In [1]:

```
# Importing Libraries
```

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
# 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)])

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

Data

In [3]:

```
# Data directory
DATADIR = 'UCI_HAR_Dataset'
```

In [4]:

```
# 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"
]
```

In [5]:

```
# 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))

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

    return X_train, X_test, y_train, y_test
```

In [6]:

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

In [7]:

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

In [8]:

```
# 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 [86]:

```
# Importing libraries
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout
```

In [87]:

```
# Initializing parameters
epochs = 30
batch_size = 16
n_hidden = 32
```

In [9]:

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

In [10]:

```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
print("X train shape :",X_train.shape)
print("X test shape :",X_test.shape)
print("Y train shape :",Y_train.shape)
print("Y train shape :",Y_test.shape)
```

```
C:\Users\patha\Anaconda3\lib\site-packages\ipykernel_launcher.py:12: FutureWarning: Method
.as_matrix will be removed in a future version. Use .values instead.
  if sys.path[0] == '':
```

```
X train shape : (7352, 128, 9)
X test shape : (2947, 128, 9)
Y train shape : (7352, 6)
Y train shape : (2947, 6)
```

```
C:\Users\patha\Anaconda3\lib\site-packages\ipykernel_launcher.py:30: FutureWarning: Method
.as_matrix will be removed in a future version. Use .values instead.
```

In [11]:

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

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

```
128
9
7352
```

- Defining the Architecture of LSTM

In [91]:

```
# 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_3 (LSTM)                (None, 32)                5376
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 198
=================================================================
Total params: 5,574
Trainable params: 5,574
Non-trainable params: 0
_________________________________________________________________
```

In [22]:

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

In [23]:

```
# Training the model
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 [==============================] - 92s 13ms/step - loss: 1.3018 - acc: 0.4395 - val_loss
: 1.1254 - val_acc: 0.4662
Epoch 2/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.9666 - acc: 0.5880 - val_loss
: 0.9491 - val_acc: 0.5714
Epoch 3/30
7352/7352 [==============================] - 97s 13ms/step - loss: 0.7812 - acc: 0.6408 - val_loss
: 0.8286 - val_acc: 0.5850
Epoch 4/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.6941 - acc: 0.6574 - val_loss
: 0.7297 - val_acc: 0.6128
Epoch 5/30
7352/7352 [==============================] - 92s 13ms/step - loss: 0.6336 - acc: 0.6912 - val_loss
: 0.7359 - val_acc: 0.6787
Epoch 6/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.5859 - acc: 0.7134 - val_loss
: 0.7015 - val_acc: 0.6939
Epoch 7/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.5692 - acc: 0.7477 - val_loss
: 0.5995 - val_acc: 0.7387
Epoch 8/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.4899 - acc: 0.7809 - val_loss
: 0.5762 - val_acc: 0.7387
Epoch 9/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.4482 - acc: 0.7886 - val_loss
: 0.7413 - val_acc: 0.7126
Epoch 10/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.4132 - acc: 0.8077 - val_loss
: 0.5048 - val_acc: 0.7513
Epoch 11/30
7352/7352 [==============================] - 89s 12ms/step - loss: 0.3985 - acc: 0.8274 - val_loss
: 0.5234 - val_acc: 0.7452
Epoch 12/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.3378 - acc: 0.8638 - val_loss
: 0.4114 - val_acc: 0.8833
Epoch 13/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.2947 - acc: 0.9051 - val_loss
: 0.4386 - val_acc: 0.8731
Epoch 14/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.2448 - acc: 0.9291 - val_loss
: 0.3768 - val_acc: 0.8921
Epoch 15/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.2157 - acc: 0.9331 - val_loss
: 0.4441 - val_acc: 0.8931
Epoch 16/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.2053 - acc: 0.9366 - val_loss
: 0.4162 - val_acc: 0.8968
Epoch 17/30
7352/7352 [==============================] - 89s 12ms/step - loss: 0.2028 - acc: 0.9404 - val_loss
: 0.4538 - val_acc: 0.8962
Epoch 18/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.1911 - acc: 0.9419 - val_loss
: 0.3964 - val_acc: 0.8999
Epoch 19/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1912 - acc: 0.9407 - val_loss
: 0.3165 - val_acc: 0.9030
Epoch 20/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1732 - acc: 0.9446 - val_loss
: 0.4546 - val_acc: 0.8904
```

```
Epoch 21/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1782 - acc: 0.9444 - val_loss
: 0.3346 - val_acc: 0.9063
Epoch 22/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1812 - acc: 0.9418 - val_loss
: 0.8164 - val_acc: 0.8582
Epoch 23/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1824 - acc: 0.9426 - val_loss
: 0.4240 - val_acc: 0.9036
Epoch 24/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1726 - acc: 0.9429 - val_loss
: 0.4067 - val_acc: 0.9148
Epoch 25/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1737 - acc: 0.9411 - val_loss
: 0.3396 - val_acc: 0.9074
Epoch 26/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1650 - acc: 0.9461 - val_loss
: 0.3806 - val_acc: 0.9019
Epoch 27/30
7352/7352 [==============================] - 89s 12ms/step - loss: 0.1925 - acc: 0.9415 - val_loss
: 0.6464 - val_acc: 0.8850
Epoch 28/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.1965 - acc: 0.9425 - val_loss
: 0.3363 - val_acc: 0.9203
Epoch 29/30
7352/7352 [==============================] - 92s 12ms/step - loss: 0.1889 - acc: 0.9431 - val_loss
: 0.3737 - val_acc: 0.9158
Epoch 30/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1945 - acc: 0.9414 - val_loss
: 0.3088 - val_acc: 0.9097
```

Out[23]:

```
<keras.callbacks.History at 0x29b5ee36a20>
```

In [24]:

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

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 512        0        25        0                   0
SITTING                  3      410        75        0                   0
STANDING                 0       87       445        0                   0
WALKING                  0        0         0      481                   2
WALKING_DOWNSTAIRS       0        0         0        0                 382
WALKING_UPSTAIRS         0        0         0        2                  18

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            3
STANDING                           0
WALKING                           13
WALKING_DOWNSTAIRS                38
WALKING_UPSTAIRS                 451
```

In [27]:

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

```
2947/2947 [==============================] - 4s 2ms/step
```

In [28]:

```
score
```

Out[28]:

```
[0.3087582236972612, 0.9097387173396675]
```

- With a simple 2 layer architecture we got 90.09% accuracy and a loss of 0.30
- We can further imporve the performace with Hyperparameter tuning

Assignment

2 LSTM Layers

In [282]:

```
# Importing libraries
from keras.models import Sequential
from keras.layers import
LSTM,Dense,Dropout,Activation,BatchNormalization,Conv2D,Flatten,TimeDistributed,Conv1D
from keras.regularizers import *
from keras.callbacks import LearningRateScheduler,TerminateOnNaN,EarlyStopping
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.initializers import VarianceScaling
from keras.optimizers import *
```

In [283]:

```
import math
def lr_decay(epoch):
    return float(0.0001 * math.pow(0.6, math.floor((1+epoch)/10)))
lr = LearningRateScheduler(lr_decay)
tm = TerminateOnNaN()
es = EarlyStopping(monitor = 'accuracy')
init = VarianceScaling(scale = 1.0,mode = 'fan_avg',distribution = 'normal')
adam = Adam(lr=0.001)
rmsprop = RMSprop(lr = 0.001)
```

In [62]:

```
model = Sequential()
model.add(LSTM(32,activation = 'relu',return_sequences=True, input_shape=(timesteps, input_dim),rec
urrent_initializer="glorot_uniform",recurrent_regularizer=l2(0.003) ))
model.add(BatchNormalization())
model.add(Dropout(0.6))
model.add(LSTM(32,recurrent_initializer="glorot_uniform",recurrent_regularizer=l2(0.003)))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Dense(6, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_24 (LSTM)               (None, 128, 32)           5376
_________________________________________________________________
batch_normalization_22 (Batc (None, 128, 32)           128
_________________________________________________________________
dropout_22 (Dropout)         (None, 128, 32)           0
_________________________________________________________________
lstm_25 (LSTM)               (None, 32)                8320
_________________________________________________________________
batch_normalization_23 (Batc (None, 32)                128
_________________________________________________________________
dropout_23 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_12 (Dense)             (None, 6)                 198
=================================================================
Total params: 14,150
Trainable params: 14,022
Non-trainable params: 128
_________________________________________________________________
```

In [63]:

```
%%time
model.compile(loss='categorical_crossentropy',
          optimizer=adam,
          metrics=['accuracy'])
```

```
Wall time: 31.9 ms
```

In [64]:

```
%%time
# Training the model
result = model.fit(X_train,
          Y_train,
          batch_size=32,
          validation_data=(X_test, Y_test),callbacks=[lr,tm],
          epochs=30,verbose = 1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 67s 9ms/step - loss: 1.3829 - accuracy: 0.5645 - val_
loss: 1.1387 - val_accuracy: 0.5826
Epoch 2/30
7352/7352 [==============================] - 70s 10ms/step - loss: 0.9055 - accuracy: 0.6557 - val
_loss: 0.8158 - val_accuracy: 0.5864
Epoch 3/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.7707 - accuracy: 0.6740 - val_
loss: 0.8092 - val_accuracy: 0.6583
Epoch 4/30
7352/7352 [==============================] - 68s 9ms/step - loss: 0.6547 - accuracy: 0.7334 - val_
loss: 1.1115 - val_accuracy: 0.6464
Epoch 5/30
7352/7352 [==============================] - 67s 9ms/step - loss: 0.4773 - accuracy: 0.8672 - val_
loss: 1.3450 - val_accuracy: 0.6844
Epoch 6/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.3395 - accuracy: 0.9142 - val_
loss: 2.3442 - val_accuracy: 0.5894
Epoch 7/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.2963 - accuracy: 0.9221 - val_
loss: 1.2052 - val_accuracy: 0.7109
Epoch 8/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.2843 - accuracy: 0.9259 - val_
loss: 0.8428 - val_accuracy: 0.7859
Epoch 9/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.2381 - accuracy: 0.9399 - val_
loss: 0.9233 - val_accuracy: 0.7896
Epoch 10/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.2241 - accuracy: 0.9385 - val_
loss: 0.8582 - val_accuracy: 0.8168
Epoch 11/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.2098 - accuracy: 0.9436 - val_
loss: 1.4082 - val_accuracy: 0.7631
Epoch 12/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.2018 - accuracy: 0.9446 - val_
loss: 0.9004 - val_accuracy: 0.8154
Epoch 13/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.2038 - accuracy: 0.9403 - val_
loss: 1.3356 - val_accuracy: 0.7730
Epoch 14/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.2054 - accuracy: 0.9415 - val_
loss: 1.2279 - val_accuracy: 0.7560
Epoch 15/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.2251 - accuracy: 0.9373 - val_
loss: 0.7518 - val_accuracy: 0.8242
Epoch 16/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.2005 - accuracy: 0.9382 - val_
loss: 0.6747 - val_accuracy: 0.8578
Epoch 17/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.1937 - accuracy: 0.9467 - val_
loss: 0.4177 - val_accuracy: 0.8904
Epoch 18/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.1917 - accuracy: 0.9397 - val_
loss: 0.4571 - val_accuracy: 0.9043
Epoch 19/30
```

```
7352/7352 [==============================] - 64s 9ms/step - loss: 0.1866 - accuracy: 0.9441 - val_
loss: 0.3585 - val_accuracy: 0.9104
Epoch 20/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.1792 - accuracy: 0.9460 - val_
loss: 0.3843 - val_accuracy: 0.9084
Epoch 21/30
7352/7352 [==============================] - 66s 9ms/step - loss: 0.1835 - accuracy: 0.9482 - val_
loss: 0.3312 - val_accuracy: 0.9128
Epoch 22/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.1753 - accuracy: 0.9465 - val_
loss: 0.3680 - val_accuracy: 0.9121
Epoch 23/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.1657 - accuracy: 0.9484 - val_
loss: 0.4147 - val_accuracy: 0.9070
Epoch 24/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.1698 - accuracy: 0.9479 - val_
loss: 0.3551 - val_accuracy: 0.9087
Epoch 25/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.1683 - accuracy: 0.9510 - val_
loss: 0.3739 - val_accuracy: 0.9114
Epoch 26/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.1646 - accuracy: 0.9517 - val_
loss: 0.3635 - val_accuracy: 0.9070
Epoch 27/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.1615 - accuracy: 0.9506 - val_
loss: 0.4292 - val_accuracy: 0.9040
Epoch 28/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.1548 - accuracy: 0.9516 - val_
loss: 0.4087 - val_accuracy: 0.9108
Epoch 29/30
7352/7352 [==============================] - 66s 9ms/step - loss: 0.1549 - accuracy: 0.9491 - val_
loss: 0.4016 - val_accuracy: 0.9135
Epoch 30/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.1513 - accuracy: 0.9508 - val_
loss: 0.3972 - val_accuracy: 0.9179
Wall time: 32min 29s
```

