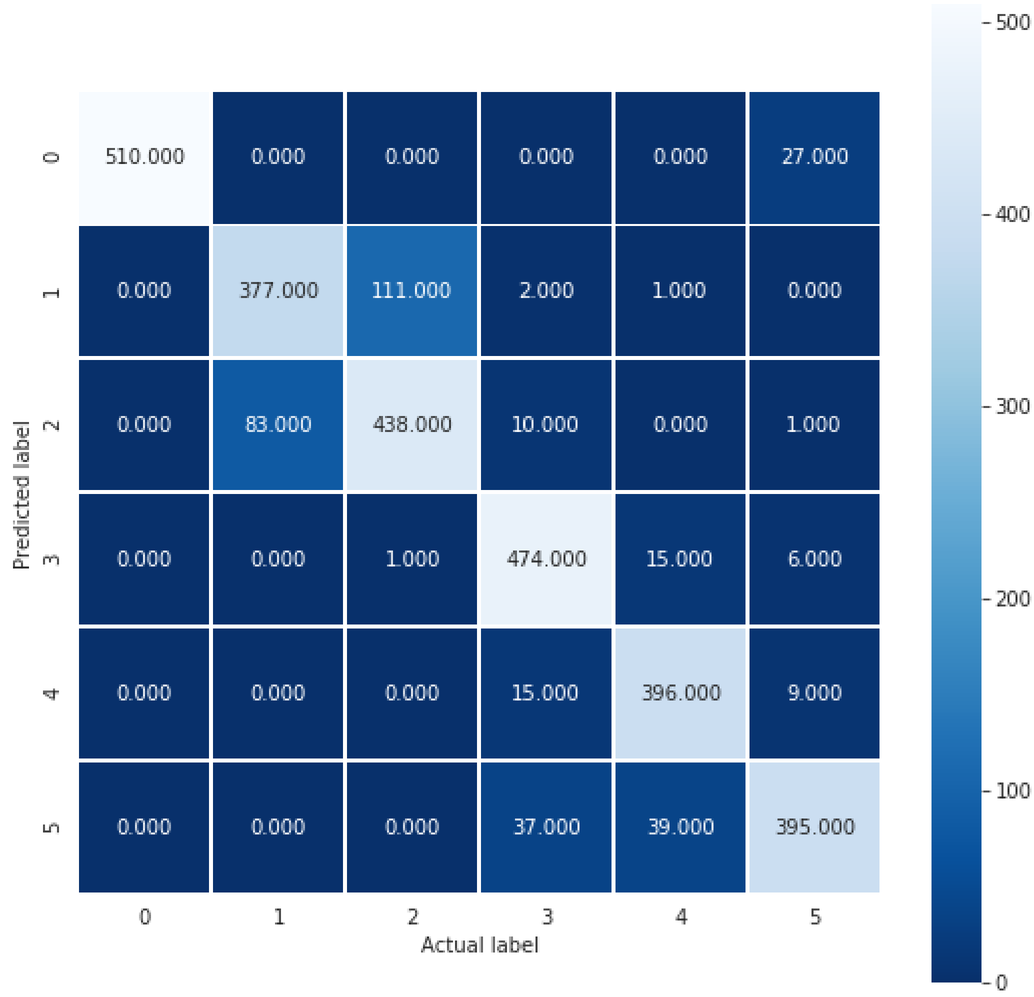
In [116]:
```python
train_1=train(X_train,Y_train,X_test, Y_test,64,30,model_1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 9s - loss: 1.2605 - acc: 0.5837 - val_loss: 1.2212 - val_acc: 0.5803
Epoch 2/30
 - 9s - loss: 1.1417 - acc: 0.6062 - val_loss: 1.1029 - val_acc: 0.6016
Epoch 3/30
 - 9s - loss: 1.0127 - acc: 0.6415 - val_loss: 1.1328 - val_acc: 0.5701
Epoch 4/30
 - 9s - loss: 0.9190 - acc: 0.6470 - val_loss: 0.9899 - val_acc: 0.6250
Epoch 5/30
 - 9s - loss: 0.8835 - acc: 0.6453 - val_loss: 0.9660 - val_acc: 0.6200
Epoch 6/30
 - 9s - loss: 0.8242 - acc: 0.6495 - val_loss: 1.0077 - val_acc: 0.6108
Epoch 7/30
 - 9s - loss: 0.7943 - acc: 0.6527 - val_loss: 0.8611 - val_acc: 0.6233
Epoch 8/30
 - 9s - loss: 0.8226 - acc: 0.6347 - val_loss: 1.0447 - val_acc: 0.6010
Epoch 9/30
 - 9s - loss: 0.7511 - acc: 0.6551 - val_loss: 0.9346 - val_acc: 0.6020
Epoch 10/30
 - 9s - loss: 0.7286 - acc: 0.6640 - val_loss: 0.8350 - val_acc: 0.6335
Epoch 11/30
 - 9s - loss: 0.6726 - acc: 0.6771 - val_loss: 0.8427 - val_acc: 0.6203
Epoch 12/30
 - 9s - loss: 0.6596 - acc: 0.6745 - val_loss: 0.8753 - val_acc: 0.6227
Epoch 13/30
 - 9s - loss: 0.6328 - acc: 0.6884 - val_loss: 1.0240 - val_acc: 0.6237
Epoch 14/30
 - 9s - loss: 0.6136 - acc: 0.7085 - val_loss: 0.8603 - val_acc: 0.6770
Epoch 15/30
 - 9s - loss: 0.6055 - acc: 0.7265 - val_loss: 1.0111 - val_acc: 0.7099
Epoch 16/30
 - 9s - loss: 0.5910 - acc: 0.7330 - val_loss: 0.7940 - val_acc: 0.7604
Epoch 17/30
 - 9s - loss: 0.5814 - acc: 0.7647 - val_loss: 0.8031 - val_acc: 0.7357
Epoch 18/30
 - 9s - loss: 0.5587 - acc: 0.7817 - val_loss: 0.8064 - val_acc: 0.7679
Epoch 19/30
 - 9s - loss: 0.5251 - acc: 0.8086 - val_loss: 0.7549 - val_acc: 0.7849
Epoch 20/30
 - 9s - loss: 0.8191 - acc: 0.7032 - val_loss: 0.9680 - val_acc: 0.6308
Epoch 21/30
 - 9s - loss: 1.3359 - acc: 0.5944 - val_loss: 0.8770 - val_acc: 0.7397
Epoch 22/30
 - 9s - loss: 1.1824 - acc: 0.6153 - val_loss: 0.7732 - val_acc: 0.7703
Epoch 23/30
 - 9s - loss: 0.6353 - acc: 0.7734 - val_loss: 0.6883 - val_acc: 0.7978
Epoch 24/30
 - 9s - loss: 0.5674 - acc: 0.8146 - val_loss: 0.6714 - val_acc: 0.7964
Epoch 25/30
 - 9s - loss: 0.4860 - acc: 0.8384 - val_loss: 0.6140 - val_acc: 0.8344
Epoch 26/30
 - 9s - loss: 0.4433 - acc: 0.8592 - val_loss: 0.8536 - val_acc: 0.7730
Epoch 27/30
 - 9s - loss: 0.3871 - acc: 0.8836 - val_loss: 0.5439 - val_acc: 0.8575
Epoch 28/30
 - 9s - loss: 0.3360 - acc: 0.8992 - val_loss: 0.5868 - val_acc: 0.8622
Epoch 29/30
 - 9s - loss: 0.3266 - acc: 0.9040 - val_loss: 0.6545 - val_acc: 0.8510
Epoch 30/30
 - 9s - loss: 0.3004 - acc: 0.9153 - val_loss: 0.5127 - val_acc: 0.8789
```

In [175]:

```
# Confusion Matrix
print(confusion_matrix(Y_test, model_1.predict(X_test)))
```

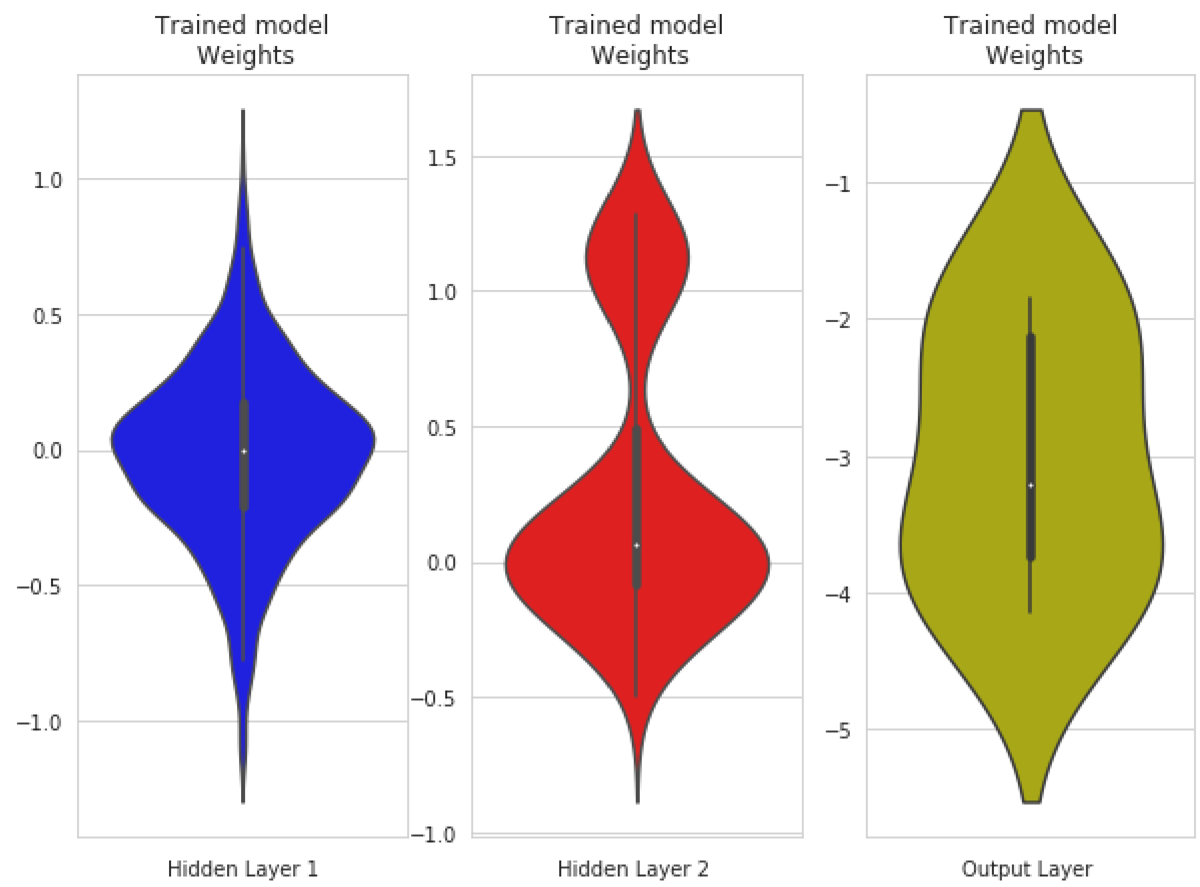
```
[[510   0   0   0   0  27]
 [  0 377 111   2   1   0]
 [  0  83 438  10   0   1]
 [  0   0   1 474  15   6]
 [  0   0   0  15 396   9]
 [  0   0   0  37  39 395]]
```

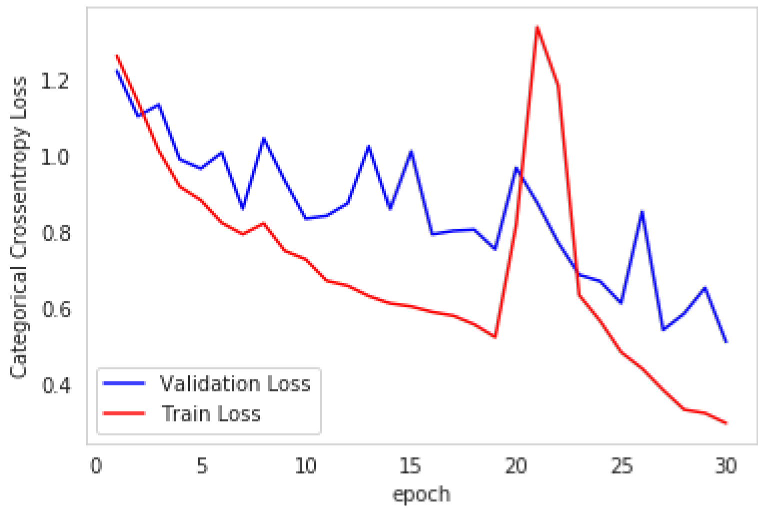



```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0         0        0                   0
SITTING                  0      377       111        2                   1
STANDING                 0       83       438       10                   0
WALKING                  0        0         1      474                  15
WALKING_DOWNSTAIRS       0        0         0       15                 396
WALKING_UPSTAIRS         0        0         0       37                  39

Pred                WALKING_UPSTAIRS
True
LAYING                            27
SITTING                            0
STANDING                           1
WALKING                            6
WALKING_DOWNSTAIRS                 9
WALKING_UPSTAIRS                 395
```

In [161]:

```
# Plot weight distribution using violin plot
import warnings
warnings.filterwarnings(action='ignore')
plot_weights(model)
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(train_1,epochs)
```





- Here by taking the number of neurons as 48 and dropout value as 0.6 the model is slightly overfitting as compared to the previous model the accuracy value is also decreased gradually by .
- So the loss has increased as compared to the previous model so by increasing the drop-out value the performance may increase.

6.7.3.2. Taking the number of neurons as 48 and drop-out value as 0.75

In [64]:
```
model_2=lstm_model(48,0.75,1)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_2 (LSTM)                (None, 48)                11136     
_________________________________________________________________
dropout_2 (Dropout)          (None, 48)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 294       
=================================================================
Total params: 11,430
Trainable params: 11,430
Non-trainable params: 0
_________________________________________________________________
```