In [65]:

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

```
2947/2947 [==============================] - 11s 4ms/step
```

Out[65]:

```
[0.3971564822520872, 0.9178826212882996]
```

In [66]:

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

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 537        0         0        0                   0
SITTING                  0      408        78        0                   0
STANDING                 0       91       441        0                   0
WALKING                  0        0         0      462                  33
WALKING_DOWNSTAIRS       0        0         0        1                 418
WALKING_UPSTAIRS         0        0         1        8                  23

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            5
STANDING                           0
WALKING                            1
WALKING_DOWNSTAIRS                 1
WALKING_UPSTAIRS                 439
```

3 LSTM Layers

In [67]:

```
model = Sequential()
model.add(LSTM(16,activation = 'relu',return_sequences=True, input_shape=(timesteps, input_dim),rec
urrent_initializer="glorot_uniform",recurrent_regularizer=l2(0.003) ))
model.add(BatchNormalization())
model.add(Dropout(0.6))
model.add(LSTM(16,return_sequences=True,recurrent_initializer="glorot_uniform",recurrent_regularize
r=l2(0.003)))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(LSTM(16,recurrent_initializer="glorot_uniform",recurrent_regularizer=l2(0.003)))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Dense(6, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_26 (LSTM)               (None, 128, 16)           1664      
_________________________________________________________________
batch_normalization_24 (Batc (None, 128, 16)           64        
_________________________________________________________________
dropout_24 (Dropout)         (None, 128, 16)           0         
_________________________________________________________________
lstm_27 (LSTM)               (None, 128, 16)           2112      
_________________________________________________________________
batch_normalization_25 (Batc (None, 128, 16)           64        
_________________________________________________________________
dropout_25 (Dropout)         (None, 128, 16)           0         
_________________________________________________________________
lstm_28 (LSTM)               (None, 16)                2112      
_________________________________________________________________
batch_normalization_26 (Batc (None, 16)                64        
_________________________________________________________________
dropout_26 (Dropout)         (None, 16)                0         
_________________________________________________________________
dense_13 (Dense)             (None, 6)                 102       
=================================================================
Total params: 6,182
Trainable params: 6,086
Non-trainable params: 96
_________________________________________________________________
```

In [68]:

```
%%time
model.compile(loss='categorical_crossentropy',
          optimizer=adam,
          metrics=['accuracy'])
```

```
Wall time: 46.9 ms
```

In [69]:

```
%%time
# Training the model
result = model.fit(X_train,
          Y_train,
          batch_size=32,
          validation_data=(X_test, Y_test),callbacks=[lr,tm],
          epochs=30,verbose = 1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 96s 13ms/step - loss: 1.0784 - accuracy: 0.6155 - val
_loss: 2.4525 - val_accuracy: 0.2104
Epoch 2/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.8143 - accuracy: 0.6495 - val
_loss: 1.8998 - val_accuracy: 0.3624
```

```
Epoch 3/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.7552 - accuracy: 0.6600 - val
_loss: 0.7603 - val_accuracy: 0.6135
Epoch 4/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.7052 - accuracy: 0.6668 - val
_loss: 0.8918 - val_accuracy: 0.5836
Epoch 5/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.6817 - accuracy: 0.6634 - val
_loss: 0.8573 - val_accuracy: 0.5813
Epoch 6/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.6373 - accuracy: 0.6685 - val
_loss: 2.4977 - val_accuracy: 0.3264
Epoch 7/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.5997 - accuracy: 0.6759 - val
_loss: 2.8237 - val_accuracy: 0.3444
Epoch 8/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.6100 - accuracy: 0.6861 - val
_loss: 2.2098 - val_accuracy: 0.3963
Epoch 9/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.5890 - accuracy: 0.6989 - val
_loss: 1.3614 - val_accuracy: 0.6179
Epoch 10/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.5710 - accuracy: 0.7180 - val
_loss: 1.0003 - val_accuracy: 0.6437
Epoch 11/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.5547 - accuracy: 0.7539 - val
_loss: 2.3520 - val_accuracy: 0.4041
Epoch 12/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.5145 - accuracy: 0.7893 - val
_loss: 1.3954 - val_accuracy: 0.5490
Epoch 13/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.4829 - accuracy: 0.7888 - val
_loss: 2.1027 - val_accuracy: 0.4744
Epoch 14/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.4618 - accuracy: 0.7983 - val
_loss: 0.9134 - val_accuracy: 0.6634
Epoch 15/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.4855 - accuracy: 0.7892 - val
_loss: 1.2600 - val_accuracy: 0.5959
Epoch 16/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.4586 - accuracy: 0.8142 - val
_loss: 0.8715 - val_accuracy: 0.7292
Epoch 17/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.4255 - accuracy: 0.8254 - val
_loss: 0.9037 - val_accuracy: 0.7557
Epoch 18/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.4076 - accuracy: 0.8424 - val
_loss: 0.7246 - val_accuracy: 0.7682
Epoch 19/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.3881 - accuracy: 0.8630 - val
_loss: 1.1181 - val_accuracy: 0.7526
Epoch 20/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.3557 - accuracy: 0.8886 - val
_loss: 0.7369 - val_accuracy: 0.8147
Epoch 21/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.3532 - accuracy: 0.8912 - val
_loss: 0.6189 - val_accuracy: 0.8239
Epoch 22/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.3318 - accuracy: 0.9025 - val
_loss: 0.6092 - val_accuracy: 0.8317
Epoch 23/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.3031 - accuracy: 0.9101 - val
_loss: 0.5088 - val_accuracy: 0.8738
Epoch 24/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.2878 - accuracy: 0.9146 - val
_loss: 0.4594 - val_accuracy: 0.8633
Epoch 25/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.2818 - accuracy: 0.9151 - val
_loss: 0.3919 - val_accuracy: 0.8853
Epoch 26/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.2665 - accuracy: 0.9222 - val
_loss: 0.3856 - val_accuracy: 0.8907
Epoch 27/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.2552 - accuracy: 0.9267 - val
_loss: 0.4395 - val_accuracy: 0.8758
Epoch 28/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.2483 - accuracy: 0.9283 - val
```

```
_loss: 0.4254 - val_accuracy: 0.9040
Epoch 29/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.2474 - accuracy: 0.9257 - val
_loss: 0.4692 - val_accuracy: 0.9023
Epoch 30/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.2402 - accuracy: 0.9300 - val
_loss: 0.4864 - val_accuracy: 0.8951
Wall time: 46min 55s
```

In [70]:

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

```
2947/2947 [==============================] - 15s 5ms/step
```

Out[70]:

```
[0.48638685676640564, 0.8951476216316223]
```

In [71]:

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

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0         0        0                   0
SITTING                  5      415        64        1                   0
STANDING                 0      104       427        0                   0
WALKING                  0        0         0      493                   1
WALKING_DOWNSTAIRS       0        0         0       54                 366
WALKING_UPSTAIRS         0        2         0       31                  11

Pred                WALKING_UPSTAIRS
True
LAYING                            27
SITTING                            6
STANDING                           1
WALKING                            2
WALKING_DOWNSTAIRS                 0
WALKING_UPSTAIRS                 427
```

In [ ]:

```

```

In [365]:

```
model = Sequential()
model.add(LSTM(100,activation = 'relu',return_sequences=True, input_shape=(timesteps, input_dim),re
current_initializer="glorot_uniform",recurrent_regularizer=l2(0.003) ))
model.add(BatchNormalization())
model.add(Dropout(0.8))
model.add(LSTM(100,recurrent_initializer="glorot_uniform",recurrent_regularizer=l2(0.0003)))
model.add(BatchNormalization())
model.add(Dropout(0.8))
#
model.add(LSTM(28,return_sequences=True,recurrent_initializer="glorot_uniform",recurrent_regulariz
(0.0003)))
# model.add(BatchNormalization())
# model.add(Dropout(0.8))
# #
model.add(LSTM(64,return_sequences=True,recurrent_initializer="glorot_uniform",recurrent_regulariz
(0.003)))
# # model.add(BatchNormalization())
# # model.add(Dropout(0.6))
# model.add(LSTM(16,recurrent_initializer="glorot_uniform",recurrent_regularizer=l2(0.0003)))
# model.add(BatchNormalization())
# model.add(Dropout(0.8))
model.add(Dense(n_classes, activation='softmax'))
model.summary()
```

```
Model: "sequential_41"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_36 (LSTM)               (None, 128, 100)          44000     
_________________________________________________________________
batch_normalization_69 (Batc (None, 128, 100)          400       
_________________________________________________________________
dropout_130 (Dropout)        (None, 128, 100)          0         
_________________________________________________________________
lstm_37 (LSTM)               (None, 100)               80400     
_________________________________________________________________
batch_normalization_70 (Batc (None, 100)               400       
_________________________________________________________________
dropout_131 (Dropout)        (None, 100)               0         
_________________________________________________________________
dense_70 (Dense)             (None, 6)                 606       
=================================================================
Total params: 125,806
Trainable params: 125,406
Non-trainable params: 400
_________________________________________________________________
```

In [366]:

```
%%time
model.compile(loss='categorical_crossentropy',
          optimizer=adam,
          metrics=['accuracy'])
```

```
Wall time: 47.9 ms
```

In [367]:

```
%%time
# Training the model
result = model.fit(X_train,
          Y_train,
          batch_size=32,
          validation_data=(X_test, Y_test),callbacks=[lr,tm],
          epochs=50,verbose = 1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/50
7352/7352 [==============================] - 71s 10ms/step - loss: 1.9772 - accuracy: 0.5590 - val
_loss: 1.1925 - val_accuracy: 0.6155
Epoch 2/50
7352/7352 [==============================] - 63s 9ms/step - loss: 1.3167 - accuracy: 0.6606 - val_
loss: 1.1295 - val_accuracy: 0.6603
Epoch 3/50
7352/7352 [==============================] - 64s 9ms/step - loss: 1.1459 - accuracy: 0.6821 - val_
loss: 1.0407 - val_accuracy: 0.7380
Epoch 4/50
7352/7352 [==============================] - 66s 9ms/step - loss: 1.0348 - accuracy: 0.7121 - val_
loss: 0.9470 - val_accuracy: 0.7513
Epoch 5/50
7352/7352 [==============================] - 69s 9ms/step - loss: 0.9272 - accuracy: 0.7422 - val_
loss: 0.9104 - val_accuracy: 0.7638
Epoch 6/50
7352/7352 [==============================] - 65s 9ms/step - loss: 0.8319 - accuracy: 0.7705 - val_
loss: 0.9305 - val_accuracy: 0.7760
Epoch 7/50
7352/7352 [==============================] - 63s 9ms/step - loss: 0.7345 - accuracy: 0.8040 - val_
loss: 0.8735 - val_accuracy: 0.8029
Epoch 8/50
7352/7352 [==============================] - 65s 9ms/step - loss: 0.6537 - accuracy: 0.8298 - val_
loss: 0.8620 - val_accuracy: 0.8174
Epoch 9/50
7352/7352 [==============================] - 65s 9ms/step - loss: 0.5461 - accuracy: 0.8700 - val_
loss: 0.8959 - val_accuracy: 0.8144
Epoch 10/50
7352/7352 [==============================] - 68s 9ms/step - loss: 0.5148 - accuracy: 0.8886 - val_
```

```
loss: 0.7962 - val_accuracy: 0.8439
Epoch 11/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.4713 - accuracy: 0.8976 - val_
loss: 0.7703 - val_accuracy: 0.8585
Epoch 12/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.4485 - accuracy: 0.9038 - val_
loss: 0.7946 - val_accuracy: 0.8585
Epoch 13/50
7352/7352 [==============================] - 69s 9ms/step - loss: 0.4188 - accuracy: 0.9138 - val_
loss: 0.7353 - val_accuracy: 0.8680
Epoch 14/50
7352/7352 [==============================] - 68s 9ms/step - loss: 0.4218 - accuracy: 0.9128 - val_
loss: 0.6979 - val_accuracy: 0.8744
Epoch 15/50
7352/7352 [==============================] - 73s 10ms/step - loss: 0.4035 - accuracy: 0.9193 - val
_loss: 0.7285 - val_accuracy: 0.8697
Epoch 16/50
7352/7352 [==============================] - 69s 9ms/step - loss: 0.3860 - accuracy: 0.9229 - val_
loss: 0.7551 - val_accuracy: 0.8687
Epoch 17/50
7352/7352 [==============================] - 75s 10ms/step - loss: 0.3784 - accuracy: 0.9232 - val
_loss: 0.6416 - val_accuracy: 0.8839
Epoch 18/50
7352/7352 [==============================] - 90s 12ms/step - loss: 0.3696 - accuracy: 0.9286 - val
_loss: 0.7278 - val_accuracy: 0.8755
Epoch 19/50
7352/7352 [==============================] - 86s 12ms/step - loss: 0.3651 - accuracy: 0.9278 - val
_loss: 0.7883 - val_accuracy: 0.8673
Epoch 20/50
7352/7352 [==============================] - 78s 11ms/step - loss: 0.3522 - accuracy: 0.9336 - val
_loss: 0.7850 - val_accuracy: 0.8666
Epoch 21/50
7352/7352 [==============================] - 72s 10ms/step - loss: 0.3471 - accuracy: 0.9331 - val
_loss: 0.6614 - val_accuracy: 0.8914
Epoch 22/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.3451 - accuracy: 0.9332 - val_
loss: 0.7449 - val_accuracy: 0.8724
Epoch 23/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.3378 - accuracy: 0.9346 - val_
loss: 0.7535 - val_accuracy: 0.8789
Epoch 24/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.3488 - accuracy: 0.9342 - val_
loss: 0.6762 - val_accuracy: 0.8839
Epoch 25/50
7352/7352 [==============================] - 74s 10ms/step - loss: 0.3296 - accuracy: 0.9396 - val
_loss: 0.6535 - val_accuracy: 0.8877
Epoch 26/50
7352/7352 [==============================] - 71s 10ms/step - loss: 0.3326 - accuracy: 0.9332 - val
_loss: 0.6109 - val_accuracy: 0.8948
Epoch 27/50
7352/7352 [==============================] - 70s 10ms/step - loss: 0.3321 - accuracy: 0.9355 - val
_loss: 0.6816 - val_accuracy: 0.8863
Epoch 28/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.3348 - accuracy: 0.9351 - val_
loss: 0.7173 - val_accuracy: 0.8867
Epoch 29/50
7352/7352 [==============================] - 68s 9ms/step - loss: 0.3132 - accuracy: 0.9400 - val_
loss: 0.6871 - val_accuracy: 0.8826
Epoch 30/50
3936/7352 [===============>..............] - ETA: 26s - loss: 0.3062 - accuracy: 0.9433

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<timed exec> in <module>

~\Anaconda3\lib\site-packages\keras\engine\training.py in fit(self, x, y, batch_size, epochs,
verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight,
initial_epoch, steps_per_epoch, validation_steps, validation_freq, max_queue_size, workers,
use_multiprocessing, **kwargs)
   1237                                         steps_per_epoch=steps_per_epoch,
   1238                                         validation_steps=validation_steps,
-> 1239                                         validation_freq=validation_freq)
   1240
   1241     def evaluate(self,

~\Anaconda3\lib\site-packages\keras\engine\training_arrays.py in fit_loop(model, fit_function,
fit_inputs, out_labels, batch_size, epochs, verbose, callbacks, val_function, val_inputs, shuffle,
```

```
initial_epoch, steps_per_epoch, validation_steps, validation_freq)
    194                         ins_batch[i] = ins_batch[i].toarray()
    195 
--> 196                     outs = fit_function(ins_batch)
    197                     outs = to_list(outs)
    198                     for l, o in zip(out_labels, outs):

~\Anaconda3\lib\site-packages\tensorflow\python\keras\backend.py in __call__(self, inputs)
   3290 
   3291     fetched = self._callable_fn(*array_vals,
-> 3292                                 run_metadata=self.run_metadata)
   3293     self._call_fetch_callbacks(fetched[-len(self._fetches):])
   3294     output_structure = nest.pack_sequence_as(

~\Anaconda3\lib\site-packages\tensorflow\python\client\session.py in __call__(self, *args,
**kwargs)
   1456         ret = tf_session.TF_SessionRunCallable(self._session._session,
   1457                                                self._handle, args,
-> 1458                                                run_metadata_ptr)
   1459         if run_metadata:
   1460           proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)

KeyboardInterrupt:
```

In [ ]:

In [ ]:

In [ ]:

Using CNN

In [12]:

```python
# Importing libraries
from keras.models import Sequential
from keras.layers import
LSTM,Dense,Dropout,Activation,BatchNormalization,Conv2D,Flatten,TimeDistributed,Conv1D,MaxPool1D
from keras.regularizers import *
from keras.callbacks import LearningRateScheduler,TerminateOnNaN
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.initializers import VarianceScaling
from keras.optimizers import *
```

In [13]:

```python
import math
def lr_decay(epoch):
    return float(0.001 * math.pow(0.6, math.floor((1+epoch)/10)))
lr = LearningRateScheduler(lr_decay)
tm = TerminateOnNaN()
init = VarianceScaling(scale = 1.0,mode = 'fan_avg',distribution = 'normal')
adam = Adam(lr=0.001)
rmsprop = RMSprop(lr = 0.001)
```

Model 1 CNN

In [29]:

```python
model1 = Sequential()
model1.add(Conv1D(128,kernel_size=3,kernel_initializer='he_normal',input_shape=(timesteps, input_di
```

```
m),kernel_regularizer = l2(0.003),activation ='relu'))
model1.add(BatchNormalization())
model1.add(Dropout(0.7))
model1.add(MaxPool1D(2))
# model1.add(Conv1D(64,kernel_size=3,kernel_initializer='he_normal',kernel_regularizer =
l2(0.003),activation ='relu'))
# model1.add(BatchNormalization())
# model1.add(Dropout(0.5))
model1.add(Conv1D(32,kernel_size=3,kernel_initializer='he_normal',kernel_regularizer = l2(0.003),ac
tivation ='relu'))
model1.add(BatchNormalization())
model1.add(Dropout(0.5))
model1.add(MaxPool1D(2))
model1.add(Flatten())
model1.add(Dense(64,activation = 'relu'))
model1.add(Dropout(0.5))
model1.add(Dense(n_classes,activation = 'softmax'))
model1.summary()
```

```
W1020 15:47:24.126830  3136 nn_ops.py:4224] Large dropout rate: 0.7 (>0.5). In TensorFlow 2.x,
dropout() uses dropout rate instead of keep_prob. Please ensure that this is intended.
W1020 15:47:24.139796  3136 deprecation_wrapper.py:119] From C:\Users\patha\Anaconda3\lib\site-pac
kages\keras\backend\tensorflow_backend.py:4070: The name tf.nn.max_pool is deprecated. Please use
tf.nn.max_pool2d instead.
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_10 (Conv1D)           (None, 126, 128)          3584      
_________________________________________________________________
batch_normalization_10 (Batc (None, 126, 128)          512       
_________________________________________________________________
dropout_13 (Dropout)         (None, 126, 128)          0         
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 63, 128)           0         
_________________________________________________________________
conv1d_11 (Conv1D)           (None, 61, 32)            12320     
_________________________________________________________________
batch_normalization_11 (Batc (None, 61, 32)            128       
_________________________________________________________________
dropout_14 (Dropout)         (None, 61, 32)            0         
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 30, 32)            0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 960)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 64)                61504     
_________________________________________________________________
dropout_15 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_8 (Dense)              (None, 6)                 390       
=================================================================
Total params: 78,438
Trainable params: 78,118
Non-trainable params: 320
_________________________________________________________________
```

In [32]:

```
%%time
model1.compile(loss='categorical_crossentropy',
          optimizer=adam,
          metrics=['accuracy'])
```

```
Wall time: 62 ms
```