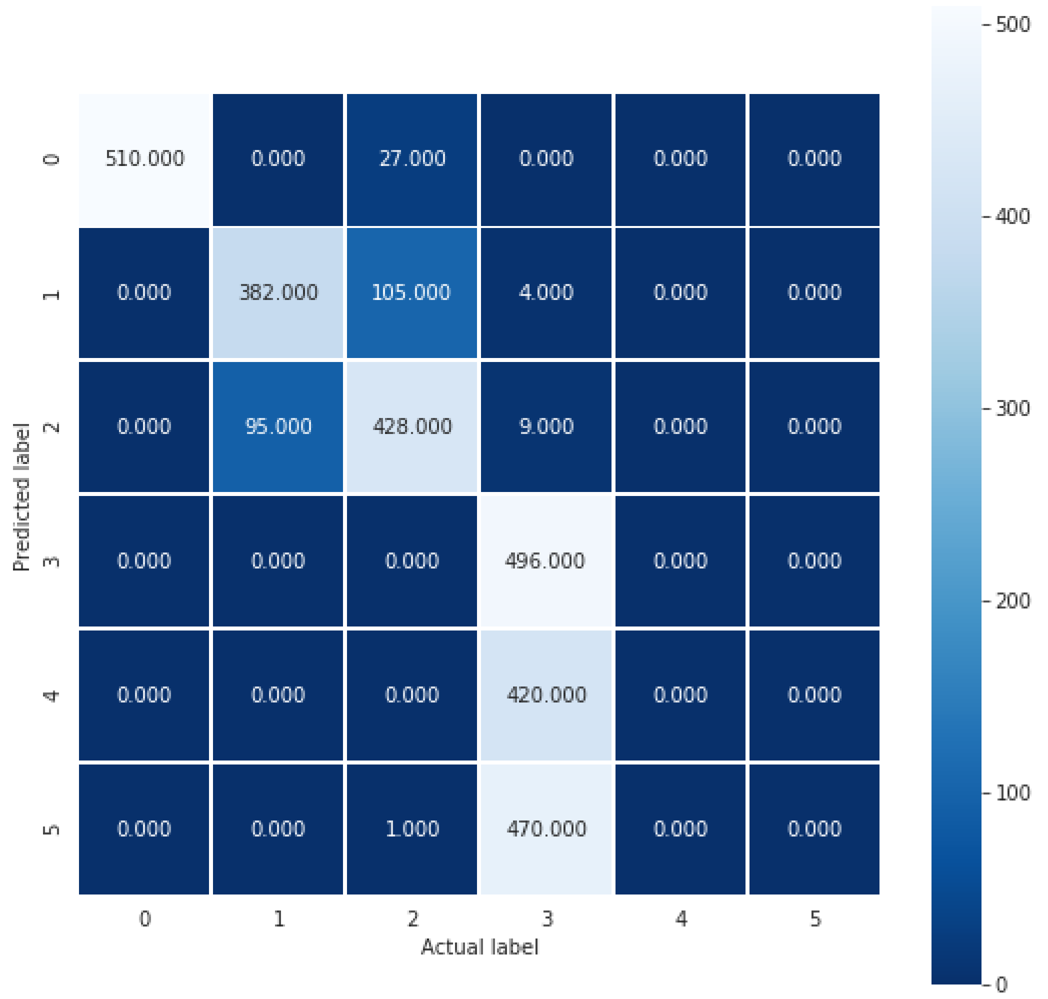
In [65]:
```
train_2=train(X_train,Y_train,X_test, Y_test,64,30,model_2)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 10s - loss: 2.3373 - acc: 0.3583 - val_loss: 2.2067 - val_acc: 0.3427
Epoch 2/30
 - 9s - loss: 2.0600 - acc: 0.3973 - val_loss: 1.9779 - val_acc: 0.4333
Epoch 3/30
 - 9s - loss: 1.8203 - acc: 0.4690 - val_loss: 1.7895 - val_acc: 0.4649
Epoch 4/30
 - 9s - loss: 1.6563 - acc: 0.4985 - val_loss: 1.5844 - val_acc: 0.4669
Epoch 5/30
 - 9s - loss: 1.4972 - acc: 0.5137 - val_loss: 1.4218 - val_acc: 0.5222
Epoch 6/30
 - 9s - loss: 1.3862 - acc: 0.5107 - val_loss: 1.4990 - val_acc: 0.4951
Epoch 7/30
 - 9s - loss: 1.3144 - acc: 0.5107 - val_loss: 1.2146 - val_acc: 0.5551
Epoch 8/30
 - 9s - loss: 1.1493 - acc: 0.5598 - val_loss: 1.1269 - val_acc: 0.5830
Epoch 9/30
 - 9s - loss: 1.1363 - acc: 0.5437 - val_loss: 1.2155 - val_acc: 0.5046
Epoch 10/30
 - 9s - loss: 1.0826 - acc: 0.5341 - val_loss: 1.2568 - val_acc: 0.4113
Epoch 11/30
 - 9s - loss: 0.9562 - acc: 0.5876 - val_loss: 1.2208 - val_acc: 0.4574
Epoch 12/30
 - 9s - loss: 0.8871 - acc: 0.6153 - val_loss: 0.8769 - val_acc: 0.5911
Epoch 13/30
 - 9s - loss: 0.8694 - acc: 0.6053 - val_loss: 1.1334 - val_acc: 0.5453
Epoch 14/30
 - 9s - loss: 0.8317 - acc: 0.6147 - val_loss: 0.9722 - val_acc: 0.5752
Epoch 15/30
 - 9s - loss: 0.8261 - acc: 0.6167 - val_loss: 0.8579 - val_acc: 0.6050
Epoch 16/30
 - 9s - loss: 0.7650 - acc: 0.6337 - val_loss: 0.8567 - val_acc: 0.6026
Epoch 17/30
 - 9s - loss: 0.7617 - acc: 0.6391 - val_loss: 0.8106 - val_acc: 0.6088
Epoch 18/30
 - 9s - loss: 0.7478 - acc: 0.6357 - val_loss: 0.7867 - val_acc: 0.6138
Epoch 19/30
 - 9s - loss: 0.7596 - acc: 0.6345 - val_loss: 0.8010 - val_acc: 0.6149
Epoch 20/30
 - 9s - loss: 0.7433 - acc: 0.6413 - val_loss: 0.8642 - val_acc: 0.6132
Epoch 21/30
 - 9s - loss: 0.7247 - acc: 0.6409 - val_loss: 0.7855 - val_acc: 0.6155
Epoch 22/30
 - 9s - loss: 0.7073 - acc: 0.6480 - val_loss: 0.7886 - val_acc: 0.6094
Epoch 23/30
 - 9s - loss: 0.6938 - acc: 0.6455 - val_loss: 0.7929 - val_acc: 0.6125
Epoch 24/30
 - 9s - loss: 0.6933 - acc: 0.6517 - val_loss: 0.9155 - val_acc: 0.6023
Epoch 25/30
 - 9s - loss: 0.7193 - acc: 0.6430 - val_loss: 0.9105 - val_acc: 0.5989
Epoch 26/30
 - 9s - loss: 0.7289 - acc: 0.6484 - val_loss: 0.8843 - val_acc: 0.6213
Epoch 27/30
 - 9s - loss: 0.7061 - acc: 0.6485 - val_loss: 0.8714 - val_acc: 0.6183
Epoch 28/30
 - 9s - loss: 0.6803 - acc: 0.6600 - val_loss: 0.8456 - val_acc: 0.6216
Epoch 29/30
 - 9s - loss: 0.6911 - acc: 0.6571 - val_loss: 0.8222 - val_acc: 0.6166
Epoch 30/30
 - 9s - loss: 0.6734 - acc: 0.6620 - val_loss: 0.8392 - val_acc: 0.6162
```

In [66]:

```python
# Confusion Matrix
print(confusion_matrix(Y_test, model_2.predict(X_test)))
```

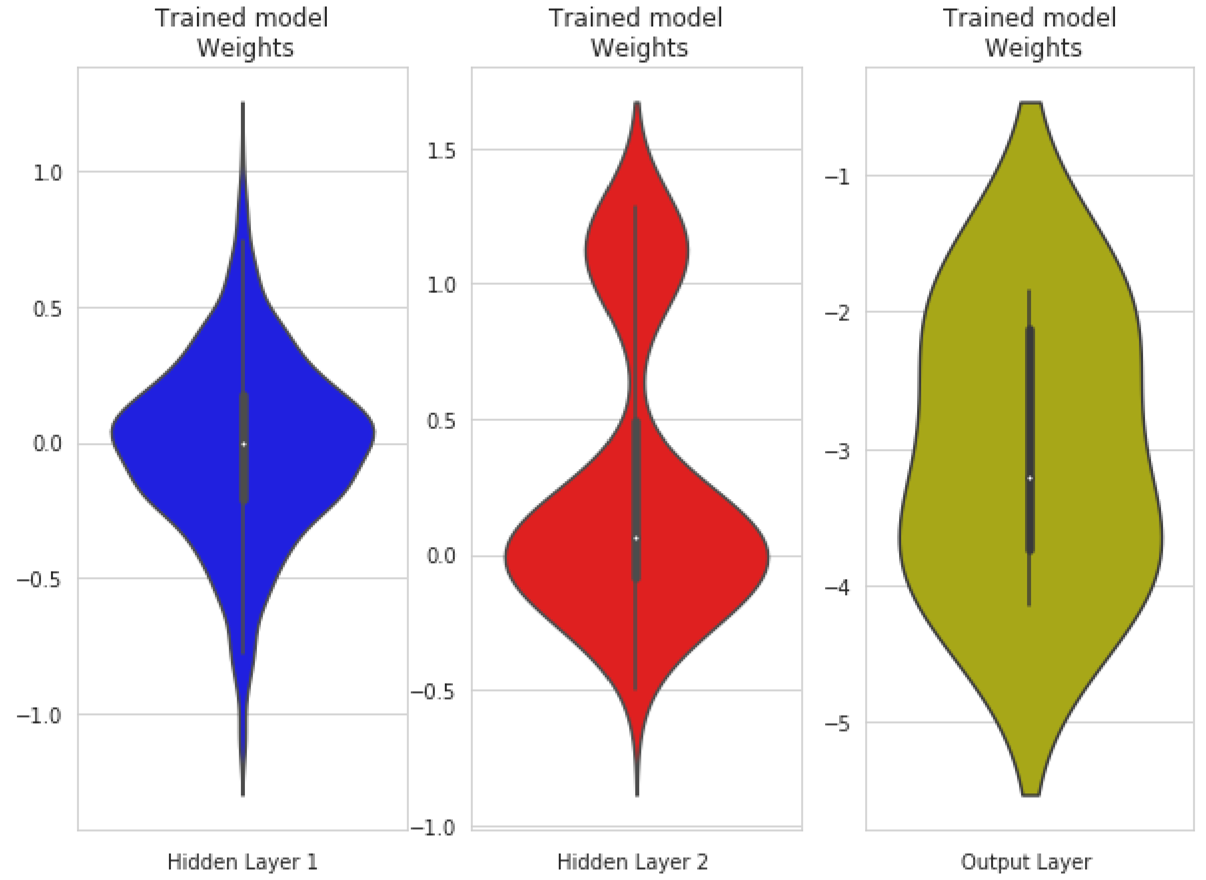
```
[[510   0  27   0   0   0]
 [  0 382 105   4   0   0]
 [  0  95 428   9   0   0]
 [  0   0   0 496   0   0]
 [  0   0   0 420   0   0]
 [  0   0   1 470   0   0]]
```

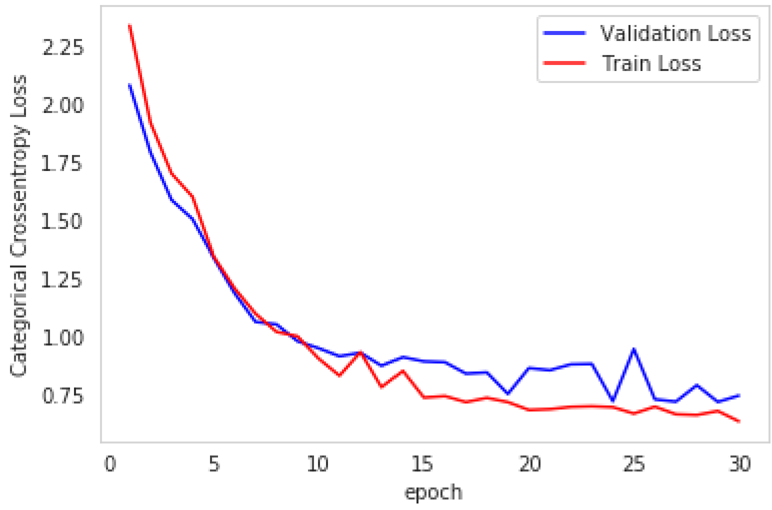



```
Pred                LAYING  SITTING  STANDING  WALKING
True
LAYING                 510        0        27        0
SITTING                  0      382       105        4
STANDING                 0       95       428        9
WALKING                  0        0         0      496
WALKING_DOWNSTAIRS       0        0         0      420
WALKING_UPSTAIRS         0        0         1      470
```

In [162]:

```
# Plot weight distribution using violin plot
import warnings
warnings.filterwarnings(action='ignore')
plot_weights(model)
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(train_2,epochs)
```





- The LSTM model with 48 neurons and 0.75 drop-out is not overfitting but the final accuracy after 30 epochs is very less as compared to previous model.
- This is happenned due to the high drop-out rate and the model is not able to learn new features due to the sparsity of neurons.
- The model is very much confused between the sitting and standing classes and totaly mis-classiffies the walking-downstairs and upstairs class-labels which in turn reduces the total accuracy of the model.

6.7.3.3. Taking the number of neurons as 64 and drop-out value as 0.6

In [119]:
```
model_3=lstm_model(64,0.6,1)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_29 (LSTM)               (None, 64)                18944     
_________________________________________________________________
dropout_26 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_26 (Dense)             (None, 6)                 390       
=================================================================
Total params: 19,334
Trainable params: 19,334
Non-trainable params: 0
_________________________________________________________________
```

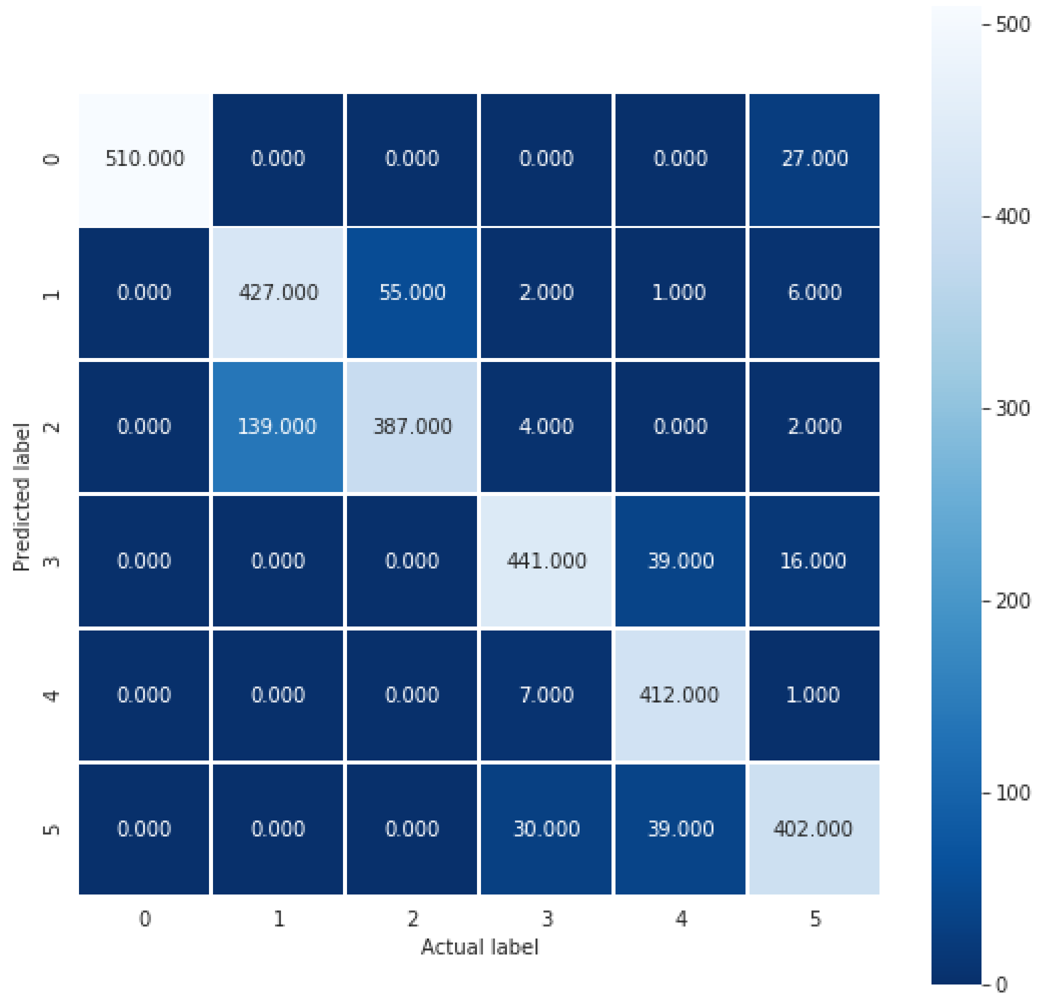
In [120]:
```
train_3=train(X_train,Y_train,X_test, Y_test,64,30,model_3)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 14s - loss: 2.5867 - acc: 0.3708 - val_loss: 2.3113 - val_acc: 0.4242
Epoch 2/30
 - 12s - loss: 2.1354 - acc: 0.4898 - val_loss: 1.9968 - val_acc: 0.4700
Epoch 3/30
 - 12s - loss: 1.8535 - acc: 0.5486 - val_loss: 1.7571 - val_acc: 0.5314
Epoch 4/30
 - 12s - loss: 1.6627 - acc: 0.5896 - val_loss: 1.6135 - val_acc: 0.5779
Epoch 5/30
 - 12s - loss: 1.4852 - acc: 0.6250 - val_loss: 1.5001 - val_acc: 0.5850
Epoch 6/30
 - 12s - loss: 1.3188 - acc: 0.6280 - val_loss: 1.6074 - val_acc: 0.4856
Epoch 7/30
 - 12s - loss: 1.1540 - acc: 0.6450 - val_loss: 1.1616 - val_acc: 0.6098
Epoch 8/30
 - 12s - loss: 1.0522 - acc: 0.6451 - val_loss: 1.0876 - val_acc: 0.6084
Epoch 9/30
 - 12s - loss: 0.9713 - acc: 0.6492 - val_loss: 0.9714 - val_acc: 0.6067
Epoch 10/30
 - 12s - loss: 0.8754 - acc: 0.6545 - val_loss: 0.8746 - val_acc: 0.6108
Epoch 11/30
 - 12s - loss: 0.7807 - acc: 0.6581 - val_loss: 1.1146 - val_acc: 0.5168
Epoch 12/30
 - 12s - loss: 0.7042 - acc: 0.6766 - val_loss: 0.7497 - val_acc: 0.6244
Epoch 13/30
 - 12s - loss: 0.6725 - acc: 0.6847 - val_loss: 0.8149 - val_acc: 0.6155
Epoch 14/30
 - 12s - loss: 0.6347 - acc: 0.7057 - val_loss: 0.8081 - val_acc: 0.6444
Epoch 15/30
 - 12s - loss: 0.5788 - acc: 0.7421 - val_loss: 0.7070 - val_acc: 0.7387
Epoch 16/30
 - 12s - loss: 0.5763 - acc: 0.7507 - val_loss: 0.6567 - val_acc: 0.7357
Epoch 17/30
 - 12s - loss: 0.5535 - acc: 0.7791 - val_loss: 0.6548 - val_acc: 0.7201
Epoch 18/30
 - 12s - loss: 0.4839 - acc: 0.7901 - val_loss: 0.7601 - val_acc: 0.7184
Epoch 19/30
 - 12s - loss: 0.4536 - acc: 0.7947 - val_loss: 0.6362 - val_acc: 0.7234
Epoch 20/30
 - 12s - loss: 0.4627 - acc: 0.8020 - val_loss: 0.6330 - val_acc: 0.7431
Epoch 21/30
 - 12s - loss: 0.4184 - acc: 0.8064 - val_loss: 0.6202 - val_acc: 0.7465
Epoch 22/30
 - 12s - loss: 0.4214 - acc: 0.8154 - val_loss: 0.6138 - val_acc: 0.7754
Epoch 23/30
 - 12s - loss: 0.4493 - acc: 0.8166 - val_loss: 0.5300 - val_acc: 0.8039
Epoch 24/30
 - 12s - loss: 0.3793 - acc: 0.8494 - val_loss: 0.5170 - val_acc: 0.8568
Epoch 25/30
 - 12s - loss: 0.3144 - acc: 0.8998 - val_loss: 0.4252 - val_acc: 0.8741
Epoch 26/30
 - 12s - loss: 0.2901 - acc: 0.9102 - val_loss: 0.5056 - val_acc: 0.8327
Epoch 27/30
 - 12s - loss: 0.2905 - acc: 0.9119 - val_loss: 0.4175 - val_acc: 0.8772
Epoch 28/30
 - 12s - loss: 0.2338 - acc: 0.9291 - val_loss: 0.4808 - val_acc: 0.8660
Epoch 29/30
 - 12s - loss: 0.2521 - acc: 0.9237 - val_loss: 0.4510 - val_acc: 0.8839
Epoch 30/30
 - 12s - loss: 0.1994 - acc: 0.9361 - val_loss: 0.6186 - val_acc: 0.8751
```