In [33]:

```
%%time
# Training the model
result = model1.fit(X_train,
```

```
          Y_train,
          batch_size=32,
          validation_data=(X_test, Y_test),callbacks=[lr,tm],
          epochs=30,verbose = 1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 4s 515us/step - loss: 1.9771 - accuracy: 0.5979 - val
_loss: 4.9397 - val_accuracy: 0.3207
Epoch 2/30
7352/7352 [==============================] - 3s 380us/step - loss: 1.4026 - accuracy: 0.7314 - val
_loss: 4.1904 - val_accuracy: 0.4869
Epoch 3/30
7352/7352 [==============================] - 3s 366us/step - loss: 1.1415 - accuracy: 0.8052 - val
_loss: 2.9938 - val_accuracy: 0.5724
Epoch 4/30
7352/7352 [==============================] - 3s 368us/step - loss: 0.9091 - accuracy: 0.8596 - val
_loss: 2.8087 - val_accuracy: 0.5935
Epoch 5/30
7352/7352 [==============================] - 3s 370us/step - loss: 0.7309 - accuracy: 0.8988 - val
_loss: 2.2223 - val_accuracy: 0.6464
Epoch 6/30
7352/7352 [==============================] - 3s 369us/step - loss: 0.6076 - accuracy: 0.9199 - val
_loss: 2.1633 - val_accuracy: 0.6793
Epoch 7/30
7352/7352 [==============================] - 3s 373us/step - loss: 0.5299 - accuracy: 0.9230 - val
_loss: 1.5285 - val_accuracy: 0.7343
Epoch 8/30
7352/7352 [==============================] - 3s 373us/step - loss: 0.4898 - accuracy: 0.9259 - val
_loss: 1.4341 - val_accuracy: 0.7642
Epoch 9/30
7352/7352 [==============================] - 3s 372us/step - loss: 0.4336 - accuracy: 0.9336 - val
_loss: 0.8901 - val_accuracy: 0.8205
Epoch 10/30
7352/7352 [==============================] - 3s 365us/step - loss: 0.3782 - accuracy: 0.9389 - val
_loss: 0.8177 - val_accuracy: 0.8398
Epoch 11/30
7352/7352 [==============================] - 3s 371us/step - loss: 0.3632 - accuracy: 0.9416 - val
_loss: 0.8873 - val_accuracy: 0.8157
Epoch 12/30
7352/7352 [==============================] - 3s 369us/step - loss: 0.3385 - accuracy: 0.9430 - val
_loss: 0.6890 - val_accuracy: 0.8599
Epoch 13/30
7352/7352 [==============================] - 3s 371us/step - loss: 0.3313 - accuracy: 0.9388 - val
_loss: 0.6125 - val_accuracy: 0.8785
Epoch 14/30
7352/7352 [==============================] - 3s 368us/step - loss: 0.3046 - accuracy: 0.9457 - val
_loss: 0.6476 - val_accuracy: 0.8575
Epoch 15/30
7352/7352 [==============================] - 3s 367us/step - loss: 0.2992 - accuracy: 0.9421 - val
_loss: 0.9274 - val_accuracy: 0.8130
Epoch 16/30
7352/7352 [==============================] - 3s 367us/step - loss: 0.2819 - accuracy: 0.9452 - val
_loss: 0.7748 - val_accuracy: 0.8283
Epoch 17/30
7352/7352 [==============================] - 3s 368us/step - loss: 0.2619 - accuracy: 0.9470 - val
_loss: 0.9110 - val_accuracy: 0.8283
Epoch 18/30
7352/7352 [==============================] - 3s 368us/step - loss: 0.2538 - accuracy: 0.9497 - val
_loss: 0.6998 - val_accuracy: 0.8415
Epoch 19/30
7352/7352 [==============================] - 3s 367us/step - loss: 0.2658 - accuracy: 0.9452 - val
_loss: 0.7977 - val_accuracy: 0.8178
Epoch 20/30
7352/7352 [==============================] - 3s 366us/step - loss: 0.2549 - accuracy: 0.9434 - val
_loss: 0.4292 - val_accuracy: 0.9023
Epoch 21/30
7352/7352 [==============================] - 3s 372us/step - loss: 0.2363 - accuracy: 0.9489 - val
_loss: 0.5025 - val_accuracy: 0.8748
Epoch 22/30
7352/7352 [==============================] - 3s 368us/step - loss: 0.2292 - accuracy: 0.9479 - val
_loss: 0.4710 - val_accuracy: 0.8829
Epoch 23/30
7352/7352 [==============================] - 3s 365us/step - loss: 0.2217 - accuracy: 0.9513 - val
_loss: 0.4679 - val_accuracy: 0.8826
Epoch 24/30
```

```
7352/7352 [==============================] - 3s 376us/step - loss: 0.2199 - accuracy: 0.9484 - val
_loss: 0.4759 - val_accuracy: 0.8785
Epoch 25/30
7352/7352 [==============================] - 3s 383us/step - loss: 0.2177 - accuracy: 0.9509 - val
_loss: 0.4614 - val_accuracy: 0.8856
Epoch 26/30
7352/7352 [==============================] - 3s 375us/step - loss: 0.2132 - accuracy: 0.9505 - val
_loss: 0.4465 - val_accuracy: 0.9033
Epoch 27/30
7352/7352 [==============================] - 3s 367us/step - loss: 0.2138 - accuracy: 0.9482 - val
_loss: 0.6949 - val_accuracy: 0.8317
Epoch 28/30
7352/7352 [==============================] - 3s 368us/step - loss: 0.2121 - accuracy: 0.9478 - val
_loss: 0.5558 - val_accuracy: 0.8724
Epoch 29/30
7352/7352 [==============================] - 3s 372us/step - loss: 0.2019 - accuracy: 0.9529 - val
_loss: 0.4052 - val_accuracy: 0.9080
Epoch 30/30
7352/7352 [==============================] - 3s 363us/step - loss: 0.1954 - accuracy: 0.9524 - val
_loss: 0.4131 - val_accuracy: 0.9016
Wall time: 1min 24s
```

In [38]:

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

```
2947/2947 [==============================] - 0s 149us/step
```

Out[38]:

```
[0.4131184876278614, 0.9015948176383972]
```

In [40]:

```
# Confusion Matrix
print(confusion_matrix(Y_test, model1.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 537        0         0        0                   0
SITTING                  0      326       164        0                   0
STANDING                 0       44       488        0                   0
WALKING                  0       11        14      449                   0
WALKING_DOWNSTAIRS       0        0         8        1                 401
WALKING_UPSTAIRS         0        7         0        0                   8

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            1
STANDING                           0
WALKING                           22
WALKING_DOWNSTAIRS                10
WALKING_UPSTAIRS                 456
```

Model2 CNN

In [54]:

```
model2 = Sequential()
model2.add(Conv1D(128,kernel_size=7,kernel_initializer='he_normal',input_shape=(timesteps, input_di
m),kernel_regularizer = l2(0.003),activation ='relu'))
model2.add(BatchNormalization())
model2.add(Dropout(0.7))
model2.add(MaxPool1D(2))
# model2.add(Conv1D(64,kernel_size=3,kernel_initializer='he_normal',kernel_regularizer =
l2(0.003),activation ='relu'))
# model2.add(BatchNormalization())
# model2.add(Dropout(0.5))
model2.add(Conv1D(64,kernel_size=7,kernel_initializer='he_normal',kernel_regularizer = l2(0.003),ac
tivation ='relu'))
model2.add(BatchNormalization())
```

```
model2.add(BatchNormalization())
model2.add(Dropout(0.7))
model2.add(MaxPool1D(2))
model2.add(Flatten())
model2.add(Dense(32,activation = 'relu'))
model2.add(Dropout(0.5))
model2.add(Dense(n_classes,activation = 'softmax'))
model2.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_22 (Conv1D)           (None, 122, 128)          8192      
_________________________________________________________________
batch_normalization_22 (Batc (None, 122, 128)          512       
_________________________________________________________________
dropout_31 (Dropout)         (None, 122, 128)          0         
_________________________________________________________________
max_pooling1d_12 (MaxPooling (None, 61, 128)           0         
_________________________________________________________________
conv1d_23 (Conv1D)           (None, 55, 64)            57408     
_________________________________________________________________
batch_normalization_23 (Batc (None, 55, 64)            256       
_________________________________________________________________
dropout_32 (Dropout)         (None, 55, 64)            0         
_________________________________________________________________
max_pooling1d_13 (MaxPooling (None, 27, 64)            0         
_________________________________________________________________
flatten_10 (Flatten)         (None, 1728)              0         
_________________________________________________________________
dense_19 (Dense)             (None, 32)                55328     
_________________________________________________________________
dropout_33 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_20 (Dense)             (None, 6)                 198       
=================================================================
Total params: 121,894
Trainable params: 121,510
Non-trainable params: 384
_________________________________________________________________
```

In [55]:

```
%%time
model2.compile(loss='categorical_crossentropy',
          optimizer=adam,
          metrics=['accuracy'])
```

```
Wall time: 40.9 ms
```

In [56]:

```
%%time
# Training the model
result = model2.fit(X_train,
          Y_train,
          batch_size=64,
          validation_data=(X_test, Y_test),callbacks=[lr,tm],
          epochs=30,verbose = 1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 3s 435us/step - loss: 2.0337 - accuracy: 0.5975 - val
_loss: 8.0444 - val_accuracy: 0.3563
Epoch 2/30
7352/7352 [==============================] - 2s 257us/step - loss: 1.4303 - accuracy: 0.6749 - val
_loss: 11.8737 - val_accuracy: 0.3539
Epoch 3/30
7352/7352 [==============================] - 2s 255us/step - loss: 1.1435 - accuracy: 0.7125 - val
_loss: 11.7078 - val_accuracy: 0.3841
Epoch 4/30
7352/7352 [==============================] - 2s 256us/step - loss: 0.8885 - accuracy: 0.7900 - val
_loss: 7.3924 - val_accuracy: 0.4835
```

```
_loss: 7.5921 - val_accuracy: 0.4655
Epoch 5/30
7352/7352 [==============================] - 2s 267us/step - loss: 0.6426 - accuracy: 0.8788 - val
_loss: 5.6671 - val_accuracy: 0.5317
Epoch 6/30
7352/7352 [==============================] - 2s 260us/step - loss: 0.5403 - accuracy: 0.8962 - val
_loss: 3.0467 - val_accuracy: 0.6603
Epoch 7/30
7352/7352 [==============================] - 2s 256us/step - loss: 0.4829 - accuracy: 0.9082 - val
_loss: 1.8925 - val_accuracy: 0.7523
Epoch 8/30
7352/7352 [==============================] - 2s 263us/step - loss: 0.4188 - accuracy: 0.9200 - val
_loss: 0.6114 - val_accuracy: 0.8568
Epoch 9/30
7352/7352 [==============================] - 2s 259us/step - loss: 0.4317 - accuracy: 0.9149 - val
_loss: 1.2068 - val_accuracy: 0.8134
Epoch 10/30
7352/7352 [==============================] - 2s 264us/step - loss: 0.3913 - accuracy: 0.9240 - val
_loss: 1.7844 - val_accuracy: 0.7445
Epoch 11/30
7352/7352 [==============================] - 2s 258us/step - loss: 0.3487 - accuracy: 0.9279 - val
_loss: 1.3529 - val_accuracy: 0.7940
Epoch 12/30
7352/7352 [==============================] - 2s 265us/step - loss: 0.3250 - accuracy: 0.9313 - val
_loss: 1.0702 - val_accuracy: 0.8107
Epoch 13/30
7352/7352 [==============================] - 2s 262us/step - loss: 0.3079 - accuracy: 0.9302 - val
_loss: 1.2088 - val_accuracy: 0.7954
Epoch 14/30
7352/7352 [==============================] - 2s 285us/step - loss: 0.3085 - accuracy: 0.9316 - val
_loss: 1.6542 - val_accuracy: 0.7984
Epoch 15/30
7352/7352 [==============================] - 2s 265us/step - loss: 0.3101 - accuracy: 0.9237 - val
_loss: 1.4639 - val_accuracy: 0.8127
Epoch 16/30
7352/7352 [==============================] - 2s 258us/step - loss: 0.2747 - accuracy: 0.9350 - val
_loss: 0.9176 - val_accuracy: 0.8344
Epoch 17/30
7352/7352 [==============================] - 2s 257us/step - loss: 0.2771 - accuracy: 0.9302 - val
_loss: 0.6340 - val_accuracy: 0.8626
Epoch 18/30
7352/7352 [==============================] - 2s 256us/step - loss: 0.2627 - accuracy: 0.9346 - val
_loss: 0.8502 - val_accuracy: 0.8561
Epoch 19/30
7352/7352 [==============================] - 2s 254us/step - loss: 0.2568 - accuracy: 0.9302 - val
_loss: 1.8618 - val_accuracy: 0.7774
Epoch 20/30
7352/7352 [==============================] - 2s 264us/step - loss: 0.2482 - accuracy: 0.9336 - val
_loss: 1.9363 - val_accuracy: 0.7543
Epoch 21/30
7352/7352 [==============================] - 2s 256us/step - loss: 0.2496 - accuracy: 0.9342 - val
_loss: 1.2623 - val_accuracy: 0.7971
Epoch 22/30
7352/7352 [==============================] - 2s 256us/step - loss: 0.2323 - accuracy: 0.9358 - val
_loss: 0.9469 - val_accuracy: 0.8117
Epoch 23/30
7352/7352 [==============================] - 2s 255us/step - loss: 0.2262 - accuracy: 0.9377 - val
_loss: 1.3494 - val_accuracy: 0.7889
Epoch 24/30
7352/7352 [==============================] - 2s 265us/step - loss: 0.2301 - accuracy: 0.9328 - val
_loss: 1.3258 - val_accuracy: 0.8018
Epoch 25/30
7352/7352 [==============================] - 2s 258us/step - loss: 0.2264 - accuracy: 0.9359 - val
_loss: 1.3989 - val_accuracy: 0.7978
Epoch 26/30
7352/7352 [==============================] - 2s 256us/step - loss: 0.2154 - accuracy: 0.9397 - val
_loss: 0.4875 - val_accuracy: 0.8955
Epoch 27/30
7352/7352 [==============================] - 2s 256us/step - loss: 0.2139 - accuracy: 0.9334 - val
_loss: 0.8968 - val_accuracy: 0.8266
Epoch 28/30
7352/7352 [==============================] - 2s 260us/step - loss: 0.2171 - accuracy: 0.9361 - val
_loss: 1.1524 - val_accuracy: 0.7961
Epoch 29/30
7352/7352 [==============================] - 2s 253us/step - loss: 0.2147 - accuracy: 0.9350 - val
_loss: 0.5527 - val_accuracy: 0.8816
Epoch 30/30
7352/7352 [==============================] - 2s 257us/step - loss: 0.2093 - accuracy: 0.9381 - val
```

```
7352/7352 [==============================] - 2s 257us/step - loss: 0.2095 - accuracy: 0.9381 - val
_loss: 0.6932 - val_accuracy: 0.8561
Wall time: 1min
```

In [57]:

```
model2.evaluate(X_test,Y_test)
```

```
2947/2947 [==============================] - 1s 171us/step
```

Out[57]:

```
[0.6932297824882919, 0.8561248779296875]
```

In [59]:

```
# Confusion Matrix
print(confusion_matrix(Y_test, model2.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 537        0         0        0                   0
SITTING                  5      321       164        0                   0
STANDING                 0       25       507        0                   0
WALKING                  0        0        80      416                   0
WALKING_DOWNSTAIRS       0       10         7       13                 385
WALKING_UPSTAIRS         0       65        27        4                  18

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            1
STANDING                           0
WALKING                            0
WALKING_DOWNSTAIRS                 5
WALKING_UPSTAIRS                 357
```

Using Divide and Conquer Based Approach

In [15]:

```
# 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))
```

In [16]:

```
def load_y_bi(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]
```

```
    y[y<=3] = 0
    y[y>3] = 1
    return pd.get_dummies(y).as_matrix()

def load_data_bi():
    """
    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_bi('train'),load_y_bi("test")
    return X_train, X_test, y_train, y_test
```

In [17]:

```
X_train_bi,X_test_bi,Y_train_bi,Y_test_bi  = load_data_bi()
```

```
C:\Users\patha\Anaconda3\lib\site-packages\ipykernel_launcher.py:12: FutureWarning: Method
.as_matrix will be removed in a future version. Use .values instead.
  if sys.path[0] == '':
```

In [18]:

```
model_bi= Sequential()
model_bi.add(Conv1D(32,kernel_size=3,kernel_initializer='he_normal',input_shape=(timesteps, input_d
im),kernel_regularizer = l2(0.003),activation ='relu'))
#model_bi.add(BatchNormalization())
#model_bi.add(MaxPool1D(2))
model_bi.add(Conv1D(32,kernel_size=3,kernel_initializer='he_normal',kernel_regularizer = l2(0.003),
activation ='relu'))
#model_bi.add(BatchNormalization())
model_bi.add(Dropout(0.6))
model_bi.add(MaxPool1D(2))
model_bi.add(Flatten())
model_bi.add(Dense(64,activation = 'relu'))
# model_bi.add(Dropout(0.5))
model_bi.add(Dense(2,activation = 'softmax'))
model_bi.summary()
```

```
WARNING: Logging before flag parsing goes to stderr.
W1023 16:24:53.602773  7024 nn_ops.py:4224] Large dropout rate: 0.6 (>0.5). In TensorFlow 2.x,
dropout() uses dropout rate instead of keep_prob. Please ensure that this is intended.
W1023 16:24:53.637979  7024 deprecation_wrapper.py:119] From C:\Users\patha\Anaconda3\lib\site-pac
kages\keras\backend\tensorflow_backend.py:4070: The name tf.nn.max_pool is deprecated. Please use
tf.nn.max_pool2d instead.
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 126, 32)           896
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 124, 32)           3104
_________________________________________________________________
dropout_1 (Dropout)          (None, 124, 32)           0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 62, 32)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1984)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                127040
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 130
=================================================================
Total params: 131,170
Trainable params: 131,170
Non-trainable params: 0
_________________________________________________________________
```

In [19]:

```
%%time
model_bi.compile(loss='categorical_crossentropy',
          optimizer=adam,
          metrics=['accuracy'])
```

```
Wall time: 165 ms
```

In [20]:

```
%%time
# Training the model
result = model_bi.fit(X_train_bi,
          Y_train_bi,
          batch_size=64,
          validation_data=(X_test_bi, Y_test_bi),callbacks=[lr,tm],
          epochs=30,verbose = 1)
```

```
W1023 16:25:01.943298  7024 deprecation_wrapper.py:119] From C:\Users\patha\Anaconda3\lib\site-pac
kages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables is deprecated. Please
use tf.compat.v1.global_variables instead.
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 8s 1ms/step - loss: 0.3814 - accuracy: 0.9814 - val_l
oss: 0.3111 - val_accuracy: 0.9891
Epoch 2/30
7352/7352 [==============================] - 2s 291us/step - loss: 0.2390 - accuracy: 0.9992 - val
_loss: 0.2380 - val_accuracy: 0.9810
Epoch 3/30
7352/7352 [==============================] - 2s 278us/step - loss: 0.1721 - accuracy: 0.9986 - val
_loss: 0.2054 - val_accuracy: 0.9779
Epoch 4/30
7352/7352 [==============================] - 2s 280us/step - loss: 0.1266 - accuracy: 0.9995 - val
_loss: 0.1782 - val_accuracy: 0.9756
Epoch 5/30
7352/7352 [==============================] - 2s 285us/step - loss: 0.0982 - accuracy: 0.9990 - val
_loss: 0.1229 - val_accuracy: 0.9844
Epoch 6/30
7352/7352 [==============================] - 2s 294us/step - loss: 0.0754 - accuracy: 0.9996 - val
_loss: 0.0871 - val_accuracy: 0.9891
Epoch 7/30
7352/7352 [==============================] - 2s 274us/step - loss: 0.0598 - accuracy: 0.9999 - val
_loss: 0.0805 - val_accuracy: 0.9908
Epoch 8/30
7352/7352 [==============================] - 2s 296us/step - loss: 0.0528 - accuracy: 0.9985 - val
_loss: 0.0607 - val_accuracy: 0.9922
Epoch 9/30
7352/7352 [==============================] - 2s 297us/step - loss: 0.0424 - accuracy: 0.9997 - val
_loss: 0.0520 - val_accuracy: 0.9936
Epoch 10/30
7352/7352 [==============================] - 2s 288us/step - loss: 0.0360 - accuracy: 1.0000 - val
_loss: 0.0464 - val_accuracy: 0.9956
Epoch 11/30
7352/7352 [==============================] - 2s 287us/step - loss: 0.0320 - accuracy: 1.0000 - val
_loss: 0.0519 - val_accuracy: 0.9888
Epoch 12/30
7352/7352 [==============================] - 2s 261us/step - loss: 0.0287 - accuracy: 0.9999 - val
_loss: 0.0368 - val_accuracy: 0.9983
Epoch 13/30
7352/7352 [==============================] - 2s 285us/step - loss: 0.0260 - accuracy: 1.0000 - val
_loss: 0.0337 - val_accuracy: 0.9973
Epoch 14/30
7352/7352 [==============================] - 2s 288us/step - loss: 0.0233 - accuracy: 1.0000 - val
_loss: 0.0315 - val_accuracy: 0.9976
Epoch 15/30
7352/7352 [==============================] - 2s 284us/step - loss: 0.0211 - accuracy: 1.0000 - val
_loss: 0.0297 - val_accuracy: 0.9976
Epoch 16/30
7352/7352 [==============================] - 2s 274us/step - loss: 0.0192 - accuracy: 1.0000 - val
_loss: 0.0237 - val_accuracy: 0.9990
Epoch 17/30
7352/7352 [==============================] - 2s 286us/step - loss: 0.0179 - accuracy: 1.0000 - val
_loss: 0.0231 - val_accuracy: 0.9980
```

```
Epoch 18/30
7352/7352 [==============================] - 2s 286us/step - loss: 0.0160 - accuracy: 1.0000 - val
_loss: 0.0225 - val_accuracy: 0.9980
Epoch 19/30
7352/7352 [==============================] - 2s 289us/step - loss: 0.0166 - accuracy: 0.9993 - val
_loss: 0.0198 - val_accuracy: 0.9983
Epoch 20/30
7352/7352 [==============================] - 2s 275us/step - loss: 0.0144 - accuracy: 1.0000 - val
_loss: 0.0184 - val_accuracy: 0.9983
Epoch 21/30
7352/7352 [==============================] - 2s 281us/step - loss: 0.0140 - accuracy: 0.9997 - val
_loss: 0.0292 - val_accuracy: 0.9952
Epoch 22/30
7352/7352 [==============================] - 2s 296us/step - loss: 0.0142 - accuracy: 0.9995 - val
_loss: 0.0192 - val_accuracy: 0.9983
Epoch 23/30
7352/7352 [==============================] - 2s 290us/step - loss: 0.0125 - accuracy: 1.0000 - val
_loss: 0.0177 - val_accuracy: 0.9986
Epoch 24/30
7352/7352 [==============================] - 2s 277us/step - loss: 0.0120 - accuracy: 0.9999 - val
_loss: 0.0176 - val_accuracy: 0.9983
Epoch 25/30
7352/7352 [==============================] - 2s 302us/step - loss: 0.0113 - accuracy: 1.0000 - val
_loss: 0.0164 - val_accuracy: 0.9983
Epoch 26/30
7352/7352 [==============================] - 2s 296us/step - loss: 0.0108 - accuracy: 1.0000 - val
_loss: 0.0196 - val_accuracy: 0.9976
Epoch 27/30
7352/7352 [==============================] - 2s 299us/step - loss: 0.0105 - accuracy: 1.0000 - val
_loss: 0.0133 - val_accuracy: 0.9993
Epoch 28/30
7352/7352 [==============================] - 2s 295us/step - loss: 0.0100 - accuracy: 1.0000 - val
_loss: 0.0164 - val_accuracy: 0.9983
Epoch 29/30
7352/7352 [==============================] - 2s 269us/step - loss: 0.0095 - accuracy: 1.0000 - val
_loss: 0.0138 - val_accuracy: 0.9986
Epoch 30/30
7352/7352 [==============================] - 2s 284us/step - loss: 0.0091 - accuracy: 1.0000 - val
_loss: 0.0131 - val_accuracy: 0.9986
Wall time: 1min 11s
```

In [21]:

```
model_bi.evaluate(X_test_bi,Y_test_bi)
```

```
2947/2947 [==============================] - 1s 216us/step
```

Out[21]:

```
[0.01310468845076993, 0.9986426830291748]
```

In [22]:

```
# Confusion Matrix
print(confusion_matrix(Y_test, model_bi.predict(X_test_bi)))
```

```
Pred                WALKING  WALKING_UPSTAIRS
True
LAYING                    0               537
SITTING                   2               489
STANDING                  2               530
WALKING                 496                 0
WALKING_DOWNSTAIRS      420                 0
WALKING_UPSTAIRS        471                 0
```

In [101]:

```
model_bi.save("model_bi_class.h5")
```

In [258]:

```
def load_y_new(subset):
```

```
def load_y_new(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]
    label_y = y>3
    new_y = y[label_y]
    return pd.get_dummies(new_y).as_matrix(),label_y

def load_data_new():
    """
    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_new, y_label_train = load_y_new('train')
    y_val_new,y_label_test = load_y_new('test')
    X_train_new = X_train[y_label_train]
    X_val_new = X_test[y_label_test]
    return X_train_new, X_val_new, y_train_new, y_val_new
```