In [177]:

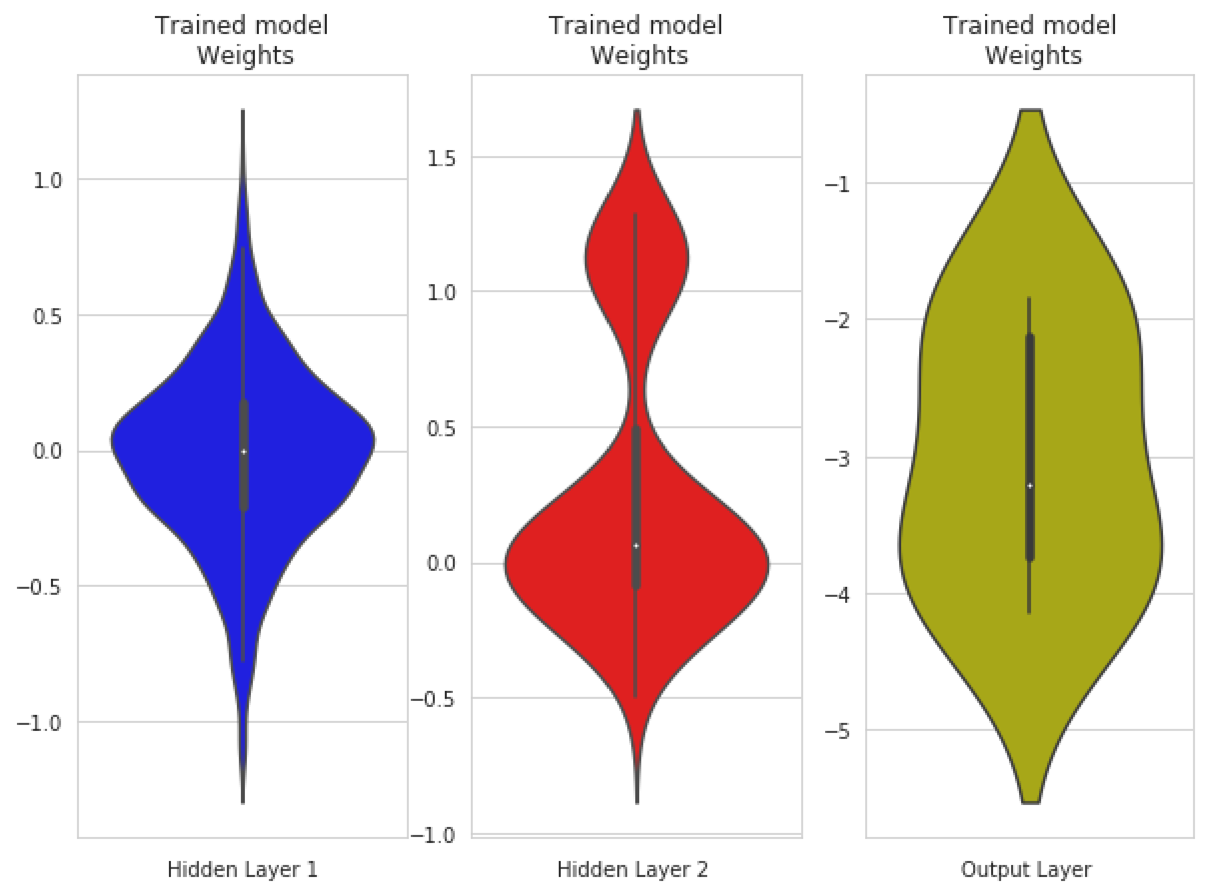
```
# Confusion Matrix
print(confusion_matrix(Y_test, model_3.predict(X_test)))
```

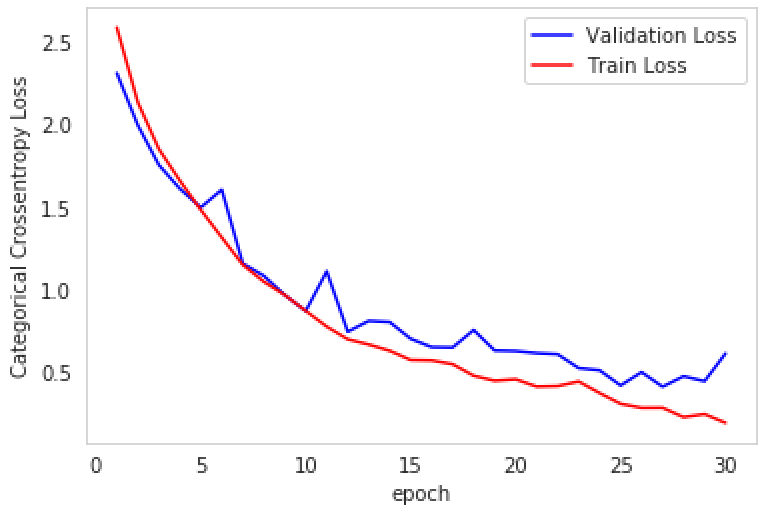
```
[[510   0   0   0   0  27]
 [  0 427  55   2   1   6]
 [  0 139 387   4   0   2]
 [  0   0   0 441  39  16]
 [  0   0   0   7 412   1]
 [  0   0   0  30  39 402]]
```




```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0         0        0                   0
SITTING                  0      427        55        2                   1
STANDING                 0      139       387        4                   0
WALKING                  0        0         0      441                  39
WALKING_DOWNSTAIRS       0        0         0        7                 412
WALKING_UPSTAIRS         0        0         0       30                  39

Pred                WALKING_UPSTAIRS
True
LAYING                            27
SITTING                            6
STANDING                           2
WALKING                           16
WALKING_DOWNSTAIRS                 1
WALKING_UPSTAIRS                 402
```

In [163]:
```python
# Plot weight distribution using violin plot
import warnings
warnings.filterwarnings(action='ignore')
plot_weights(model)
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(train_3,epochs)
```





- This LSTM model has 64 neurons with a drop-out rate of 0.6 and results are quite satisfactory than the previous tuned models.
- The model is less over-fitting and with a greater and better accuracy than the previous models which also have very low error rates.
- Still there is some miss-classification among the sitting and standing-class but mangeable.So this LSTM model is best tuned model so far.

6.7.3.4. Taking the number of neurons as 64 and drop-out value as 0.75

In [121]:
```
model_3_1=lstm_model(64,0.75,1)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_30 (LSTM)               (None, 64)                18944     
_________________________________________________________________
dropout_27 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_27 (Dense)             (None, 6)                 390       
=================================================================
Total params: 19,334
Trainable params: 19,334
Non-trainable params: 0
_________________________________________________________________
```

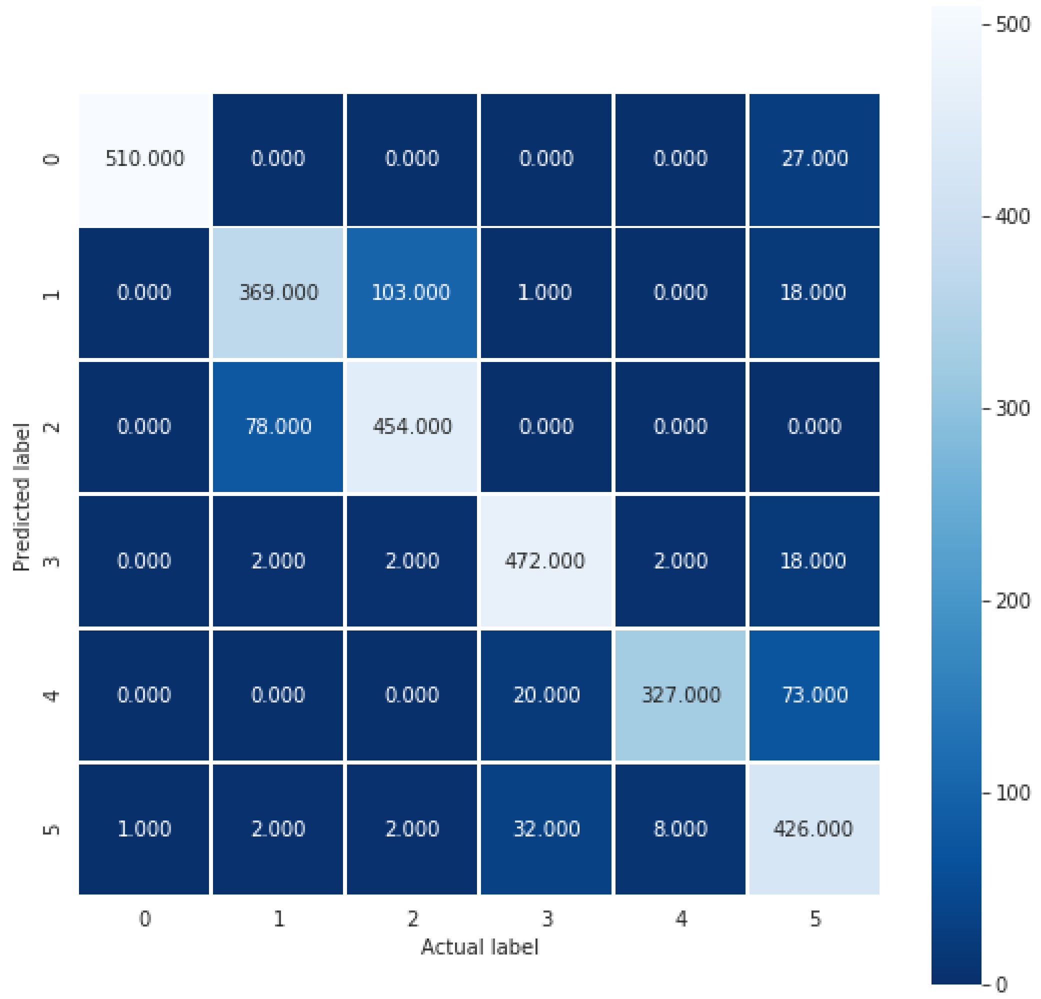
In [122]:
```
train_3_1=train(X_train,Y_train,X_test, Y_test,64,30,model_3_1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 14s - loss: 2.6324 - acc: 0.3792 - val_loss: 2.3381 - val_acc: 0.4520
Epoch 2/30
 - 11s - loss: 2.2883 - acc: 0.4463 - val_loss: 2.2429 - val_acc: 0.3882
Epoch 3/30
 - 11s - loss: 2.0337 - acc: 0.5011 - val_loss: 1.8749 - val_acc: 0.4978
Epoch 4/30
 - 12s - loss: 1.7139 - acc: 0.5604 - val_loss: 1.5438 - val_acc: 0.6539
Epoch 5/30
 - 11s - loss: 1.4814 - acc: 0.5967 - val_loss: 1.3780 - val_acc: 0.6101
Epoch 6/30
 - 12s - loss: 1.3227 - acc: 0.6140 - val_loss: 1.2137 - val_acc: 0.6423
Epoch 7/30
 - 12s - loss: 1.1567 - acc: 0.6387 - val_loss: 1.1195 - val_acc: 0.6590
Epoch 8/30
 - 12s - loss: 1.0351 - acc: 0.6466 - val_loss: 0.9697 - val_acc: 0.6084
Epoch 9/30
 - 11s - loss: 0.9457 - acc: 0.6572 - val_loss: 0.9198 - val_acc: 0.6155
Epoch 10/30
 - 11s - loss: 0.8995 - acc: 0.6631 - val_loss: 0.8365 - val_acc: 0.6105
Epoch 11/30
 - 11s - loss: 0.8413 - acc: 0.6770 - val_loss: 0.7893 - val_acc: 0.6115
Epoch 12/30
 - 11s - loss: 0.7679 - acc: 0.6834 - val_loss: 0.7635 - val_acc: 0.6172
Epoch 13/30
 - 11s - loss: 0.8130 - acc: 0.6700 - val_loss: 0.7190 - val_acc: 0.6152
Epoch 14/30
 - 11s - loss: 0.6688 - acc: 0.6974 - val_loss: 1.2293 - val_acc: 0.5100
Epoch 15/30
 - 11s - loss: 0.6577 - acc: 0.7084 - val_loss: 0.9192 - val_acc: 0.6617
Epoch 16/30
 - 11s - loss: 0.6212 - acc: 0.7387 - val_loss: 0.6467 - val_acc: 0.7346
Epoch 17/30
 - 11s - loss: 0.6556 - acc: 0.7436 - val_loss: 0.5909 - val_acc: 0.7581
Epoch 18/30
 - 11s - loss: 0.5879 - acc: 0.7824 - val_loss: 0.5428 - val_acc: 0.7995
Epoch 19/30
 - 11s - loss: 0.5605 - acc: 0.7911 - val_loss: 0.6602 - val_acc: 0.7550
Epoch 20/30
 - 11s - loss: 0.5698 - acc: 0.8187 - val_loss: 1.3455 - val_acc: 0.7065
Epoch 21/30
 - 11s - loss: 0.5273 - acc: 0.8328 - val_loss: 0.4985 - val_acc: 0.8140
Epoch 22/30
 - 11s - loss: 0.4257 - acc: 0.8751 - val_loss: 0.4894 - val_acc: 0.8599
Epoch 23/30
 - 11s - loss: 0.3932 - acc: 0.8890 - val_loss: 0.4405 - val_acc: 0.8772
Epoch 24/30
 - 11s - loss: 0.3985 - acc: 0.8803 - val_loss: 0.3912 - val_acc: 0.8867
Epoch 25/30
 - 11s - loss: 0.3762 - acc: 0.8946 - val_loss: 0.3822 - val_acc: 0.8792
Epoch 26/30
 - 11s - loss: 0.3181 - acc: 0.9100 - val_loss: 0.4105 - val_acc: 0.8707
Epoch 27/30
 - 12s - loss: 0.3132 - acc: 0.9123 - val_loss: 0.4740 - val_acc: 0.8802
Epoch 28/30
 - 12s - loss: 0.2857 - acc: 0.9170 - val_loss: 0.4276 - val_acc: 0.8873
Epoch 29/30
 - 12s - loss: 0.2716 - acc: 0.9210 - val_loss: 0.6931 - val_acc: 0.8517
Epoch 30/30
 - 12s - loss: 0.2559 - acc: 0.9195 - val_loss: 0.5220 - val_acc: 0.8680
```