In [259]:

```
X_train_st,X_test_st,Y_train_st,Y_test_st  = load_data_new()
```

In [273]:

```
model_st= Sequential()
model_st.add(Conv1D(30,kernel_size=3,kernel_initializer='glorot_normal',input_shape=(timesteps, inp
ut_dim),kernel_regularizer = l2(0.0003),activation ='relu'))
model_st.add(BatchNormalization())
model_st.add(Dropout(0.7))
# model_st.add(MaxPool1D(2))
model_st.add(Conv1D(50,kernel_size=3,kernel_initializer='glorot_normal',kernel_regularizer = l2(0.0
003),activation ='relu'))
model_st.add(BatchNormalization())
model_st.add(Dropout(0.4))
model_st.add(Conv1D(100,kernel_size=3,kernel_initializer='glorot_normal',kernel_regularizer = l2(0.
0003),activation ='relu'))
model_st.add(BatchNormalization())
model_st.add(Dropout(0.4))
#model_st.add(MaxPool1D(2))
model_st.add(Flatten())
model_st.add(Dense(100,activation = 'relu'))
model_st.add(Dropout(0.6))
model_st.add(Dense(3,activation = 'softmax'))
model_st.summary()
# model_st = Sequential()
# model_st.add(Conv1D(100,kernel_size=3,kernel_initializer='he_normal',input_shape=(timesteps, inp
ut_dim),kernel_regularizer = l2(0.0003),activation ='relu'))
# model_st.add(BatchNormalization())
# model_st.add(Dropout(0.7))
# # model_st.add(MaxPool1D(2))
# model_st.add(Conv1D(100,kernel_size=3,kernel_initializer='he_normal',kernel_regularizer = l2(0.0
003),activation ='relu'))
# model_st.add(BatchNormalization())
# model_st.add(Dropout(0.7))
# model_st.add(Conv1D(500,kernel_size=3,kernel_initializer='he_normal',kernel_regularizer = l2(0.0
003),activation ='relu'))
# model_st.add(BatchNormalization())
# model_st.add(Dropout(0.7))
# #model_st.add(MaxPool1D(2))
# model_st.add(Flatten())
# model_st.add(Dense(32,activation = 'relu'))
# model_st.add(Dropout(0.2))
# model_st.add(Dense(3,activation = 'softmax'))
# model_st.summary()
```

```
Model: "sequential_25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
conv1d_57 (Conv1D)           (None, 126, 30)           840
_________________________________________________________________
batch_normalization_25 (Batc (None, 126, 30)           120
_________________________________________________________________
dropout_63 (Dropout)         (None, 126, 30)           0
_________________________________________________________________
conv1d_58 (Conv1D)           (None, 124, 50)           4550
_________________________________________________________________
batch_normalization_26 (Batc (None, 124, 50)           200
_________________________________________________________________
dropout_64 (Dropout)         (None, 124, 50)           0
_________________________________________________________________
conv1d_59 (Conv1D)           (None, 122, 100)          15100
_________________________________________________________________
batch_normalization_27 (Batc (None, 122, 100)          400
_________________________________________________________________
dropout_65 (Dropout)         (None, 122, 100)          0
_________________________________________________________________
flatten_25 (Flatten)         (None, 12200)             0
_________________________________________________________________
dense_50 (Dense)             (None, 100)               1220100
_________________________________________________________________
dropout_66 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_51 (Dense)             (None, 3)                 303
=================================================================
Total params: 1,241,613
Trainable params: 1,241,253
Non-trainable params: 360
_________________________________________________________________
```

In [274]:

```
%%time
model_st.compile(loss='categorical_crossentropy',
          optimizer=adam,
          metrics=['accuracy'])
```

```
Wall time: 57.8 ms
```

In [275]:

```
%%time
# Training the model
result = model_st.fit(X_train_st,
          Y_train_st,
          batch_size=64,
          validation_data=(X_test_st, Y_test_st),callbacks=[lr,tm],
          epochs=30,verbose = 1)
```

```
Train on 4067 samples, validate on 1560 samples
Epoch 1/30
4067/4067 [==============================] - 3s 690us/step - loss: 0.8004 - accuracy: 0.8328 - val
_loss: 1.5153 - val_accuracy: 0.5128
Epoch 2/30
4067/4067 [==============================] - 1s 346us/step - loss: 0.3486 - accuracy: 0.8925 - val
_loss: 0.6679 - val_accuracy: 0.5929
Epoch 3/30
4067/4067 [==============================] - 1s 342us/step - loss: 0.2994 - accuracy: 0.8945 - val
_loss: 0.5829 - val_accuracy: 0.7795
Epoch 4/30
4067/4067 [==============================] - 1s 345us/step - loss: 0.3227 - accuracy: 0.9002 - val
_loss: 0.4661 - val_accuracy: 0.8276
Epoch 5/30
4067/4067 [==============================] - 1s 336us/step - loss: 0.3862 - accuracy: 0.8849 - val
_loss: 0.3554 - val_accuracy: 0.8635
Epoch 6/30
4067/4067 [==============================] - 1s 334us/step - loss: 0.2901 - accuracy: 0.8930 - val
_loss: 0.3619 - val_accuracy: 0.8718
Epoch 7/30
4067/4067 [==============================] - 1s 341us/step - loss: 0.2782 - accuracy: 0.9056 - val
_loss: 0.3442 - val_accuracy: 0.8859
```

```
Epoch 8/30
4067/4067 [==============================] - 1s 332us/step - loss: 0.2703 - accuracy: 0.8999 - val
_loss: 0.4160 - val_accuracy: 0.8846
Epoch 9/30
4067/4067 [==============================] - 1s 339us/step - loss: 0.2720 - accuracy: 0.9009 - val
_loss: 0.3667 - val_accuracy: 0.8821
Epoch 10/30
4067/4067 [==============================] - 1s 338us/step - loss: 0.2654 - accuracy: 0.9078 - val
_loss: 0.3699 - val_accuracy: 0.8891
Epoch 11/30
4067/4067 [==============================] - 1s 339us/step - loss: 0.2618 - accuracy: 0.9100 - val
_loss: 0.4152 - val_accuracy: 0.8865
Epoch 12/30
4067/4067 [==============================] - 1s 330us/step - loss: 0.2650 - accuracy: 0.9066 - val
_loss: 0.4335 - val_accuracy: 0.8859
Epoch 13/30
4067/4067 [==============================] - 1s 336us/step - loss: 0.2628 - accuracy: 0.9053 - val
_loss: 0.4117 - val_accuracy: 0.8936
Epoch 14/30
4067/4067 [==============================] - 1s 335us/step - loss: 0.2590 - accuracy: 0.9046 - val
_loss: 0.4061 - val_accuracy: 0.8885
Epoch 15/30
4067/4067 [==============================] - 1s 336us/step - loss: 0.2610 - accuracy: 0.9036 - val
_loss: 0.4055 - val_accuracy: 0.8853
Epoch 16/30
4067/4067 [==============================] - 1s 330us/step - loss: 0.2577 - accuracy: 0.9078 - val
_loss: 0.4225 - val_accuracy: 0.8878
Epoch 17/30
4067/4067 [==============================] - 1s 339us/step - loss: 0.2663 - accuracy: 0.9093 - val
_loss: 0.3744 - val_accuracy: 0.8917
Epoch 18/30
4067/4067 [==============================] - 1s 333us/step - loss: 0.2518 - accuracy: 0.9105 - val
_loss: 0.4570 - val_accuracy: 0.8712
Epoch 19/30
4067/4067 [==============================] - 1s 334us/step - loss: 0.2481 - accuracy: 0.9122 - val
_loss: 0.4341 - val_accuracy: 0.8904
Epoch 20/30
4067/4067 [==============================] - 1s 339us/step - loss: 0.2504 - accuracy: 0.9110 - val
_loss: 0.4224 - val_accuracy: 0.8897
Epoch 21/30
4067/4067 [==============================] - 1s 333us/step - loss: 0.2476 - accuracy: 0.9144 - val
_loss: 0.4934 - val_accuracy: 0.8853
Epoch 22/30
4067/4067 [==============================] - 1s 330us/step - loss: 0.2405 - accuracy: 0.9144 - val
_loss: 0.5077 - val_accuracy: 0.8923
Epoch 23/30
4067/4067 [==============================] - 1s 331us/step - loss: 0.2427 - accuracy: 0.9098 - val
_loss: 0.5086 - val_accuracy: 0.8897
Epoch 24/30
4067/4067 [==============================] - 1s 344us/step - loss: 0.2477 - accuracy: 0.9132 - val
_loss: 0.4303 - val_accuracy: 0.8923
Epoch 25/30
4067/4067 [==============================] - 1s 332us/step - loss: 0.2493 - accuracy: 0.9139 - val
_loss: 0.4313 - val_accuracy: 0.8904
Epoch 26/30
4067/4067 [==============================] - 1s 328us/step - loss: 0.2451 - accuracy: 0.9184 - val
_loss: 0.5022 - val_accuracy: 0.8891
Epoch 27/30
4067/4067 [==============================] - 1s 339us/step - loss: 0.2418 - accuracy: 0.9152 - val
_loss: 0.4845 - val_accuracy: 0.8904
Epoch 28/30
4067/4067 [==============================] - 1s 333us/step - loss: 0.2387 - accuracy: 0.9159 - val
_loss: 0.4856 - val_accuracy: 0.8923
Epoch 29/30
4067/4067 [==============================] - 1s 330us/step - loss: 0.2477 - accuracy: 0.9174 - val
_loss: 0.4358 - val_accuracy: 0.8904
Epoch 30/30
4067/4067 [==============================] - 1s 334us/step - loss: 0.2458 - accuracy: 0.9203 - val
_loss: 0.3938 - val_accuracy: 0.8936
Wall time: 46.6 s
```

In [ ]:

In [277]:

```
# Confusion Matrix
#print(confusion_matrix(np.argmax(Y_test_st,axis=1),
np.argmax(model_st.predict(X_test_st),axis=1)))
model_st.evaluate(X_test_st,Y_test_st)
```

```
1560/1560 [==============================] - 0s 199us/step
```

Out[277]:

```
[0.3938026012136386, 0.8935897350311279]
```

In [ ]:

```
print(confusion_matrix(np.argmax(Y_test_st,axis=1),
np.argmax(model_st.predict(X_test_st),axis=1)))
```

In [276]:

```
model_st.save("model_st_class.h5")
```

In [23]:

```
def load_y_new(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]
    label_y = y<=3
    new_y = y[label_y]
    return pd.get_dummies(new_y).as_matrix(),label_y

def load_data_new():
    """
    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_new, y_label_train = load_y_new('train')
    y_val_new,y_label_test = load_y_new('test')
    X_train_new = X_train[y_label_train]
    X_val_new = X_test[y_label_test]
    return X_train_new, X_val_new, y_train_new, y_val_new
```

In [24]:

```
X_train_dy,X_test_dy,Y_train_dy,Y_test_dy  = load_data_new()
```

```
C:\Users\patha\Anaconda3\lib\site-packages\ipykernel_launcher.py:12: FutureWarning: Method
.as_matrix will be removed in a future version. Use .values instead.
  if sys.path[0] == '':
```

In [207]:

```
model_dy= Sequential()
model_dy.add(Conv1D(64,kernel_size=5,kernel_initializer='glorot_normal',input_shape=(timesteps, inp
ut_dim),kernel_regularizer = l2(0.0003),activation ='relu'))
#model_dy.add(BatchNormalization())
#model_dy.add(MaxPool1D(2))
model_dy.add(Conv1D(32,kernel_size=5,kernel_initializer='glorot_normal',kernel_regularizer = l2(0.0
003),activation ='relu'))
#model_dy.add(BatchNormalization())
model_dy.add(Dropout(0.7))
# model_dy.add(Conv1D(32,kernel_size=7,kernel_initializer='he_normal',kernel_regularizer = l2(0.00
3),activation ='relu'))
# #model_dy.add(BatchNormalization())
```

```
# model_dy.add(Dropout(0.6))
model_dy.add(MaxPool1D(2))
model_dy.add(Flatten())
model_dy.add(Dense(16,activation = 'relu'))
model_dy.add(Dropout(0.2))
model_dy.add(Dense(3,activation = 'softmax'))
model_dy.summary()
```

```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_37 (Conv1D)           (None, 124, 64)           2944
_________________________________________________________________
conv1d_38 (Conv1D)           (None, 120, 32)           10272
_________________________________________________________________
dropout_39 (Dropout)         (None, 120, 32)           0
_________________________________________________________________
max_pooling1d_13 (MaxPooling (None, 60, 32)            0
_________________________________________________________________
flatten_17 (Flatten)         (None, 1920)              0
_________________________________________________________________
dense_34 (Dense)             (None, 16)                30736
_________________________________________________________________
dropout_40 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_35 (Dense)             (None, 3)                 51
=================================================================
Total params: 44,003
Trainable params: 44,003
Non-trainable params: 0
_________________________________________________________________
```

In [208]:

```
%%time
model_dy.compile(loss='categorical_crossentropy',
          optimizer=adam,
          metrics=['accuracy'])
```

```
Wall time: 122 ms
```

In [209]:

```
%%time
# Training the model
result = model_dy.fit(X_train_dy,
          Y_train_dy,
          batch_size=64,
          validation_data=(X_test_dy, Y_test_dy),callbacks=[lr,tm],
          epochs=30,verbose = 1)
```

```
Train on 3285 samples, validate on 1387 samples
Epoch 1/30
3285/3285 [==============================] - 1s 439us/step - loss: 0.6504 - accuracy: 0.7233 - val
_loss: 0.5196 - val_accuracy: 0.8558
Epoch 2/30
3285/3285 [==============================] - 1s 216us/step - loss: 0.1447 - accuracy: 0.9589 - val
_loss: 0.2770 - val_accuracy: 0.9387
Epoch 3/30
3285/3285 [==============================] - 1s 201us/step - loss: 0.0706 - accuracy: 0.9854 - val
_loss: 0.3495 - val_accuracy: 0.9315
Epoch 4/30
3285/3285 [==============================] - 1s 196us/step - loss: 0.0435 - accuracy: 0.9942 - val
_loss: 0.3510 - val_accuracy: 0.9481
Epoch 5/30
3285/3285 [==============================] - 1s 195us/step - loss: 0.0377 - accuracy: 0.9957 - val
_loss: 0.2735 - val_accuracy: 0.9618
Epoch 6/30
3285/3285 [==============================] - 1s 160us/step - loss: 0.0324 - accuracy: 0.9970 - val
_loss: 0.3314 - val_accuracy: 0.9531
Epoch 7/30
3285/3285 [==============================] - 1s 163us/step - loss: 0.0315 - accuracy: 0.9963 - val
```

```
_loss: 0.2419 - val_accuracy: 0.9567
Epoch 8/30
3285/3285 [==============================] - 1s 161us/step - loss: 0.0253 - accuracy: 0.9985 - val
_loss: 0.3123 - val_accuracy: 0.9589
Epoch 9/30
3285/3285 [==============================] - 1s 168us/step - loss: 0.0354 - accuracy: 0.9951 - val
_loss: 0.2665 - val_accuracy: 0.9603
Epoch 10/30
3285/3285 [==============================] - 1s 170us/step - loss: 0.0264 - accuracy: 0.9976 - val
_loss: 0.2918 - val_accuracy: 0.9452
Epoch 11/30
3285/3285 [==============================] - 1s 169us/step - loss: 0.0253 - accuracy: 0.9979 - val
_loss: 0.2884 - val_accuracy: 0.9632
Epoch 12/30
3285/3285 [==============================] - 1s 168us/step - loss: 0.0261 - accuracy: 0.9985 - val
_loss: 0.2508 - val_accuracy: 0.9553
Epoch 13/30
3285/3285 [==============================] - 1s 170us/step - loss: 0.0230 - accuracy: 0.9994 - val
_loss: 0.2899 - val_accuracy: 0.9618
Epoch 14/30
3285/3285 [==============================] - 1s 171us/step - loss: 0.0234 - accuracy: 0.9976 - val
_loss: 0.2945 - val_accuracy: 0.9611
Epoch 15/30
3285/3285 [==============================] - 1s 162us/step - loss: 0.0218 - accuracy: 0.9991 - val
_loss: 0.3251 - val_accuracy: 0.9611
Epoch 16/30
3285/3285 [==============================] - 1s 160us/step - loss: 0.0201 - accuracy: 1.0000 - val
_loss: 0.3274 - val_accuracy: 0.9611
Epoch 17/30
3285/3285 [==============================] - 1s 169us/step - loss: 0.0211 - accuracy: 0.9991 - val
_loss: 0.3114 - val_accuracy: 0.9611
Epoch 18/30
3285/3285 [==============================] - 1s 158us/step - loss: 0.0204 - accuracy: 0.9994 - val
_loss: 0.2790 - val_accuracy: 0.9611
Epoch 19/30
3285/3285 [==============================] - 1s 173us/step - loss: 0.0210 - accuracy: 0.9991 - val
_loss: 0.3069 - val_accuracy: 0.9632
Epoch 20/30
3285/3285 [==============================] - 1s 172us/step - loss: 0.0194 - accuracy: 0.9994 - val
_loss: 0.3064 - val_accuracy: 0.9618
Epoch 21/30
3285/3285 [==============================] - 1s 162us/step - loss: 0.0197 - accuracy: 0.9994 - val
_loss: 0.3333 - val_accuracy: 0.9603
Epoch 22/30
3285/3285 [==============================] - 1s 177us/step - loss: 0.0189 - accuracy: 0.9994 - val
_loss: 0.3040 - val_accuracy: 0.9611
Epoch 23/30
3285/3285 [==============================] - 1s 174us/step - loss: 0.0201 - accuracy: 0.9988 - val
_loss: 0.3651 - val_accuracy: 0.9553
Epoch 24/30
3285/3285 [==============================] - 1s 174us/step - loss: 0.0227 - accuracy: 0.9976 - val
_loss: 0.2609 - val_accuracy: 0.9582
Epoch 25/30
3285/3285 [==============================] - 1s 162us/step - loss: 0.0187 - accuracy: 0.9997 - val
_loss: 0.3188 - val_accuracy: 0.9603
Epoch 26/30
3285/3285 [==============================] - 1s 171us/step - loss: 0.0180 - accuracy: 0.9997 - val
_loss: 0.2902 - val_accuracy: 0.9618
Epoch 27/30
3285/3285 [==============================] - 1s 164us/step - loss: 0.0191 - accuracy: 0.9991 - val
_loss: 0.3629 - val_accuracy: 0.9603
Epoch 28/30
3285/3285 [==============================] - 1s 168us/step - loss: 0.0180 - accuracy: 0.9997 - val
_loss: 0.3207 - val_accuracy: 0.9625
Epoch 29/30
3285/3285 [==============================] - 1s 194us/step - loss: 0.0190 - accuracy: 0.9991 - val
_loss: 0.3451 - val_accuracy: 0.9589
Epoch 30/30
3285/3285 [==============================] - 1s 200us/step - loss: 0.0182 - accuracy: 0.9985 - val
_loss: 0.3329 - val_accuracy: 0.9611
Wall time: 20.4 s
```

In [212]:

```
# Confusion Matrix
#print(confusion_matrix(Y_test_dv, model_dv.predict(X_test_dv)))
```

```
#print(confusion_matrix(Y_test_dy, model_dy.predict(X_test_dy)))
model_dy.evaluate(X_test_dy,Y_test_dy)
```

```
1387/1387 [==============================] - 0s 128us/step
```

Out[212]:

```
[0.13676273131712186, 0.9769286513328552]
```

In [49]:

```
model_dy.save("model_dy_class.h5")
```

In [210]:

```
model_dy.load_weights("model_dy_class.h5")
```

Sharpening Test Data

In [95]:

```
def sharpen(x_test, sigma, alpha):
    d = x_test.shape[0]
    r = x_test.shape[1]
    c = x_test.shape[2]
    container = np.empty((d,r, c))
    i = 0

    for row in x_test:
        test = row
        blurred = ndimage.gaussian_filter(test, sigma)
        sharpened = test + alpha * (test - blurred)
        container[i] = sharpened
        i = i + 1
    return container
```

In [96]:

```
from scipy import ndimage
alpha = np.arange(0.05, 2.55, 0.05)
sigma = np.arange(5, 10, 1)
from sklearn.metrics import accuracy_score,confusion_matrix
def display_output(X_test,Y_test):
    accuracy=[]
    for s in sigma:
        for a in alpha:
            # Sharpen test data with various sigma (for Gaussian filter) and alpha value combinati
ns
            X_test_sharpen = sharpen(X_test, s, a)
            pred_dyna_sharpen = model_dy.predict(X_test_sharpen)
            accuracy.append(accuracy_score(Y_test, np.argmax(pred_dyna_sharpen, axis=1)))
    return(accuracy)
            #print(">>> sigma={}, alpha={:.2f}".format(s, a))
#             print(accuracy_score(np.argmax(Y_test,axis=1), np.argmax(pred_dyna_sharpen, axis=1)),
#             print(confusion_matrix(np.argmax(Y_test,axis=1), np.argmax(pred_dyna_sharpen, axis=1,
))
```

In [69]:

```
# 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()
```

```
        _read_csv(filename).as_matrix()
        )

    return np.transpose(signals_data, (1, 2, 0))

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

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

    return X_train, X_test, y_train, y_test
```

In [71]:

```
import warnings
warnings.filterwarnings("ignore")
X_train, X_test, Y_train, Y_test = load_data()
print("X train shape :",X_train.shape)
print("X test shape :",X_test.shape)
print("Y train shape :",Y_train.shape)
print("Y train shape :",Y_test.shape)
```

```
X train shape : (7352, 128, 9)
X test shape : (2947, 128, 9)
Y train shape : (7352, 6)
Y train shape : (2947, 6)
```

Classification of Sitting and Laying

In [196]:

```
X_train_st_sitlay = X_train_st[np.argmax(Y_train_st,axis=1)!=1]
X_test_st_sitlay = X_test_st[np.argmax(Y_test_st,axis=1)!=1]
Y_train_st_sitlay = Y_train_st[np.argmax(Y_train_st,axis=1)!=1]
Y_test_st_sitlay = Y_test_st[np.argmax(Y_test_st,axis=1)!=1]
```