In [178]:

```
# Confusion Matrix
print(confusion_matrix(Y_test, model_3_1.predict(X_test)))
```

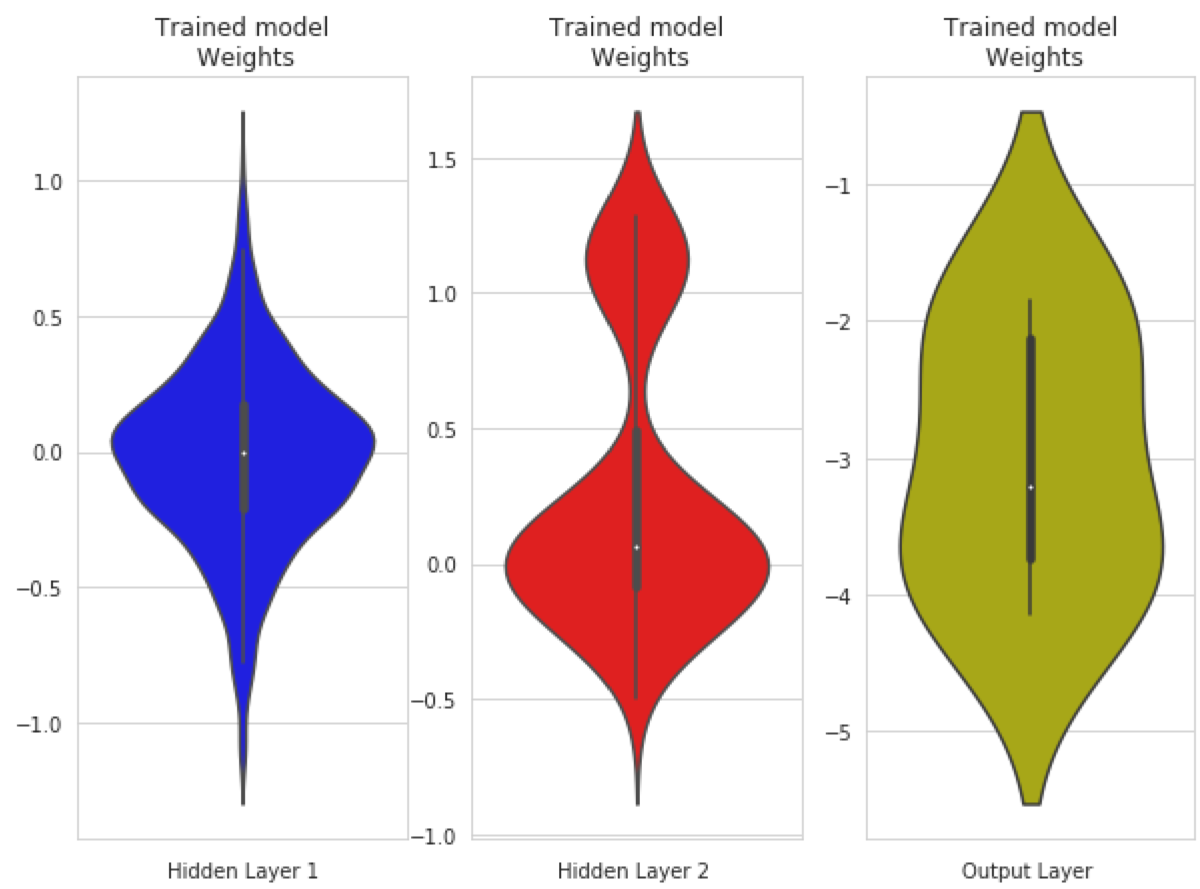
```
[[510   0   0   0   0  27]
 [  0 369 103   1   0  18]
 [  0  78 454   0   0   0]
 [  0   2   2 472   2  18]
 [  0   0   0  20 327  73]
 [  1   2   2  32   8 426]]
```

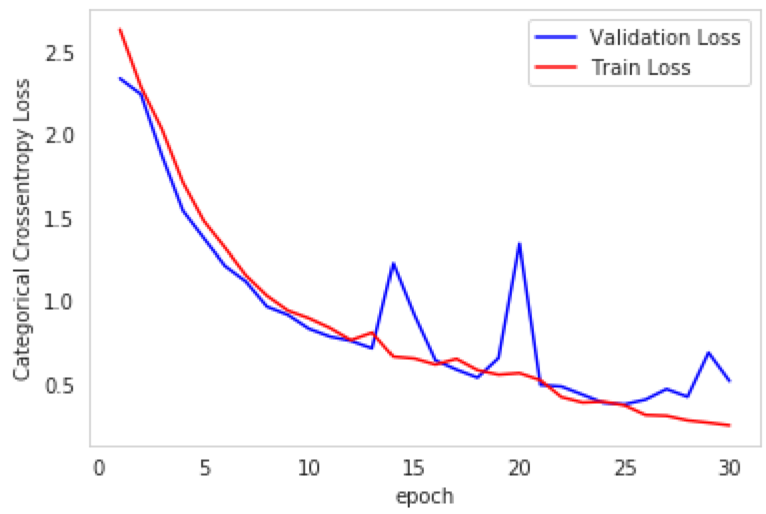



```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0         0        0                   0
SITTING                  0      369       103        1                   0
STANDING                 0       78       454        0                   0
WALKING                  0        2         2      472                   2
WALKING_DOWNSTAIRS       0        0         0       20                 327
WALKING_UPSTAIRS         1        2         2       32                   8

Pred                WALKING_UPSTAIRS
True
LAYING                            27
SITTING                           18
STANDING                           0
WALKING                           18
WALKING_DOWNSTAIRS                73
WALKING_UPSTAIRS                 426
```

In [164]:

```
# Plot weight distribution using violin plot
import warnings
warnings.filterwarnings(action='ignore')
plot_weights(model)
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(train_3_1,epochs)
```





- This tuned LSTM model is good but the error rate is increased as compared with the previous tuned model and the accuracy also dropped by 2%.
- The erorr plot also not that good than previous one so the previous tuning is better than thios one.

6.7.3.5. Taking the number of neurons as 128 and drop-out value as 0.6

In [123]:
```
model_4=lstm_model(128,0.6,1)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_31 (LSTM)               (None, 128)               70656     
_________________________________________________________________
dropout_28 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_28 (Dense)             (None, 6)                 774       
=================================================================
Total params: 71,430
Trainable params: 71,430
Non-trainable params: 0
_________________________________________________________________
```

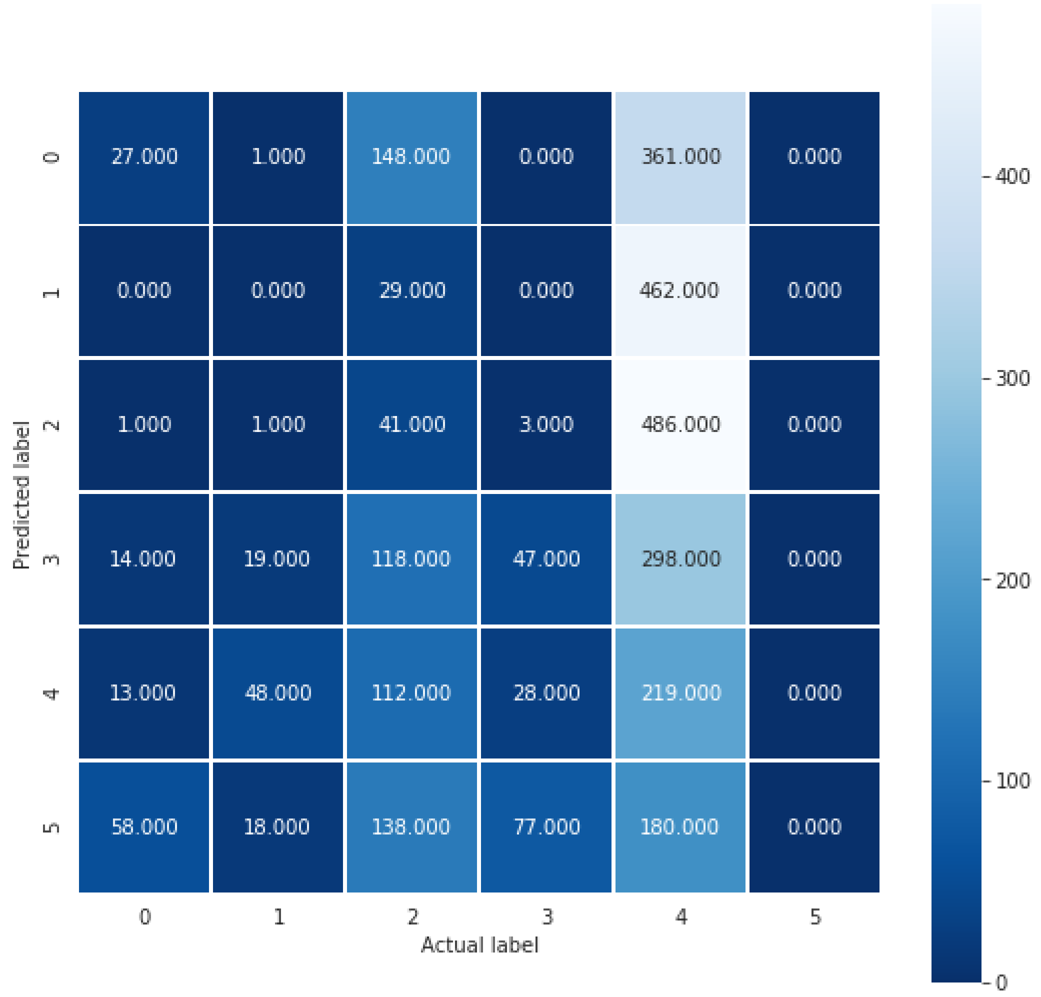
In [124]:
```
train_4=train(X_train,Y_train,X_test, Y_test,64,30,model_4)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 31s - loss: 3.6848 - acc: 0.3834 - val_loss: 3.3890 - val_acc: 0.4574
Epoch 2/30
 - 29s - loss: 3.2367 - acc: 0.4323 - val_loss: 3.0940 - val_acc: 0.4245
Epoch 3/30
 - 28s - loss: 2.8122 - acc: 0.4769 - val_loss: 2.6624 - val_acc: 0.4662
Epoch 4/30
 - 29s - loss: 2.3312 - acc: 0.5506 - val_loss: 2.2281 - val_acc: 0.5439
Epoch 5/30
 - 28s - loss: 2.1028 - acc: 0.5152 - val_loss: 1.9789 - val_acc: 0.5070
Epoch 6/30
 - 29s - loss: 1.6437 - acc: 0.5963 - val_loss: 1.6749 - val_acc: 0.5338
Epoch 7/30
 - 29s - loss: 1.4139 - acc: 0.6077 - val_loss: 1.3404 - val_acc: 0.5935
Epoch 8/30
 - 29s - loss: 1.1441 - acc: 0.6281 - val_loss: 1.1593 - val_acc: 0.5969
Epoch 9/30
 - 28s - loss: 0.9793 - acc: 0.6268 - val_loss: 1.0446 - val_acc: 0.6155
Epoch 10/30
 - 28s - loss: 0.7999 - acc: 0.6423 - val_loss: 0.8776 - val_acc: 0.6006
Epoch 11/30
 - 28s - loss: 0.7215 - acc: 0.6556 - val_loss: 1.2390 - val_acc: 0.5232
Epoch 12/30
 - 28s - loss: 0.6949 - acc: 0.6859 - val_loss: 0.8930 - val_acc: 0.6423
Epoch 13/30
 - 28s - loss: 0.6298 - acc: 0.7148 - val_loss: 0.7291 - val_acc: 0.6987
Epoch 14/30
 - 28s - loss: 0.5969 - acc: 0.7293 - val_loss: 0.6889 - val_acc: 0.6922
Epoch 15/30
 - 28s - loss: 0.5458 - acc: 0.7622 - val_loss: 0.7991 - val_acc: 0.6936
Epoch 16/30
 - 28s - loss: 0.5532 - acc: 0.7722 - val_loss: 0.6366 - val_acc: 0.7838
Epoch 17/30
 - 28s - loss: 0.5213 - acc: 0.8086 - val_loss: 0.7221 - val_acc: 0.7750
Epoch 18/30
 - 29s - loss: 0.5212 - acc: 0.8036 - val_loss: 0.6846 - val_acc: 0.7978
Epoch 19/30
 - 28s - loss: 0.3994 - acc: 0.8464 - val_loss: 0.4362 - val_acc: 0.8442
Epoch 20/30
 - 28s - loss: 0.3528 - acc: 0.8783 - val_loss: 0.4696 - val_acc: 0.8599
Epoch 21/30
 - 28s - loss: 0.2986 - acc: 0.8981 - val_loss: 0.3565 - val_acc: 0.8802
Epoch 22/30
 - 29s - loss: 0.2306 - acc: 0.9176 - val_loss: 0.3421 - val_acc: 0.8941
Epoch 23/30
 - 29s - loss: 0.2278 - acc: 0.9191 - val_loss: 0.7984 - val_acc: 0.8347
Epoch 24/30
 - 29s - loss: 0.2189 - acc: 0.9272 - val_loss: 0.3240 - val_acc: 0.9006
Epoch 25/30
 - 29s - loss: 0.2483 - acc: 0.9200 - val_loss: 0.3176 - val_acc: 0.9016
Epoch 26/30
 - 29s - loss: 0.2086 - acc: 0.9294 - val_loss: 0.2894 - val_acc: 0.9026
Epoch 27/30
 - 29s - loss: 0.1737 - acc: 0.9392 - val_loss: 0.3590 - val_acc: 0.8958
Epoch 28/30
 - 28s - loss: 0.1705 - acc: 0.9399 - val_loss: 0.3468 - val_acc: 0.8996
Epoch 29/30
 - 29s - loss: 0.1679 - acc: 0.9380 - val_loss: 0.2849 - val_acc: 0.9128
Epoch 30/30
 - 29s - loss: 0.1494 - acc: 0.9441 - val_loss: 0.3388 - val_acc: 0.9077
```

In [179]:

```
# Confusion Matrix
print(confusion_matrix(Y_test, model_4.predict(X_test)))
```