In [213]:

```
model_st_bi= Sequential()
model_st_bi.add(Conv1D(64,kernel_size=5,kernel_initializer='glorot_normal',input_shape=(timesteps,
input_dim),kernel_regularizer = l2(0.0003),activation ='relu'))
#model_st_bi.add(BatchNormalization())
#model_st_bi.add(MaxPool1D(2))
model_st_bi.add(Conv1D(32,kernel_size=5,kernel_initializer='glorot_normal',kernel_regularizer = l2(
0.0003),activation ='relu'))
#model_st_bi.add(BatchNormalization())
model_st_bi.add(Dropout(0.7))
# model_st_bi.add(Conv1D(32,kernel_size=7,kernel_initializer='he_normal',kernel_regularizer = l2(0
.003),activation ='relu'))
# #model_st_bi.add(BatchNormalization())
# model_st_bi.add(Dropout(0.6))
model_st_bi.add(MaxPool1D(2))
model_st_bi.add(Flatten())
model_st_bi.add(Dense(16,activation = 'relu'))
model_st_bi.add(Dropout(0.3))
model_st_bi.add(Dense(3,activation = 'softmax'))
model_st_bi.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_39 (Conv1D)           (None, 124, 64)           2944      
_________________________________________________________________
conv1d_40 (Conv1D)           (None, 120, 32)           10272     
_________________________________________________________________
dropout_41 (Dropout)         (None, 120, 32)           0         
_________________________________________________________________
max_pooling1d_14 (MaxPooling (None, 60, 32)            0         
_________________________________________________________________
flatten_18 (Flatten)         (None, 1920)              0         
_________________________________________________________________
dense_36 (Dense)             (None, 16)                30736     
_________________________________________________________________
dropout_42 (Dropout)         (None, 16)                0         
_________________________________________________________________
dense_37 (Dense)             (None, 3)                 51        
=================================================================
Total params: 44,003
Trainable params: 44,003
Non-trainable params: 0
_________________________________________________________________
```

In [214]:

```
%%time
model_st_bi.compile(loss='categorical_crossentropy',
          optimizer=adam,
          metrics=['accuracy'])
```

```
Wall time: 40.9 ms
```

In [215]:

```
%%time
# Training the model
result = model_st_bi.fit(X_train_st_sitlay,
          Y_train_st_sitlay,
          batch_size=64,
          validation_data=(X_test_st_sitlay, Y_test_st_sitlay),callbacks=[lr,tm],
          epochs=30,verbose = 1)
```

```
Train on 2693 samples, validate on 1028 samples
Epoch 1/30
2693/2693 [==============================] - 2s 616us/step - loss: 0.1667 - accuracy: 0.9577 - val
_loss: 0.0167 - val_accuracy: 1.0000
Epoch 2/30
2693/2693 [==============================] - 0s 161us/step - loss: 0.0235 - accuracy: 0.9978 - val
_loss: 0.0149 - val_accuracy: 1.0000
Epoch 3/30
2693/2693 [==============================] - 0s 173us/step - loss: 0.0161 - accuracy: 0.9996 - val
_loss: 0.0134 - val_accuracy: 1.0000
Epoch 4/30
2693/2693 [==============================] - 0s 169us/step - loss: 0.0141 - accuracy: 1.0000 - val
_loss: 0.0123 - val_accuracy: 1.0000
Epoch 5/30
2693/2693 [==============================] - 0s 168us/step - loss: 0.0134 - accuracy: 1.0000 - val
_loss: 0.0114 - val_accuracy: 1.0000
Epoch 6/30
2693/2693 [==============================] - 0s 169us/step - loss: 0.0114 - accuracy: 1.0000 - val
_loss: 0.0108 - val_accuracy: 1.0000
Epoch 7/30
2693/2693 [==============================] - 0s 163us/step - loss: 0.0112 - accuracy: 0.9996 - val
_loss: 0.0102 - val_accuracy: 1.0000
Epoch 8/30
2693/2693 [==============================] - 0s 168us/step - loss: 0.0105 - accuracy: 1.0000 - val
_loss: 0.0097 - val_accuracy: 1.0000
Epoch 9/30
2693/2693 [==============================] - 0s 169us/step - loss: 0.0105 - accuracy: 0.9996 - val
_loss: 0.0093 - val_accuracy: 1.0000
Epoch 10/30
2693/2693 [==============================] - 0s 167us/step - loss: 0.0102 - accuracy: 0.9993 - val
```

```
2693/2693 [==============================] - 0s 167us/step - loss: 0.0102 - accuracy: 0.9993 - val
_loss: 0.0091 - val_accuracy: 1.0000
Epoch 11/30
2693/2693 [==============================] - 1s 221us/step - loss: 0.0108 - accuracy: 0.9989 - val
_loss: 0.0088 - val_accuracy: 1.0000
Epoch 12/30
2693/2693 [==============================] - 0s 173us/step - loss: 0.0105 - accuracy: 0.9996 - val
_loss: 0.0086 - val_accuracy: 1.0000
Epoch 13/30
2693/2693 [==============================] - 0s 173us/step - loss: 0.0095 - accuracy: 0.9993 - val
_loss: 0.0085 - val_accuracy: 1.0000
Epoch 14/30
2693/2693 [==============================] - 0s 164us/step - loss: 0.0094 - accuracy: 0.9989 - val
_loss: 0.0083 - val_accuracy: 1.0000
Epoch 15/30
2693/2693 [==============================] - 0s 165us/step - loss: 0.0091 - accuracy: 0.9989 - val
_loss: 0.0081 - val_accuracy: 1.0000
Epoch 16/30
2693/2693 [==============================] - 0s 162us/step - loss: 0.0096 - accuracy: 0.9996 - val
_loss: 0.0079 - val_accuracy: 1.0000
Epoch 17/30
2693/2693 [==============================] - 0s 157us/step - loss: 0.0092 - accuracy: 1.0000 - val
_loss: 0.0078 - val_accuracy: 1.0000
Epoch 18/30
2693/2693 [==============================] - 0s 170us/step - loss: 0.0085 - accuracy: 0.9996 - val
_loss: 0.0076 - val_accuracy: 1.0000
Epoch 19/30
2693/2693 [==============================] - 0s 185us/step - loss: 0.0085 - accuracy: 1.0000 - val
_loss: 0.0075 - val_accuracy: 1.0000
Epoch 20/30
2693/2693 [==============================] - 1s 212us/step - loss: 0.0078 - accuracy: 0.9996 - val
_loss: 0.0074 - val_accuracy: 1.0000
Epoch 21/30
2693/2693 [==============================] - 1s 186us/step - loss: 0.0083 - accuracy: 0.9996 - val
_loss: 0.0073 - val_accuracy: 1.0000
Epoch 22/30
2693/2693 [==============================] - 0s 179us/step - loss: 0.0088 - accuracy: 1.0000 - val
_loss: 0.0072 - val_accuracy: 1.0000
Epoch 23/30
2693/2693 [==============================] - 1s 203us/step - loss: 0.0093 - accuracy: 0.9985 - val
_loss: 0.0071 - val_accuracy: 1.0000
Epoch 24/30
2693/2693 [==============================] - 1s 221us/step - loss: 0.0076 - accuracy: 1.0000 - val
_loss: 0.0070 - val_accuracy: 1.0000
Epoch 25/30
2693/2693 [==============================] - 0s 180us/step - loss: 0.0089 - accuracy: 0.9993 - val
_loss: 0.0070 - val_accuracy: 1.0000
Epoch 26/30
2693/2693 [==============================] - 0s 163us/step - loss: 0.0078 - accuracy: 1.0000 - val
_loss: 0.0069 - val_accuracy: 1.0000
Epoch 27/30
2693/2693 [==============================] - 0s 168us/step - loss: 0.0085 - accuracy: 0.9993 - val
_loss: 0.0068 - val_accuracy: 1.0000
Epoch 28/30
2693/2693 [==============================] - 0s 175us/step - loss: 0.0078 - accuracy: 1.0000 - val
_loss: 0.0067 - val_accuracy: 1.0000
Epoch 29/30
2693/2693 [==============================] - 0s 160us/step - loss: 0.0083 - accuracy: 1.0000 - val
_loss: 0.0066 - val_accuracy: 1.0000
Epoch 30/30
2693/2693 [==============================] - 0s 161us/step - loss: 0.0083 - accuracy: 1.0000 - val
_loss: 0.0066 - val_accuracy: 1.0000
Wall time: 17.7 s
```

In [219]:

```
print(confusion_matrix(np.argmax(Y_test_st_sitlay,axis=1), np.argmax(model_st_bi.predict(X_test_st_
sitlay),axis=1)))
model_st_bi.evaluate(X_test_st_sitlay,Y_test_st_sitlay)
```

```
[[491   0]
 [  0 537]]
1028/1028 [==============================] - 0s 125us/step
```

Out[219]:

[0.00658893180061639, 1.0]

Best Sigma and alpha for Test Sharpening

In [ ]:

```
accuracy=[]
alpha = np.arange(0.05, 2.55, 0.05)
sigma = np.arange(0, 5, 0.5)
for s in sigma:
    for a in alpha:
        X_test_st_ = sharpen(X_test_st_,s,a)
        X_test_dy_ = sharpen(X_test_dy_,s,a)
        y_pred_st = model_st.predict_classes(X_test_st_)+3
        y_pred_dy = model_dy.predict_classes(X_test_dy_)
        total_y = np.concatenate([y_pred_st,y_pred_dy])
        accuracy.append(accuracy_score(np.argmax(y_test,axis=1),total_y))
```

Stacking the models for Testing

In [351]:

```
y_pred_bi = model_bi.predict_classes(X_test_bi)
```

In [352]:

```
X_test_st_= X_test[y_pred_bi>0]    # Static class
X_test_dy_ = X_test[y_pred_bi<1]   # Dynamic Class
```

In [353]:

```
Y_test_st_ = Y_test[y_pred_bi>0]
Y_test_dy_ = Y_test[y_pred_bi<1]
```

In [354]:

```
y_test = np.concatenate([Y_test_st_,Y_test_dy_]) #
```

Sharpening of Test Data

In [355]:

```
X_test_st_ = sharpen(X_test_st_,5,0.05)
X_test_dy_ = sharpen(X_test_dy_,5,0.05)
```

In [356]:

```
y_pred_st = model_st.predict_classes(X_test_st_)+3
y_pred_dy = model_dy.predict_classes(X_test_dy_)
```

In [357]:

```
total_y = np.concatenate([y_pred_st,y_pred_dy])
```

In [359]:

```
print("Accuracy Using Divide and Conquer Method->",accuracy_score(np.argmax(y_test,axis=1),total_y
))
```

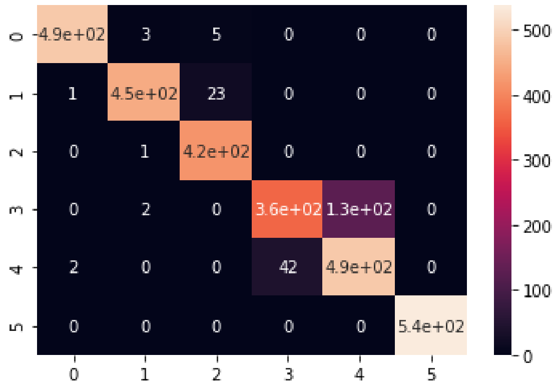
```
Accuracy Using Divide and Conquer Method-> 0.9424377332880896
```

In [309]:

```python
import seaborn as sns
sns.heatmap(confusion_matrix(np.argmax(y_test,axis=1),total_y),annot=True)
```

Out[309]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2000806e5f8>
```



Conclusion

In [360]:

```python
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","Accuracy","loss"]
x.add_row(["LSTM with one Layer",90.97,0.3087])
x.add_row(["LSTM with two Layer",91.78,0.3971])
x.add_row(["LSTM with three Layer",89.51,0.4863])
x.add_row(["Divide-Conquer Method",94.24,0.2042])
print(x)
```

```
+-----------------------+----------+--------+
|         Model         | Accuracy |  loss  |
+-----------------------+----------+--------+
|  LSTM with one Layer  |  90.97   | 0.3087 |
|  LSTM with two Layer  |  91.78   | 0.3971 |
| LSTM with three Layer |  89.51   | 0.4863 |
| Divide-Conquer Method |  94.24   | 0.2042 |
+-----------------------+----------+--------+
```

1. All three model is getting confused between standing and sitting, after running so many code at last I got 91.8% accuracy in LSTM with 2 layers.
2. Some code gives nan as my losses and get stuck in same accuracy of 16.83% for many epochs .