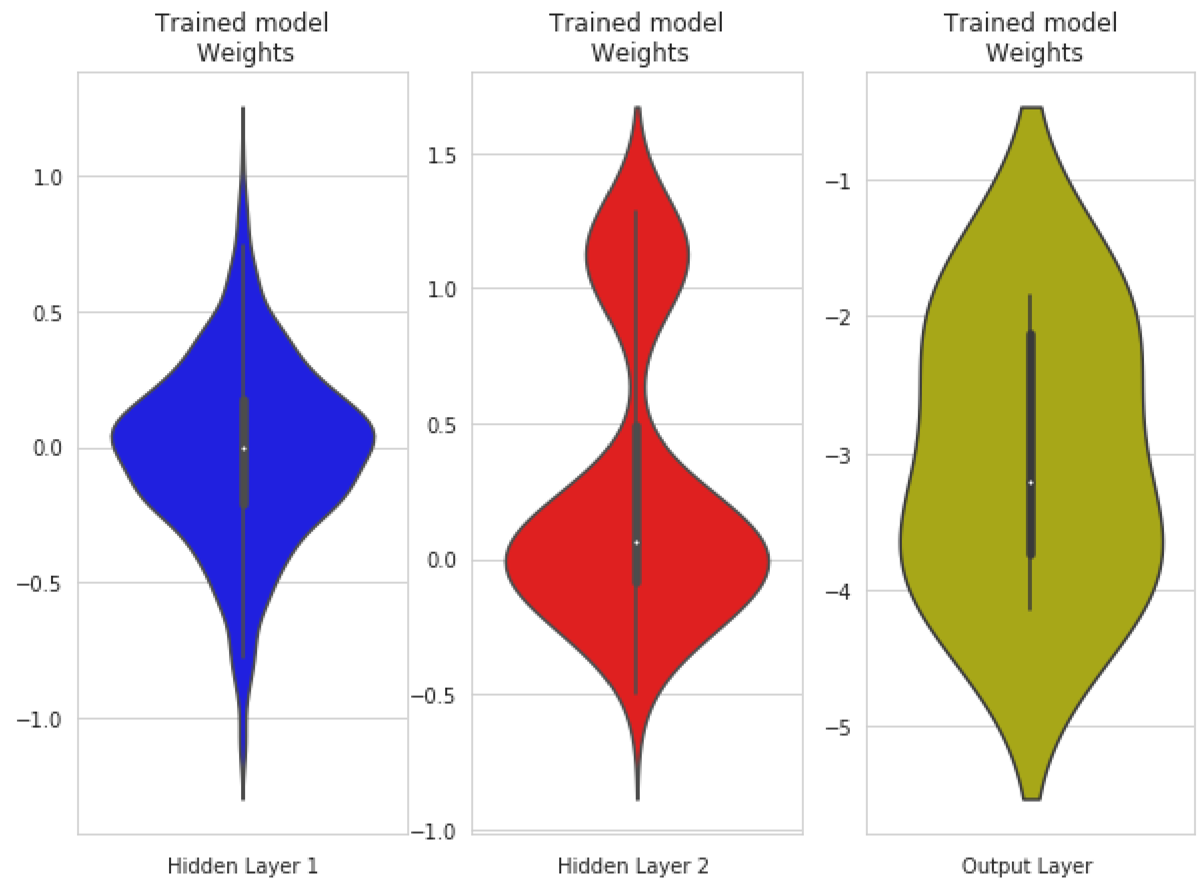
```
[[ 27   1 148   0 361   0]
 [  0   0  29   0 462   0]
 [  1   1  41   3 486   0]
 [ 14  19 118  47 298   0]
 [ 13  48 112  28 219   0]
 [ 58  18 138  77 180   0]]
```

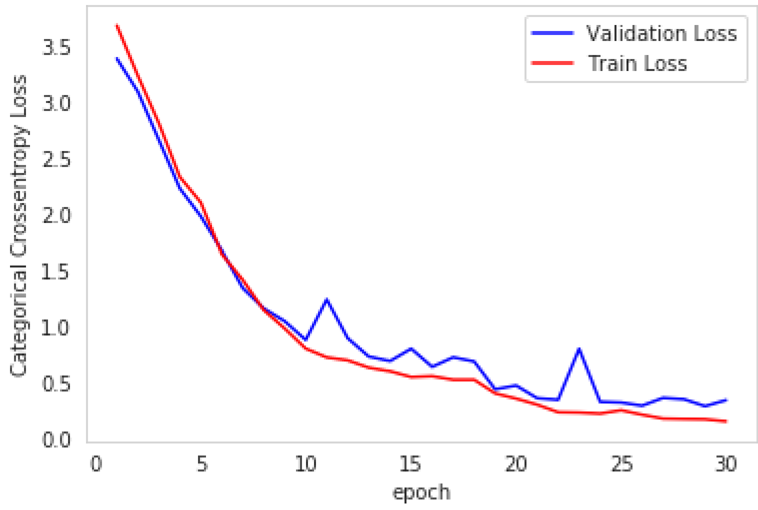



```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS
True
LAYING                  27        1       148        0                 361
SITTING                  0        0        29        0                 462
STANDING                 1        1        41        3                 486
WALKING                 14       19       118       47                 298
WALKING_DOWNSTAIRS      13       48       112       28                 219
WALKING_UPSTAIRS        58       18       138       77                 180
```

In [165]:

```python
# Plot weight distribution using violin plot
import warnings
warnings.filterwarnings(action='ignore')
plot_weights(model)
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(train_4,epochs)
```





- This LSTM model with 128 neurons and 0.6 drop-out value has the best performace values with 94.41% accuracy and 0.14 error which is better than all the previous models.

6.7.3.6. Taking the number of neurons as 128 and drop-out value as 0.75

In [126]:
```
model_4_1=lstm_model(128,0.75,1)
```

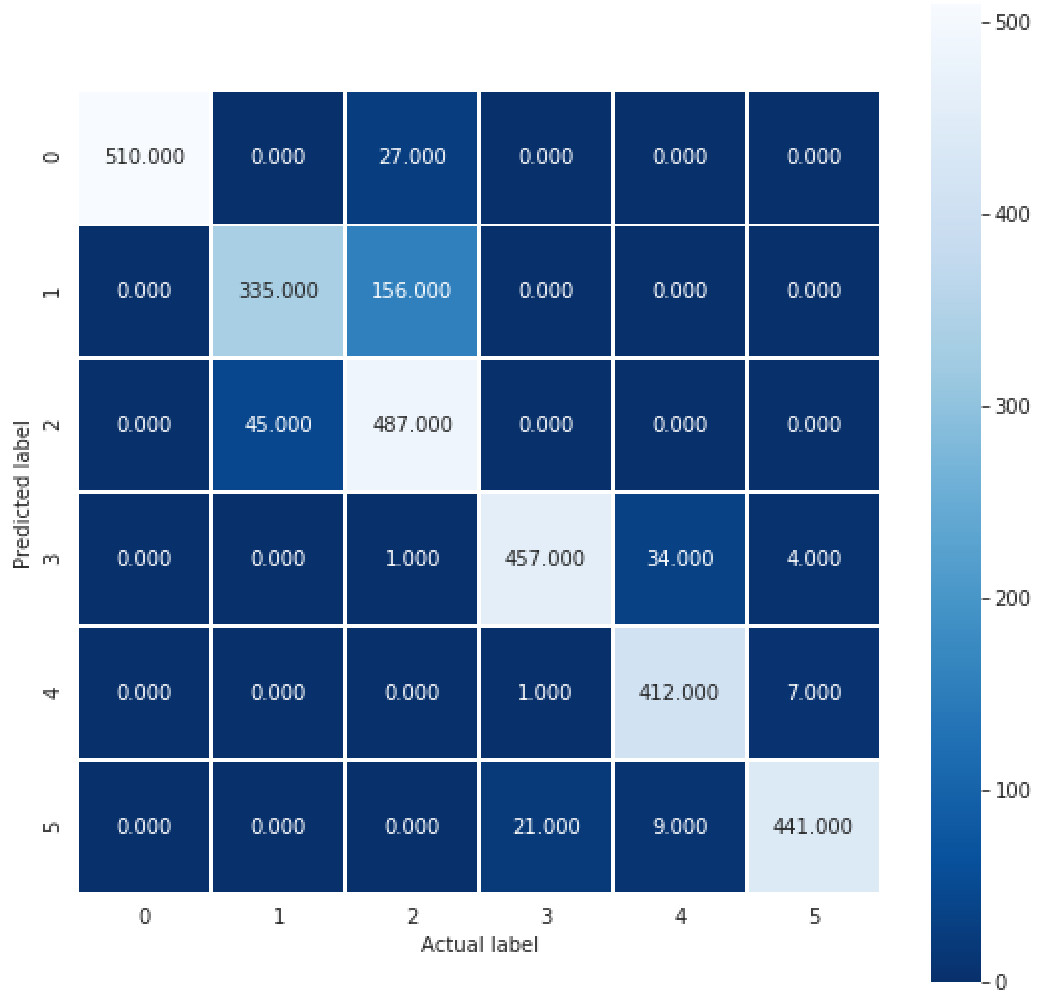
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_33 (LSTM)               (None, 128)               70656     
_________________________________________________________________
dropout_30 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_30 (Dense)             (None, 6)                 774       
=================================================================
Total params: 71,430
Trainable params: 71,430
Non-trainable params: 0
_________________________________________________________________
```

In [127]:
```
train_4_1=train(X_train,Y_train,X_test, Y_test,64,30,model_4_1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 32s - loss: 3.7256 - acc: 0.3920 - val_loss: 3.5065 - val_acc: 0.3441
Epoch 2/30
 - 29s - loss: 3.1490 - acc: 0.4837 - val_loss: 3.1700 - val_acc: 0.4153
Epoch 3/30
 - 29s - loss: 2.7626 - acc: 0.4966 - val_loss: 2.5844 - val_acc: 0.5029
Epoch 4/30
 - 30s - loss: 2.3055 - acc: 0.5264 - val_loss: 2.0685 - val_acc: 0.5606
Epoch 5/30
 - 30s - loss: 1.8926 - acc: 0.5715 - val_loss: 1.7402 - val_acc: 0.6149
Epoch 6/30
 - 30s - loss: 1.6156 - acc: 0.6064 - val_loss: 1.6212 - val_acc: 0.5477
Epoch 7/30
 - 30s - loss: 1.4091 - acc: 0.6175 - val_loss: 1.3276 - val_acc: 0.6013
Epoch 8/30
 - 30s - loss: 1.1595 - acc: 0.6383 - val_loss: 1.1000 - val_acc: 0.6043
Epoch 9/30
 - 30s - loss: 0.9437 - acc: 0.6519 - val_loss: 0.9340 - val_acc: 0.6216
Epoch 10/30
 - 30s - loss: 0.9565 - acc: 0.6299 - val_loss: 0.9670 - val_acc: 0.5803
Epoch 11/30
 - 30s - loss: 0.7591 - acc: 0.6508 - val_loss: 0.8622 - val_acc: 0.5765
Epoch 12/30
 - 29s - loss: 0.6978 - acc: 0.6590 - val_loss: 0.7220 - val_acc: 0.6223
Epoch 13/30
 - 30s - loss: 0.7188 - acc: 0.6570 - val_loss: 0.8565 - val_acc: 0.6064
Epoch 14/30
 - 30s - loss: 0.7438 - acc: 0.6492 - val_loss: 0.7556 - val_acc: 0.6152
Epoch 15/30
 - 30s - loss: 0.6465 - acc: 0.6791 - val_loss: 0.7188 - val_acc: 0.6366
Epoch 16/30
 - 29s - loss: 0.6323 - acc: 0.6726 - val_loss: 0.7269 - val_acc: 0.6166
Epoch 17/30
 - 29s - loss: 0.7210 - acc: 0.6579 - val_loss: 3.0968 - val_acc: 0.4825
Epoch 18/30
 - 29s - loss: 0.7829 - acc: 0.6451 - val_loss: 0.6934 - val_acc: 0.6278
Epoch 19/30
 - 29s - loss: 0.6731 - acc: 0.6572 - val_loss: 0.7198 - val_acc: 0.6454
Epoch 20/30
 - 29s - loss: 0.5919 - acc: 0.6980 - val_loss: 1.0953 - val_acc: 0.6586
Epoch 21/30
 - 30s - loss: 0.5319 - acc: 0.7870 - val_loss: 0.9875 - val_acc: 0.7268
Epoch 22/30
 - 29s - loss: 0.4391 - acc: 0.8316 - val_loss: 0.6024 - val_acc: 0.7543
Epoch 23/30
 - 30s - loss: 0.4147 - acc: 0.8343 - val_loss: 0.5896 - val_acc: 0.7669
Epoch 24/30
 - 30s - loss: 0.3546 - acc: 0.8856 - val_loss: 0.5453 - val_acc: 0.7933
Epoch 25/30
 - 30s - loss: 0.3077 - acc: 0.9090 - val_loss: 0.4952 - val_acc: 0.8806
Epoch 26/30
 - 30s - loss: 0.2840 - acc: 0.9142 - val_loss: 0.4607 - val_acc: 0.8792
Epoch 27/30
 - 30s - loss: 0.2587 - acc: 0.9203 - val_loss: 0.8064 - val_acc: 0.8022
Epoch 28/30
 - 30s - loss: 0.2599 - acc: 0.9232 - val_loss: 0.4176 - val_acc: 0.8660
Epoch 29/30
 - 30s - loss: 0.2412 - acc: 0.9274 - val_loss: 0.4691 - val_acc: 0.8856
Epoch 30/30
 - 30s - loss: 0.2080 - acc: 0.9340 - val_loss: 0.4679 - val_acc: 0.8965
```

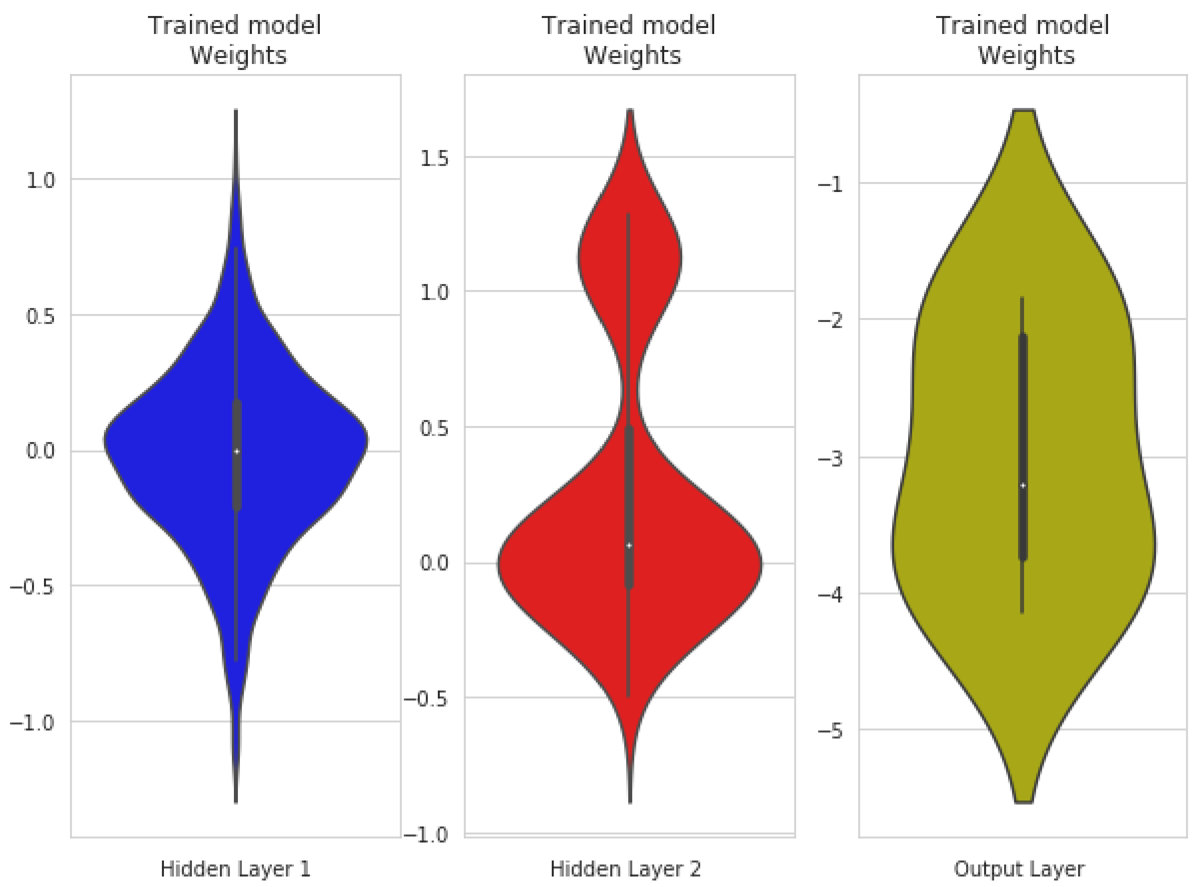
In [181]:
```
# Confusion Matrix
print(confusion_matrix(Y_test, model_4_1.predict(X_test)))
```

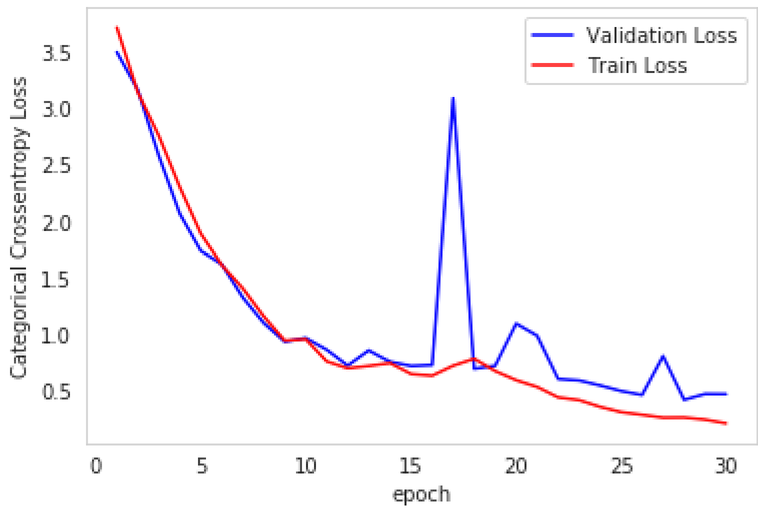
```
[[510   0  27   0   0   0]
 [  0 335 156   0   0   0]
 [  0  45 487   0   0   0]
 [  0   0   1 457  34   4]
 [  0   0   0   1 412   7]
 [  0   0   0  21   9 441]]
```




```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0        27        0                   0
SITTING                  0      335       156        0                   0
STANDING                 0       45       487        0                   0
WALKING                  0        0         1      457                  34
WALKING_DOWNSTAIRS       0        0         0        1                 412
WALKING_UPSTAIRS         0        0         0       21                   9

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            0
STANDING                           0
WALKING                            4
WALKING_DOWNSTAIRS                 7
WALKING_UPSTAIRS                 441
```

In [167]:
```python
# Plot weight distribution using violin plot
import warnings
warnings.filterwarnings(action='ignore')
plot_weights(model)
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(train_4_1,epochs)
```





6.8. Implementing two-layered LSTM model

Utility function for building the architecture of a 2-layer LSTM model

In [150]:
```
def lstm_model_2L(neurons1,neurons2,Dp):
    # Initiliazing the sequential model
    Model = Sequential()
    # Configuring the parameters
    Model.add(LSTM(neurons1, input_shape=(timesteps, input_dim),bias_regularizer=reg,return_sequences=
True))
    Model.add(BatchNormalization())
    Model.add(Dropout(Dp))

    Model.add(LSTM(neurons2,bias_regularizer=reg))
    Model.add(Dropout(Dp))

    # Adding a dense output layer with sigmoid activation
    Model.add(Dense(n_classes, activation='sigmoid'))
    Model.summary()
    # Compiling the model
    Model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

    return Model
```

6.8.1 Two layered LSTM model with 48,32 neurons and 0.6 drop-out

In [151]:
```
model_2l_1=lstm_model_2L(48,32,0.6)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_54 (LSTM)               (None, 128, 48)           11136     
_________________________________________________________________
batch_normalization_6 (Batch (None, 128, 48)           192       
_________________________________________________________________
dropout_40 (Dropout)         (None, 128, 48)           0         
_________________________________________________________________
lstm_55 (LSTM)               (None, 32)                10368     
_________________________________________________________________
dropout_41 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_32 (Dense)             (None, 6)                 198       
=================================================================
Total params: 21,894
Trainable params: 21,798
Non-trainable params: 96
_________________________________________________________________
```

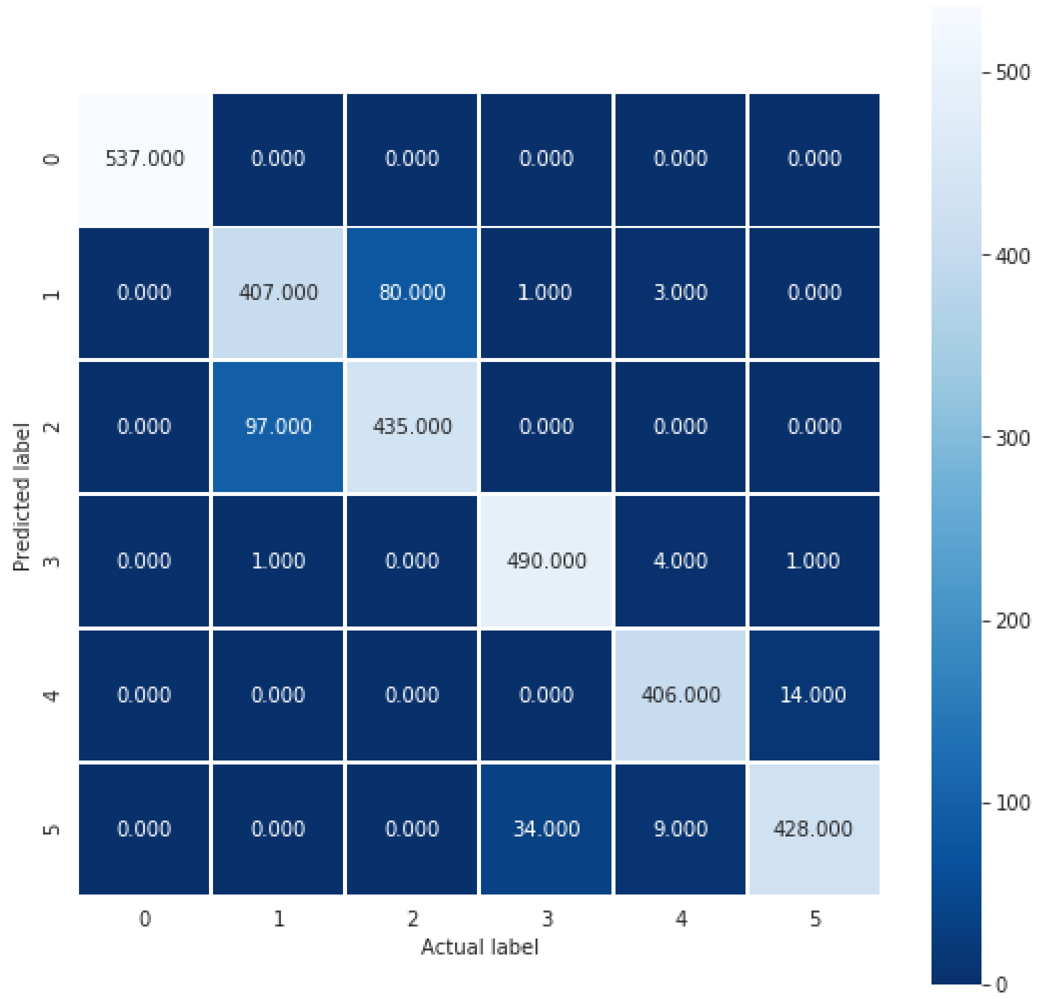
In [152]:
```
train_2l_1=train(X_train,Y_train,X_test, Y_test,128,30,model_2l_1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 27s - loss: 3.0369 - acc: 0.4739 - val_loss: 2.8159 - val_acc: 0.5137
Epoch 2/30
 - 23s - loss: 2.5733 - acc: 0.6136 - val_loss: 2.4458 - val_acc: 0.6485
Epoch 3/30
 - 23s - loss: 2.2431 - acc: 0.6538 - val_loss: 2.2168 - val_acc: 0.6871
Epoch 4/30
 - 23s - loss: 2.0099 - acc: 0.6795 - val_loss: 1.8880 - val_acc: 0.6827
Epoch 5/30
 - 23s - loss: 1.8184 - acc: 0.7164 - val_loss: 1.7817 - val_acc: 0.6485
Epoch 6/30
 - 23s - loss: 1.6652 - acc: 0.7418 - val_loss: 1.5462 - val_acc: 0.8120
Epoch 7/30
 - 23s - loss: 1.5083 - acc: 0.7737 - val_loss: 2.2952 - val_acc: 0.4248
Epoch 8/30
 - 24s - loss: 1.3695 - acc: 0.7933 - val_loss: 1.5605 - val_acc: 0.6515
Epoch 9/30
 - 23s - loss: 1.2513 - acc: 0.8139 - val_loss: 1.1558 - val_acc: 0.8208
Epoch 10/30
 - 24s - loss: 1.1295 - acc: 0.8345 - val_loss: 1.1076 - val_acc: 0.8124
Epoch 11/30
 - 24s - loss: 1.0395 - acc: 0.8334 - val_loss: 1.0189 - val_acc: 0.8368
Epoch 12/30
 - 23s - loss: 0.9161 - acc: 0.8708 - val_loss: 0.8522 - val_acc: 0.8649
Epoch 13/30
 - 24s - loss: 0.8217 - acc: 0.8851 - val_loss: 0.7908 - val_acc: 0.8724
Epoch 14/30
 - 24s - loss: 0.7500 - acc: 0.8875 - val_loss: 0.7071 - val_acc: 0.8724
Epoch 15/30
 - 23s - loss: 0.6877 - acc: 0.8973 - val_loss: 0.7332 - val_acc: 0.8565
Epoch 16/30
 - 24s - loss: 0.6225 - acc: 0.9019 - val_loss: 1.0125 - val_acc: 0.7431
Epoch 17/30
 - 23s - loss: 0.5485 - acc: 0.9108 - val_loss: 0.6371 - val_acc: 0.8327
Epoch 18/30
 - 23s - loss: 0.4831 - acc: 0.9226 - val_loss: 0.5953 - val_acc: 0.8578
Epoch 19/30
 - 24s - loss: 0.4425 - acc: 0.9172 - val_loss: 0.4303 - val_acc: 0.8894
Epoch 20/30
 - 23s - loss: 0.4014 - acc: 0.9238 - val_loss: 0.6079 - val_acc: 0.8582
Epoch 21/30
 - 23s - loss: 0.3632 - acc: 0.9214 - val_loss: 0.3417 - val_acc: 0.8996
Epoch 22/30
 - 23s - loss: 0.3236 - acc: 0.9285 - val_loss: 0.4791 - val_acc: 0.8568
Epoch 23/30
 - 23s - loss: 0.2817 - acc: 0.9285 - val_loss: 0.3980 - val_acc: 0.8744
Epoch 24/30
 - 23s - loss: 0.2583 - acc: 0.9241 - val_loss: 0.2879 - val_acc: 0.9094
Epoch 25/30
 - 24s - loss: 0.2457 - acc: 0.9350 - val_loss: 0.3712 - val_acc: 0.8809
Epoch 26/30
 - 24s - loss: 0.2405 - acc: 0.9346 - val_loss: 0.3276 - val_acc: 0.9074
Epoch 27/30
 - 24s - loss: 0.2411 - acc: 0.9293 - val_loss: 0.2473 - val_acc: 0.9138
Epoch 28/30
 - 24s - loss: 0.2232 - acc: 0.9354 - val_loss: 0.2467 - val_acc: 0.9199
Epoch 29/30
 - 24s - loss: 0.2146 - acc: 0.9376 - val_loss: 0.3407 - val_acc: 0.9036
Epoch 30/30
 - 23s - loss: 0.2092 - acc: 0.9370 - val_loss: 0.2597 - val_acc: 0.9172
```

In [182]:

```
# Confusion Matrix
print(confusion_matrix(Y_test,model_2l_1.predict(X_test)))
```

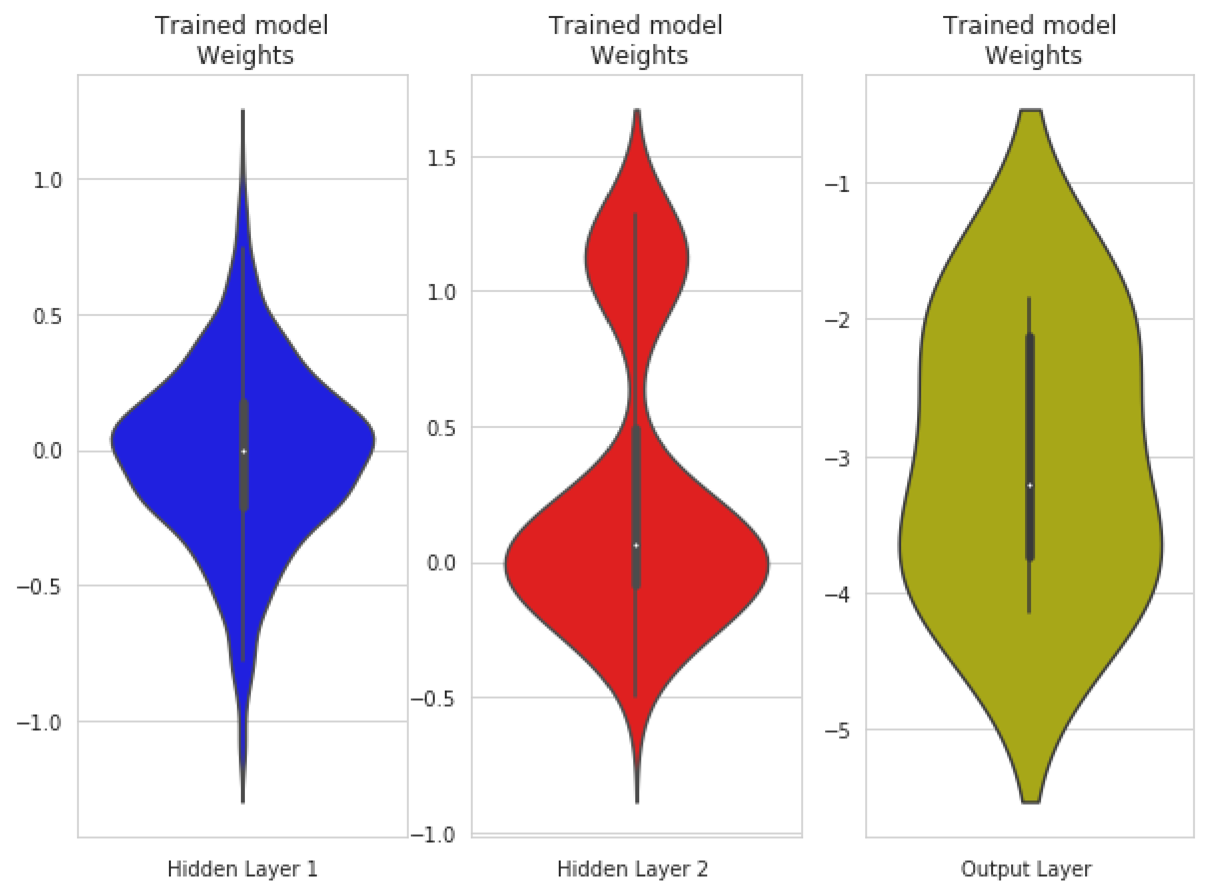
```
[[537   0   0   0   0   0]
 [  0 407  80   1   3   0]
 [  0  97 435   0   0   0]
 [  0   1   0 490   4   1]
 [  0   0   0   0 406  14]
 [  0   0   0  34   9 428]]
```

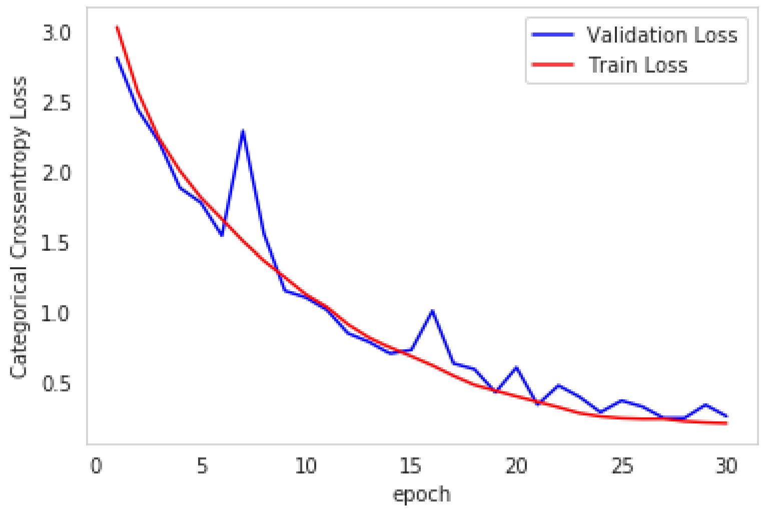



```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 537        0         0        0                   0
SITTING                  0      407        80        1                   3
STANDING                 0       97       435        0                   0
WALKING                  0        1         0      490                   4
WALKING_DOWNSTAIRS       0        0         0        0                 406
WALKING_UPSTAIRS         0        0         0       34                   9

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            0
STANDING                           0
WALKING                            1
WALKING_DOWNSTAIRS                14
WALKING_UPSTAIRS                 428
```

In [168]:

```
# Plot weight distribution using violin plot
import warnings
warnings.filterwarnings(action='ignore')
plot_weights(model)
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(train_2l_1,epochs)
```





6.8.2 Two layered LSTM model with 68,48 neurons and 0.75 drop-out

In [155]:
```
model_2l_2=lstm_model_2L(68,48,0.75)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_56 (LSTM)               (None, 128, 68)           21216     
_________________________________________________________________
batch_normalization_7 (Batch (None, 128, 68)           272       
_________________________________________________________________
dropout_42 (Dropout)         (None, 128, 68)           0         
_________________________________________________________________
lstm_57 (LSTM)               (None, 48)                22464     
_________________________________________________________________
dropout_43 (Dropout)         (None, 48)                0         
_________________________________________________________________
dense_33 (Dense)             (None, 6)                 294       
=================================================================
Total params: 44,246
Trainable params: 44,110
Non-trainable params: 136
_________________________________________________________________
```

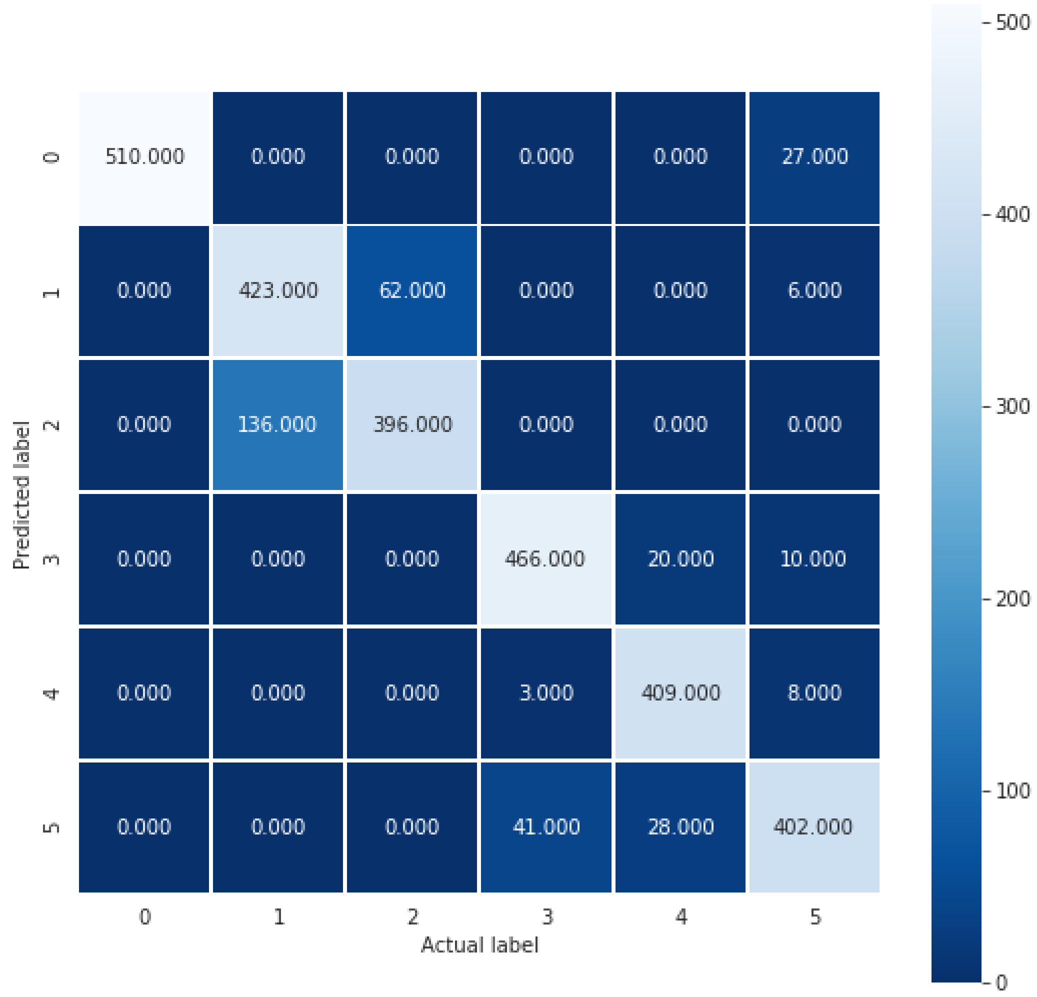
In [157]:
```
train_2l_2=train(X_train,Y_train,X_test, Y_test,128,30,model_2l_2)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 39s - loss: 3.7437 - acc: 0.4460 - val_loss: 3.4338 - val_acc: 0.4917
Epoch 2/30
 - 36s - loss: 3.2116 - acc: 0.5823 - val_loss: 3.2978 - val_acc: 0.4041
Epoch 3/30
 - 36s - loss: 2.8100 - acc: 0.6356 - val_loss: 2.5696 - val_acc: 0.6980
Epoch 4/30
 - 35s - loss: 2.5396 - acc: 0.6752 - val_loss: 2.3058 - val_acc: 0.7163
Epoch 5/30
 - 35s - loss: 2.2923 - acc: 0.7184 - val_loss: 2.1641 - val_acc: 0.7268
Epoch 6/30
 - 35s - loss: 2.0834 - acc: 0.7554 - val_loss: 1.8648 - val_acc: 0.7754
Epoch 7/30
 - 35s - loss: 1.8829 - acc: 0.7811 - val_loss: 1.8323 - val_acc: 0.7289
Epoch 8/30
 - 35s - loss: 1.7385 - acc: 0.7860 - val_loss: 1.7661 - val_acc: 0.7930
Epoch 9/30
 - 35s - loss: 1.5741 - acc: 0.8300 - val_loss: 1.3736 - val_acc: 0.8568
Epoch 10/30
 - 35s - loss: 1.4073 - acc: 0.8502 - val_loss: 1.2503 - val_acc: 0.8616
Epoch 11/30
 - 35s - loss: 1.2319 - acc: 0.8833 - val_loss: 1.0695 - val_acc: 0.8924
Epoch 12/30
 - 35s - loss: 1.0842 - acc: 0.9017 - val_loss: 1.4219 - val_acc: 0.7662
Epoch 13/30
 - 35s - loss: 1.0828 - acc: 0.8632 - val_loss: 1.2207 - val_acc: 0.7954
Epoch 14/30
 - 35s - loss: 0.9473 - acc: 0.8780 - val_loss: 0.9101 - val_acc: 0.8697
Epoch 15/30
 - 35s - loss: 0.8736 - acc: 0.8766 - val_loss: 1.1530 - val_acc: 0.7523
Epoch 16/30
 - 35s - loss: 0.7916 - acc: 0.8882 - val_loss: 0.8649 - val_acc: 0.8035
Epoch 17/30
 - 35s - loss: 0.6822 - acc: 0.8958 - val_loss: 0.7669 - val_acc: 0.8476
Epoch 18/30
 - 36s - loss: 0.5822 - acc: 0.9150 - val_loss: 0.8482 - val_acc: 0.8235
Epoch 19/30
 - 36s - loss: 0.5190 - acc: 0.9147 - val_loss: 0.5431 - val_acc: 0.8748
Epoch 20/30
 - 35s - loss: 0.4614 - acc: 0.9185 - val_loss: 0.4581 - val_acc: 0.8979
Epoch 21/30
 - 35s - loss: 0.4182 - acc: 0.9123 - val_loss: 0.5880 - val_acc: 0.8534
Epoch 22/30
 - 35s - loss: 0.3619 - acc: 0.9219 - val_loss: 0.4182 - val_acc: 0.8816
Epoch 23/30
 - 35s - loss: 0.3290 - acc: 0.9222 - val_loss: 0.5385 - val_acc: 0.8663
Epoch 24/30
 - 35s - loss: 0.3096 - acc: 0.9199 - val_loss: 0.6696 - val_acc: 0.8351
Epoch 25/30
 - 35s - loss: 0.2938 - acc: 0.9237 - val_loss: 0.4134 - val_acc: 0.8738
Epoch 26/30
 - 35s - loss: 0.2712 - acc: 0.9270 - val_loss: 0.5353 - val_acc: 0.8595
Epoch 27/30
 - 35s - loss: 0.2798 - acc: 0.9177 - val_loss: 0.4545 - val_acc: 0.8504
Epoch 28/30
 - 35s - loss: 0.3590 - acc: 0.9134 - val_loss: 0.7247 - val_acc: 0.8008
Epoch 29/30
 - 35s - loss: 0.2882 - acc: 0.9230 - val_loss: 0.3731 - val_acc: 0.8901
Epoch 30/30
 - 36s - loss: 0.2501 - acc: 0.9313 - val_loss: 0.3799 - val_acc: 0.8843
```

In [183]:

```
# Confusion Matrix
print(confusion_matrix(Y_test,model_2l_2.predict(X_test)))
```

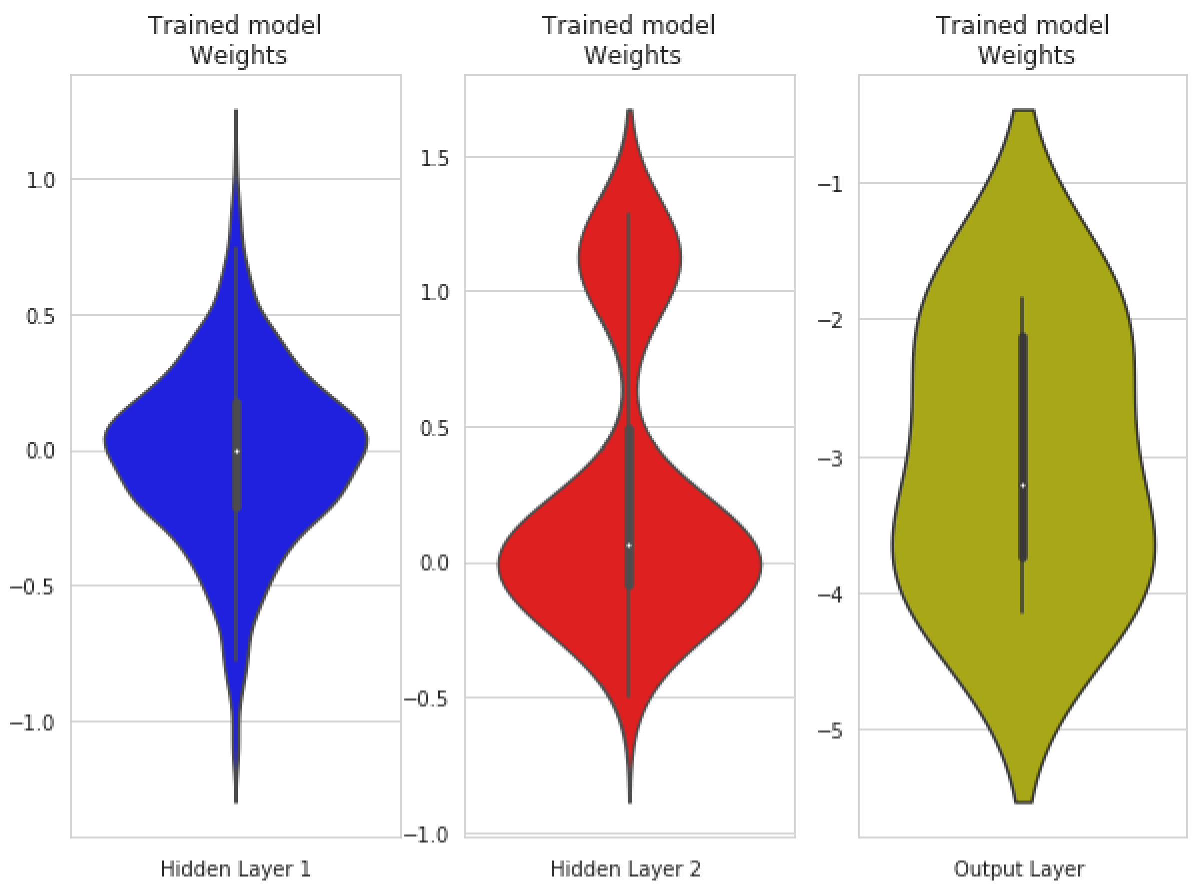
```
[[510   0   0   0   0  27]
 [  0 423  62   0   0   6]
 [  0 136 396   0   0   0]
 [  0   0   0 466  20  10]
 [  0   0   0   3 409   8]
 [  0   0   0  41  28 402]]
```

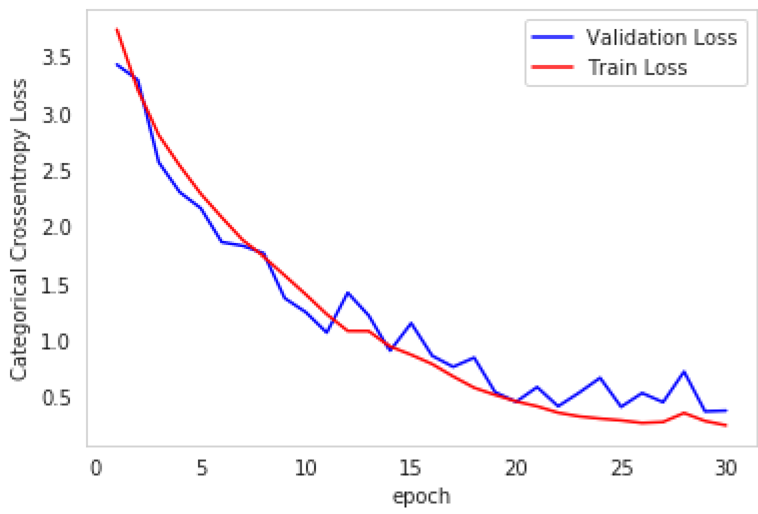



```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0         0        0                   0
SITTING                  0      423        62        0                   0
STANDING                 0      136       396        0                   0
WALKING                  0        0         0      466                  20
WALKING_DOWNSTAIRS       0        0         0        3                 409
WALKING_UPSTAIRS         0        0         0       41                  28

Pred                WALKING_UPSTAIRS
True
LAYING                            27
SITTING                            6
STANDING                           0
WALKING                           10
WALKING_DOWNSTAIRS                 8
WALKING_UPSTAIRS                 402
```

In [169]:

```
# Plot weight distribution using violin plot
import warnings
warnings.filterwarnings(action='ignore')
plot_weights(model)
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(train_2l_2,epochs)
```





6.8.3 Two layered LSTM model with 128,128 neurons and 0.85 drop-out

In [158]:
```
model_2l_3=lstm_model_2L(128,128,0.85)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_58 (LSTM)               (None, 128, 128)          70656     
_________________________________________________________________
batch_normalization_8 (Batch (None, 128, 128)          512       
_________________________________________________________________
dropout_44 (Dropout)         (None, 128, 128)          0         
_________________________________________________________________
lstm_59 (LSTM)               (None, 64)                49408     
_________________________________________________________________
dropout_45 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_34 (Dense)             (None, 6)                 390       
=================================================================
Total params: 120,966
Trainable params: 120,710
Non-trainable params: 256
_________________________________________________________________
```

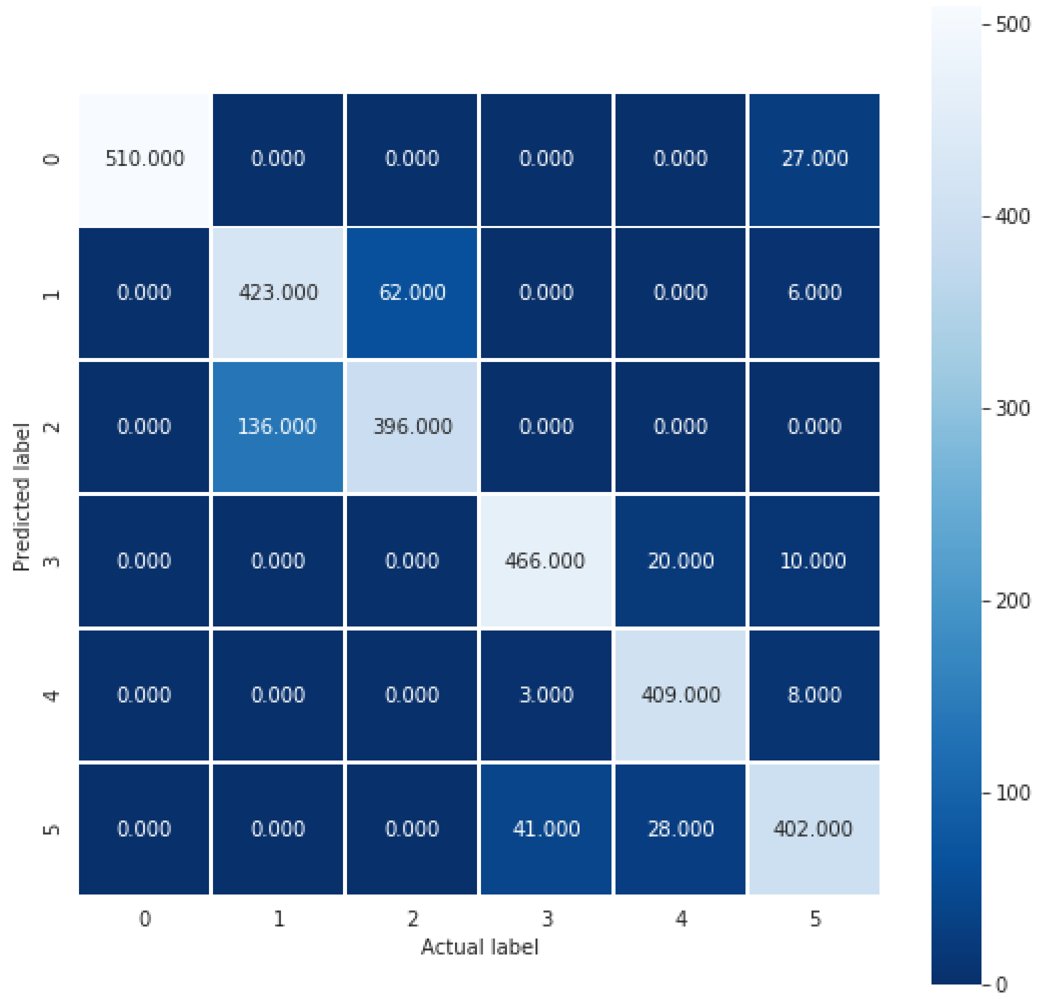
In [ ]:
```

```

In [160]:
```
train_2l_3=train(X_train,Y_train,X_test, Y_test,198,30,model_2l_3)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 71s - loss: 1.4969 - acc: 0.6559 - val_loss: 1.6184 - val_acc: 0.6403
Epoch 2/30
 - 72s - loss: 1.4084 - acc: 0.6575 - val_loss: 1.5060 - val_acc: 0.6417
Epoch 3/30
 - 72s - loss: 1.3269 - acc: 0.6642 - val_loss: 1.3017 - val_acc: 0.6719
Epoch 4/30
 - 72s - loss: 1.2506 - acc: 0.6640 - val_loss: 1.4520 - val_acc: 0.6308
Epoch 5/30
 - 73s - loss: 1.1677 - acc: 0.6593 - val_loss: 1.2104 - val_acc: 0.6420
Epoch 6/30
 - 74s - loss: 1.0977 - acc: 0.6649 - val_loss: 1.3235 - val_acc: 0.6420
Epoch 7/30
 - 73s - loss: 1.0288 - acc: 0.6732 - val_loss: 1.0335 - val_acc: 0.6593
Epoch 8/30
 - 74s - loss: 0.9526 - acc: 0.6772 - val_loss: 1.0337 - val_acc: 0.6566
Epoch 9/30
 - 74s - loss: 0.8877 - acc: 0.6870 - val_loss: 0.9647 - val_acc: 0.6440
Epoch 10/30
 - 73s - loss: 0.8366 - acc: 0.6770 - val_loss: 0.8706 - val_acc: 0.6770
Epoch 11/30
 - 73s - loss: 0.7675 - acc: 0.6880 - val_loss: 0.8022 - val_acc: 0.6478
Epoch 12/30
 - 73s - loss: 0.7212 - acc: 0.6896 - val_loss: 1.3917 - val_acc: 0.6210
Epoch 13/30
 - 73s - loss: 0.6640 - acc: 0.7038 - val_loss: 0.6284 - val_acc: 0.6882
Epoch 14/30
 - 73s - loss: 0.6438 - acc: 0.7074 - val_loss: 0.7523 - val_acc: 0.6719
Epoch 15/30
 - 73s - loss: 0.6266 - acc: 0.7175 - val_loss: 0.6535 - val_acc: 0.6912
Epoch 16/30
 - 74s - loss: 0.6169 - acc: 0.7214 - val_loss: 0.6166 - val_acc: 0.6725
Epoch 17/30
 - 73s - loss: 0.6090 - acc: 0.7303 - val_loss: 0.7029 - val_acc: 0.6793
Epoch 18/30
 - 73s - loss: 0.5948 - acc: 0.7391 - val_loss: 0.6560 - val_acc: 0.7072
Epoch 19/30
 - 74s - loss: 0.5814 - acc: 0.7462 - val_loss: 0.8362 - val_acc: 0.7988
Epoch 20/30
 - 73s - loss: 0.6133 - acc: 0.7515 - val_loss: 0.9580 - val_acc: 0.7618
Epoch 21/30
 - 73s - loss: 0.5794 - acc: 0.7696 - val_loss: 0.5727 - val_acc: 0.8534
Epoch 22/30
 - 76s - loss: 0.5343 - acc: 0.7912 - val_loss: 0.6593 - val_acc: 0.7954
Epoch 23/30
 - 76s - loss: 0.5266 - acc: 0.8025 - val_loss: 0.6217 - val_acc: 0.8113
Epoch 24/30
 - 76s - loss: 0.5181 - acc: 0.8150 - val_loss: 0.5580 - val_acc: 0.8741
Epoch 25/30
 - 77s - loss: 0.5002 - acc: 0.8364 - val_loss: 0.7744 - val_acc: 0.8419
Epoch 26/30
 - 77s - loss: 0.4999 - acc: 0.8232 - val_loss: 0.9647 - val_acc: 0.8083
Epoch 27/30
 - 76s - loss: 0.4958 - acc: 0.8225 - val_loss: 0.6814 - val_acc: 0.8269
Epoch 28/30
 - 76s - loss: 0.5140 - acc: 0.8300 - val_loss: 1.1205 - val_acc: 0.6749
Epoch 29/30
 - 76s - loss: 0.5945 - acc: 0.7958 - val_loss: 1.2269 - val_acc: 0.5348
Epoch 30/30
 - 77s - loss: 0.4990 - acc: 0.8283 - val_loss: 0.4161 - val_acc: 0.8592
```

In [184]:
```
# Confusion Matrix
print(confusion_matrix(Y_test,model_2l_2.predict(X_test)))
```

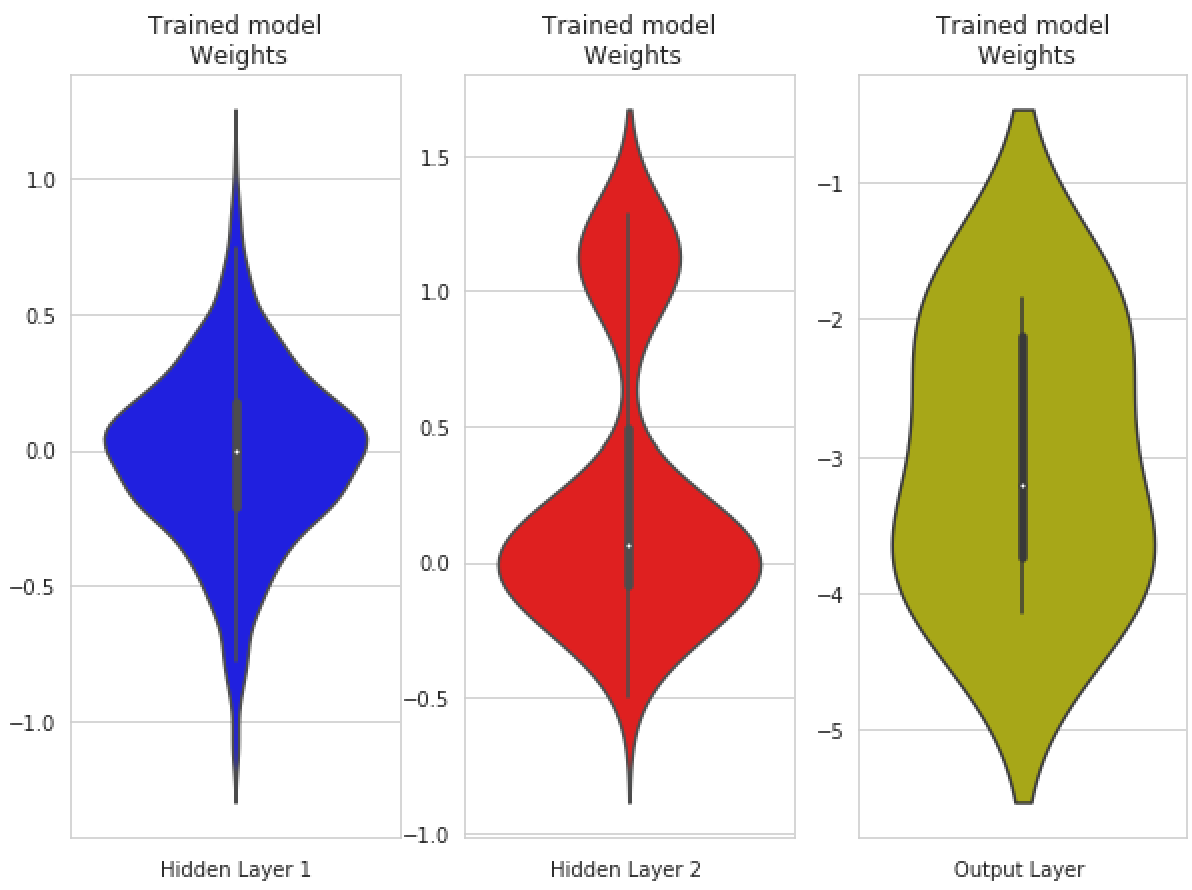
```
[[510   0   0   0   0  27]
 [  0 423  62   0   0   6]
 [  0 136 396   0   0   0]
 [  0   0   0 466  20  10]
 [  0   0   0   3 409   8]
 [  0   0   0  41  28 402]]
```

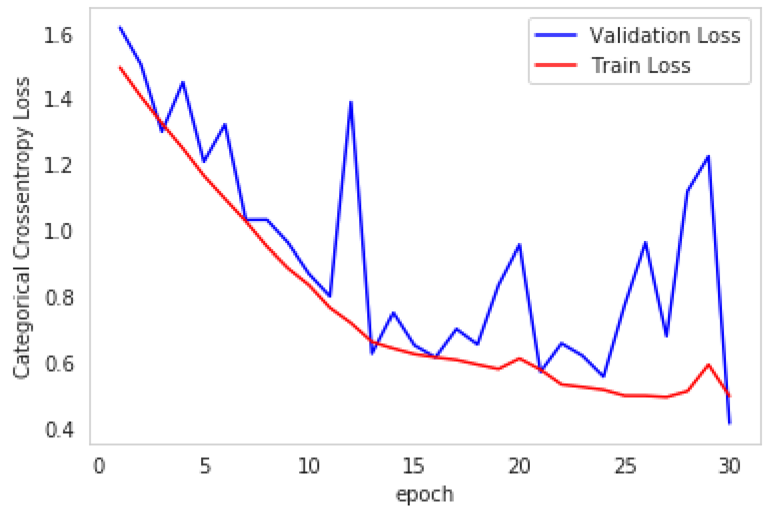



```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0         0        0                   0
SITTING                  0      423        62        0                   0
STANDING                 0      136       396        0                   0
WALKING                  0        0         0      466                  20
WALKING_DOWNSTAIRS       0        0         0        3                 409
WALKING_UPSTAIRS         0        0         0       41                  28

Pred                WALKING_UPSTAIRS
True
LAYING                            27
SITTING                            6
STANDING                           0
WALKING                           10
WALKING_DOWNSTAIRS                 8
WALKING_UPSTAIRS                 402
```

In [170]:

```python
# Plot weight distribution using violin plot
import warnings
warnings.filterwarnings(action='ignore')
plot_weights(model)
print()
print()

# Plot train and cross validation error
plot_train_cv_loss(train_2l_3,epochs)
```





Observations:-

- In the 2-layered LSTM architecture I have tried various combinations of neurons and drop-outs values and all gave good results with accuracy more than 90% except the last architecture.
- The 2-layered LSTM with (48,32) set of neurons and 0.6 drop-out rates have the best accuracy which is 93.70 with a error rate of 20%.
- The models with higher dropout rates tends to have higher error rates since the data is less.

In [5]:

```
def conclusion_table1():
    print()
    ptable=PrettyTable()
    ptable.title="The performance comparisons of all the Machine-learning algorithms are as follows: "
    ptable.field_names=["Algorithm","Accuracy","Error-rates"]
    ptable.add_row(["Logistic-regression",96.3,3.69])
    ptable.add_row(["Linear SVM",96.64,3.35])
    ptable.add_row(["RBF-Kernel SVM",96.27,3.73])
    ptable.add_row(["Decision Tree",86.49,13.51])
    ptable.add_row(["Random-forest",91.35,8.65])
    ptable.add_row(["Gradient-boosted Decsion tree",91.35,8.653])
    print(ptable)

def conclusion_table2():
    print()
    ptable=PrettyTable()
    ptable.title="The performance comparisons of all the single layered LSTM model are as follows: "
    ptable.field_names=["Number of neurons","Drop-out_rates","Percentage_Accuracy","Percentage_Loss"]
    ptable.add_row([48,0.6,91.53,30])
    ptable.add_row([48,0.75,65,67])
    ptable.add_row([64,0.6,93.61,19])
    ptable.add_row([64,0.75,91.95,25])
    ptable.add_row([128,0.6,94.41,14])
    ptable.add_row([128,0.75,93.40,20])

    Ptable=PrettyTable()
    Ptable.title="The performance comparisons of all the 2-Layered LSTM models are as follows: "
    Ptable.field_names=["Neuron in layer_1","Neuron in layer_2","Drop-out_rates","Percentage_Accuracy"
,"Percentage_Loss"]
    Ptable.add_row([48,32,0.6,93.70,20.9])
    Ptable.add_row([68,48,0.75,93.13,25.01])
    Ptable.add_row([128,64,0.85,82.83,49.90])
    print(ptable)
    print(Ptable)
```

In [7]:

```
conclusion_table1()
conclusion_table2()
```

The performance comparisons of all the Machine-learning algorithms are as follows:		
Algorithm	Accuracy	Error-rates
Logistic-regression	96.3	3.69
Linear SVM	96.64	3.35
RBF-Kernel SVM	96.27	3.73
Decision Tree	86.49	13.51
Random-forest	91.35	8.65
Gradient-boosted Decsion tree	91.35	8.653

The performance comparisons of all the single layered LSTM model are as follows:			
Number of neurons	Drop-out_rates	Percentage_Accuracy	Percentage_Loss
48	0.6	91.53	30
48	0.75	65	67
64	0.6	93.61	19
64	0.75	91.95	25
128	0.6	94.41	14
128	0.75	93.4	20

The performance comparisons of all the 2-Layered LSTM models are as follows:				
Neuron in layer_1	Neuron in layer_2	Drop-out_rates	Percentage_Accuracy	Percentage_Loss
48	32	0.6	93.7	20.9
68	48	0.75	93.13	25.01
128	64	0.85	82.83	49.9

Conclusion

- In this above case-study of Human-activity-recognition system the expert engineered features plays an important role for developing proper supervised machine-learning models with greater accuracy and lesser error values.
- So in machine-learning domain knowledge cannot be completely ignored as it is a key tool in solving complex real life problems but it is a slow and tedious process.
- To fast up the process the deep-learning models are used and especially in sequenced data as input LSTM models are used widely.These models can automatically learn the features as raw input is given to it.
- In this case-study raw input vectors which are prepared by using the over-lapped sampling techniques over the 9-signal data.
- No expert features are given as input and the LSTM model learned the features over time and gave pretty descent accuracy and error-rates.
- After tuning the hyper-parameters of a single layered LSTM I got an accuracy around 94.41% which is pretty close to the accuracy of the machine-learning model which was build using the expert-engineered features.
- So clearly with larger dataset and deeper layers the Deep-Learning model can surpass the performance of the machine-learning models.
- It is very easy to over-fitt an deep-learning model so to avoid it proper regularization is neccessary which can be done by Drop-out layers and regulating the layers in a Deep-learning model.

In [ ]: