HAR_LSTM ASSIGNMENT

September 25, 2018

```
In [1]: # Importing Libraries

In [2]: import pandas as pd
        import numpy as np
        import warnings
        warnings.filterwarnings('ignore')

In [3]: # 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',
        }
```

0.0.1 Data

```
In [4]: # Data directory
        DATADIR = 'UCI_HAR_Dataset'

In [5]: # 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 [6]:
```
# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the load
def load_signals(subset):
    signals_data = []

    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 [7]:
```
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()
```

In [8]:
```
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 [9]:
```
# Importing tensorflow
np.random.seed(42)
import tensorflow as tf
tf.set_random_seed(42)
```

```
In [10]: # Configuring a session
         session_conf = tf.ConfigProto(
             intra_op_parallelism_threads=1,
             inter_op_parallelism_threads=1
         )
```

```
In [11]: # 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 [12]: # Importing libraries
         from keras.models import Sequential
         from keras.layers import LSTM
         from keras.layers.core import Dense, Dropout
```

```
In [13]: # Initializing parameters
         epochs = 30
         batch_size = 16
         n_hidden = 32
```

```
In [14]: # Utility function to count the number of classes
         def _count_classes(y):
             return len(set([tuple(category) for category in y]))
```

```
In [15]: # Loading the train and test data
         X_train, X_test, Y_train, Y_test = load_data()
```

```
In [16]: 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

1 (1) Model having 1 LSTM layer with 32 LSTM Units

```
In [17]: # Initiliazing the sequential model
         model = Sequential()
```

```python
# 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()

# Compiling the model
model.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['accuracy'
                                                                             
# Training the model
history = model.fit(X_train, Y_train, batch_size=batch_size,validation_data=(X_test, 
```

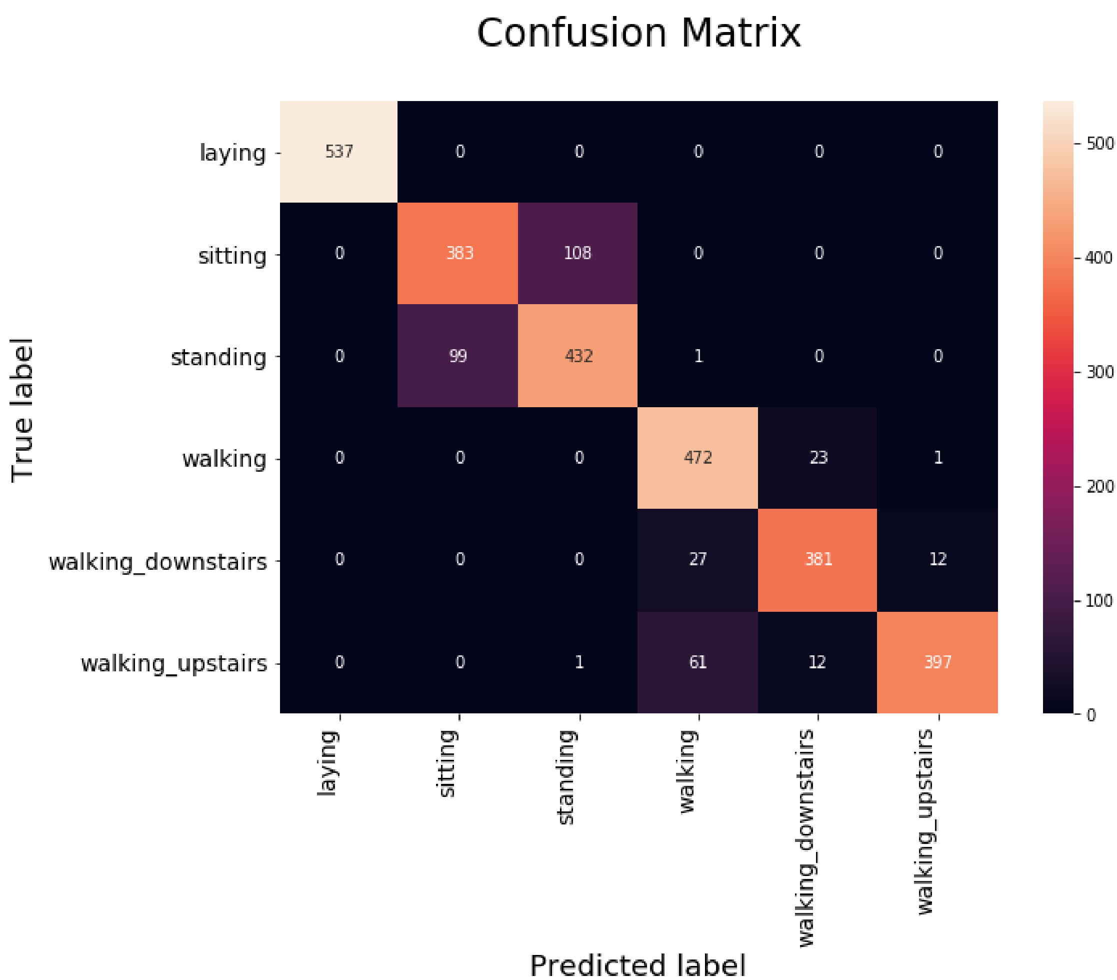
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 32)                5376
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 198
=================================================================
Total params: 5,574
Trainable params: 5,574
Non-trainable params: 0
_________________________________________________________________
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 40s 5ms/step - loss: 1.3139 - acc: 0.4358 - val_lo
Epoch 2/30
7352/7352 [==============================] - 38s 5ms/step - loss: 0.9788 - acc: 0.5773 - val_lo
Epoch 3/30
7352/7352 [==============================] - 39s 5ms/step - loss: 0.7977 - acc: 0.6457 - val_lo
Epoch 4/30
7352/7352 [==============================] - 39s 5ms/step - loss: 0.6989 - acc: 0.6582 - val_lo
Epoch 5/30
7352/7352 [==============================] - 37s 5ms/step - loss: 0.6359 - acc: 0.6797 - val_lo
Epoch 6/30
7352/7352 [==============================] - 37s 5ms/step - loss: 0.5819 - acc: 0.6865 - val_lo
Epoch 7/30
7352/7352 [==============================] - 37s 5ms/step - loss: 0.5676 - acc: 0.7058 - val_lo
Epoch 8/30
7352/7352 [==============================] - 37s 5ms/step - loss: 0.5583 - acc: 0.7217 - val_lo
Epoch 9/30
7352/7352 [==============================] - 37s 5ms/step - loss: 0.5386 - acc: 0.7557 - val_lo
Epoch 10/30
7352/7352 [==============================] - 37s 5ms/step - loss: 0.4804 - acc: 0.7911 - val_lo
```

```
Epoch 11/30
7352/7352 [==============================] - 37s 5ms/step - loss: 0.4320 - acc: 0.8052 - val_lo
Epoch 12/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.4279 - acc: 0.8062 - val_lo
Epoch 13/30
7352/7352 [==============================] - 38s 5ms/step - loss: 0.3911 - acc: 0.8130 - val_lo
Epoch 14/30
7352/7352 [==============================] - 37s 5ms/step - loss: 0.3898 - acc: 0.8313 - val_lo
Epoch 15/30
7352/7352 [==============================] - 37s 5ms/step - loss: 0.3308 - acc: 0.8942 - val_lo
Epoch 16/30
7352/7352 [==============================] - 37s 5ms/step - loss: 0.2891 - acc: 0.9176 - val_lo
Epoch 17/30
7352/7352 [==============================] - 38s 5ms/step - loss: 0.2660 - acc: 0.9246 - val_lo
Epoch 18/30
7352/7352 [==============================] - 38s 5ms/step - loss: 0.2538 - acc: 0.9251 - val_lo
Epoch 19/30
7352/7352 [==============================] - 38s 5ms/step - loss: 0.2502 - acc: 0.9312 - val_lo
Epoch 20/30
7352/7352 [==============================] - 46s 6ms/step - loss: 0.1980 - acc: 0.9382 - val_lo
Epoch 21/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2018 - acc: 0.9372 - val_lo
Epoch 22/30
7352/7352 [==============================] - 39s 5ms/step - loss: 0.2455 - acc: 0.9310 - val_lo
Epoch 23/30
7352/7352 [==============================] - 40s 5ms/step - loss: 0.2194 - acc: 0.9329 - val_lo
Epoch 24/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2282 - acc: 0.9304 - val_lo
Epoch 25/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.2166 - acc: 0.9359 - val_lo
Epoch 26/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2173 - acc: 0.9350 - val_lo
Epoch 27/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.2224 - acc: 0.9353 - val_lo
Epoch 28/30
7352/7352 [==============================] - 39s 5ms/step - loss: 0.1961 - acc: 0.9385 - val_lo
Epoch 29/30
7352/7352 [==============================] - 39s 5ms/step - loss: 0.1876 - acc: 0.9416 - val_lo
Epoch 30/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1999 - acc: 0.9411 - val_lo
```

```
In [18]: import matplotlib.pyplot as plt
         %matplotlib inline
         import seaborn as sns
         from sklearn.metrics import confusion_matrix

         # Final evaluation of the model
```

```
scores = model.evaluate(X_test, Y_test, verbose=0)
print("Test Score: %f" % (scores[0]))
print("Test Accuracy: %f%%" % (scores[1]*100))

# Confusion Matrix
Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_test, axis=1)])
Y_predictions = pd.Series([ACTIVITIES[y] for y in np.argmax(model.predict(X_test), ax

# Code for drawing seaborn heatmaps
class_names = ['laying','sitting','standing','walking','walking_downstairs','walking_
df_heatmap = pd.DataFrame(confusion_matrix(Y_true, Y_predictions), index=class_names,
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right',
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=90, ha='right',
plt.ylabel('True label',size=18)
plt.xlabel('Predicted label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```

```
Test Score: 0.488270
Test Accuracy: 88.293180%
```



- With a simple 2 layer architecture we got 88.29% accuracy and a loss of 0.488
- We can further imporve the performace with Hyperparameter tuning

2 (2) Model having 1 LSTM layer with 48 LSTM Units and 'adam' as an optimizer

```
In [19]: # Initiliazing the sequential model
         model1 = Sequential()
         # Configuring the parameters
         model1.add(LSTM(48, input_shape=(timesteps, input_dim)))
         # Adding a dropout layer
         model1.add(Dropout(0.5))
         # Adding a dense output layer with sigmoid activation
         model1.add(Dense(n_classes, activation='sigmoid'))
         print(model1.summary())
```

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

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

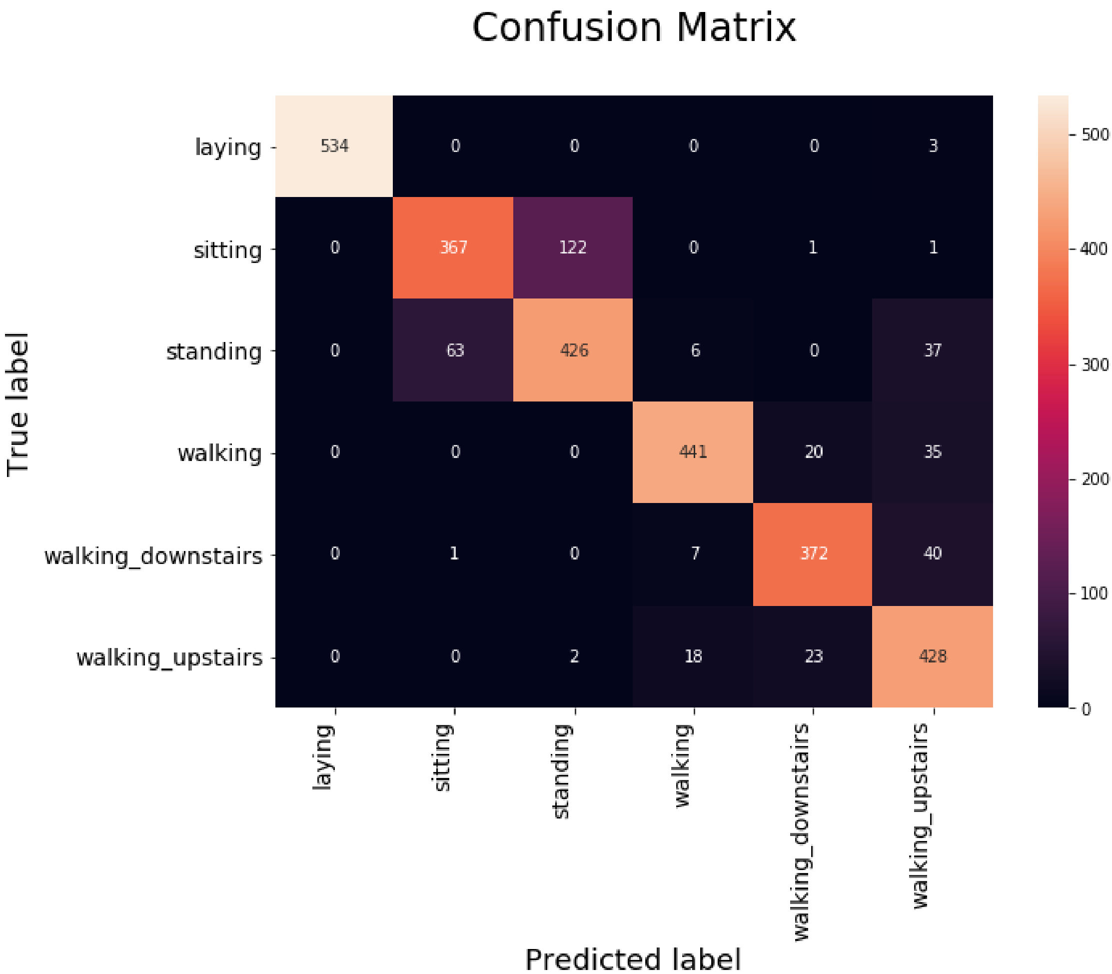
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 48)                11136
_________________________________________________________________
dropout_2 (Dropout)          (None, 48)                0
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 294
=================================================================
Total params: 11,430
Trainable params: 11,430
Non-trainable params: 0
_________________________________________________________________
None
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 45s 6ms/step - loss: 1.4210 - acc: 0.3677 - val_lo
Epoch 2/30
7352/7352 [==============================] - 43s 6ms/step - loss: 1.3615 - acc: 0.3659 - val_lo
Epoch 3/30
7352/7352 [==============================] - 43s 6ms/step - loss: 1.2965 - acc: 0.4147 - val_lo
Epoch 4/30
7352/7352 [==============================] - 43s 6ms/step - loss: 1.2413 - acc: 0.4645 - val_lo
Epoch 5/30
7352/7352 [==============================] - 43s 6ms/step - loss: 1.1199 - acc: 0.5102 - val_lo
Epoch 6/30
7352/7352 [==============================] - 42s 6ms/step - loss: 1.0028 - acc: 0.5439 - val_lo
Epoch 7/30
7352/7352 [==============================] - 43s 6ms/step - loss: 1.0453 - acc: 0.5098 - val_lo
Epoch 8/30
7352/7352 [==============================] - 43s 6ms/step - loss: 1.1810 - acc: 0.4523 - val_lo
Epoch 9/30
7352/7352 [==============================] - 43s 6ms/step - loss: 1.2428 - acc: 0.4329 - val_lo
Epoch 10/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.9496 - acc: 0.5747 - val_lo
Epoch 11/30
7352/7352 [==============================] - 43s 6ms/step - loss: 1.0623 - acc: 0.5399 - val_lo
Epoch 12/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.8686 - acc: 0.6114 - val_lo
Epoch 13/30
7352/7352 [==============================] - 42s 6ms/step - loss: 1.0787 - acc: 0.4974 - val_lo
Epoch 14/30
```

```
7352/7352 [==============================] - 43s 6ms/step - loss: 0.9513 - acc: 0.5822 - val_lo
Epoch 15/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.8773 - acc: 0.5929 - val_lo
Epoch 16/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.7541 - acc: 0.6250 - val_lo
Epoch 17/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.7139 - acc: 0.6499 - val_lo
Epoch 18/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.7097 - acc: 0.6468 - val_lo
Epoch 19/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.6794 - acc: 0.6575 - val_lo
Epoch 20/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.6810 - acc: 0.6553 - val_lo
Epoch 21/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.6905 - acc: 0.6468 - val_lo
Epoch 22/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.6648 - acc: 0.6712 - val_lo
Epoch 23/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.6891 - acc: 0.6727 - val_lo
Epoch 24/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.6327 - acc: 0.7331 - val_lo
Epoch 25/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.5341 - acc: 0.8074 - val_lo
Epoch 26/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.3767 - acc: 0.8696 - val_lo
Epoch 27/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.3015 - acc: 0.9015 - val_lo
Epoch 28/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.3263 - acc: 0.8989 - val_lo
Epoch 29/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.3337 - acc: 0.8976 - val_lo
Epoch 30/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.2294 - acc: 0.9272 - val_lo
```

In [20]:
```python
# Final evaluation of the model
scores1 = model1.evaluate(X_test, Y_test, verbose=0)
print("Test Score: %f" % (scores1[0]))
print("Test Accuracy: %f%%" % (scores1[1]*100))

# Confusion Matrix
Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_test, axis=1)])
Y_predictions = pd.Series([ACTIVITIES[y] for y in np.argmax(model1.predict(X_test), ax

# Code for drawing seaborn heatmaps
class_names = ['laying','sitting','standing','walking','walking_downstairs','walking_u
df_heatmap = pd.DataFrame(confusion_matrix(Y_true, Y_predictions), index=class_names,
fig = plt.figure(figsize=(10,7))
```

```
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', f
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=90, ha='right',
plt.ylabel('True label',size=18)
plt.xlabel('Predicted label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```

```
Test Score: 0.344224
Test Accuracy: 87.139464%
```



3 (3) Model having 1 LSTM layer with 48 LSTM Units and 'rmsprop' as an optimizer

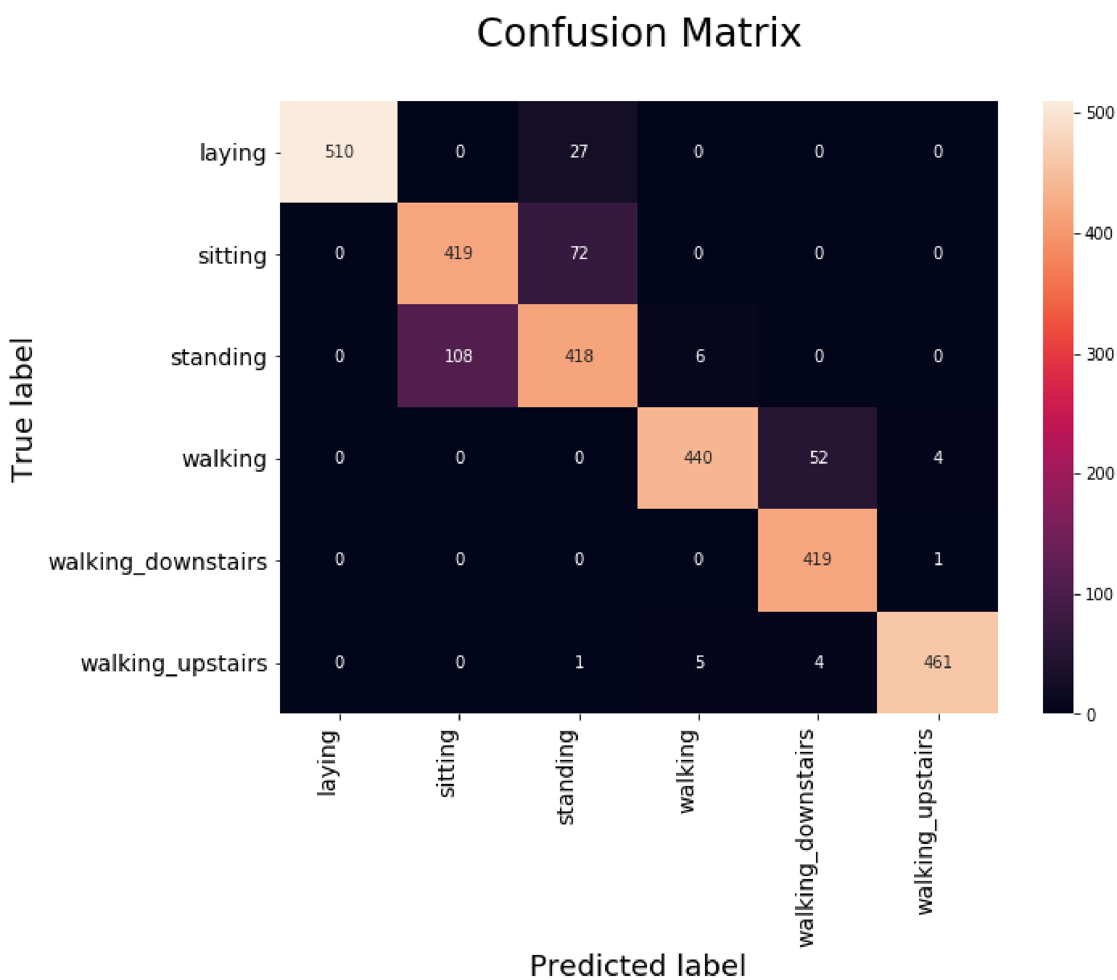
In [21]:
```
# Initiliazing the sequential model
model2 = Sequential()
# Configuring the parameters
model2.add(LSTM(48, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model2.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model2.add(Dense(n_classes, activation='sigmoid'))
print(model2.summary())

# Compiling the model
model2.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['accuracy

# Training the model
history2 = model2.fit(X_train,Y_train,batch_size=batch_size,validation_data=(X_test, Y
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 48)                11136
_________________________________________________________________
dropout_3 (Dropout)          (None, 48)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 294
=================================================================
Total params: 11,430
Trainable params: 11,430
Non-trainable params: 0
_________________________________________________________________
None
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 43s 6ms/step - loss: 1.2313 - acc: 0.4780 - val_lo
Epoch 2/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.8782 - acc: 0.6073 - val_lo
Epoch 3/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.7840 - acc: 0.6542 - val_lo
Epoch 4/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.6928 - acc: 0.6900 - val_lo
Epoch 5/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.6225 - acc: 0.7348 - val_lo
Epoch 6/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.5056 - acc: 0.8290 - val_lo
Epoch 7/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.3532 - acc: 0.8900 - val_lo
```

```
Epoch 8/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.2994 - acc: 0.9113 - val_lo
Epoch 9/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2638 - acc: 0.9212 - val_lo
Epoch 10/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.2297 - acc: 0.9276 - val_lo
Epoch 11/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2190 - acc: 0.9336 - val_lo
Epoch 12/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2153 - acc: 0.9329 - val_lo
Epoch 13/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2055 - acc: 0.9376 - val_lo
Epoch 14/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.1898 - acc: 0.9366 - val_lo
Epoch 15/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2032 - acc: 0.9319 - val_lo
Epoch 16/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.1801 - acc: 0.9416 - val_lo
Epoch 17/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.1810 - acc: 0.9423 - val_lo
Epoch 18/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.1714 - acc: 0.9452 - val_lo
Epoch 19/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1654 - acc: 0.9411 - val_lo
Epoch 20/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1795 - acc: 0.9455 - val_lo
Epoch 21/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1676 - acc: 0.9404 - val_lo
Epoch 22/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1811 - acc: 0.9423 - val_lo
Epoch 23/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1563 - acc: 0.9449 - val_lo
Epoch 24/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.1495 - acc: 0.9449 - val_lo
Epoch 25/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.1740 - acc: 0.9436 - val_lo
Epoch 26/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1564 - acc: 0.9446 - val_lo
Epoch 27/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1648 - acc: 0.9475 - val_lo
Epoch 28/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.1504 - acc: 0.9438 - val_lo
Epoch 29/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1501 - acc: 0.9468 - val_lo
Epoch 30/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1647 - acc: 0.9471 - val_lo
```

In [22]:
```python
# Final evaluation of the model
scores2 = model2.evaluate(X_test, Y_test, verbose=0)
print("Test Score: %f" % (scores2[0]))
print("Test Accuracy: %f%%" % (scores2[1]*100))

# Confusion Matrix
Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_test, axis=1)])
Y_predictions = pd.Series([ACTIVITIES[y] for y in np.argmax(model2.predict(X_test), ax

# Code for drawing seaborn heatmaps
class_names = ['laying','sitting','standing','walking','walking_downstairs','walking_u
df_heatmap = pd.DataFrame(confusion_matrix(Y_true, Y_predictions), index=class_names,
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', f
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=90, ha='right',
plt.ylabel('True label',size=18)
plt.xlabel('Predicted label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```

```
Test Score: 0.410484
Test Accuracy: 90.498812%
```



4 (4) Model having 1 LSTM layer with 64 LSTM Units and 'rmsprop' as an optimizer

```
In [23]: # Initiliazing the sequential model
         model3 = Sequential()
         # Configuring the parameters
         model3.add(LSTM(64, input_shape=(timesteps, input_dim)))
         # Adding a dropout layer
         model3.add(Dropout(0.5))
         # Adding a dense output layer with sigmoid activation
         model3.add(Dense(n_classes, activation='sigmoid'))
         print(model3.summary())

         # Compiling the model
         model3.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['accuracy
```

```python
# Training the model
history3 = model3.fit(X_train,Y_train,batch_size=batch_size,validation_data=(X_test, Y
```

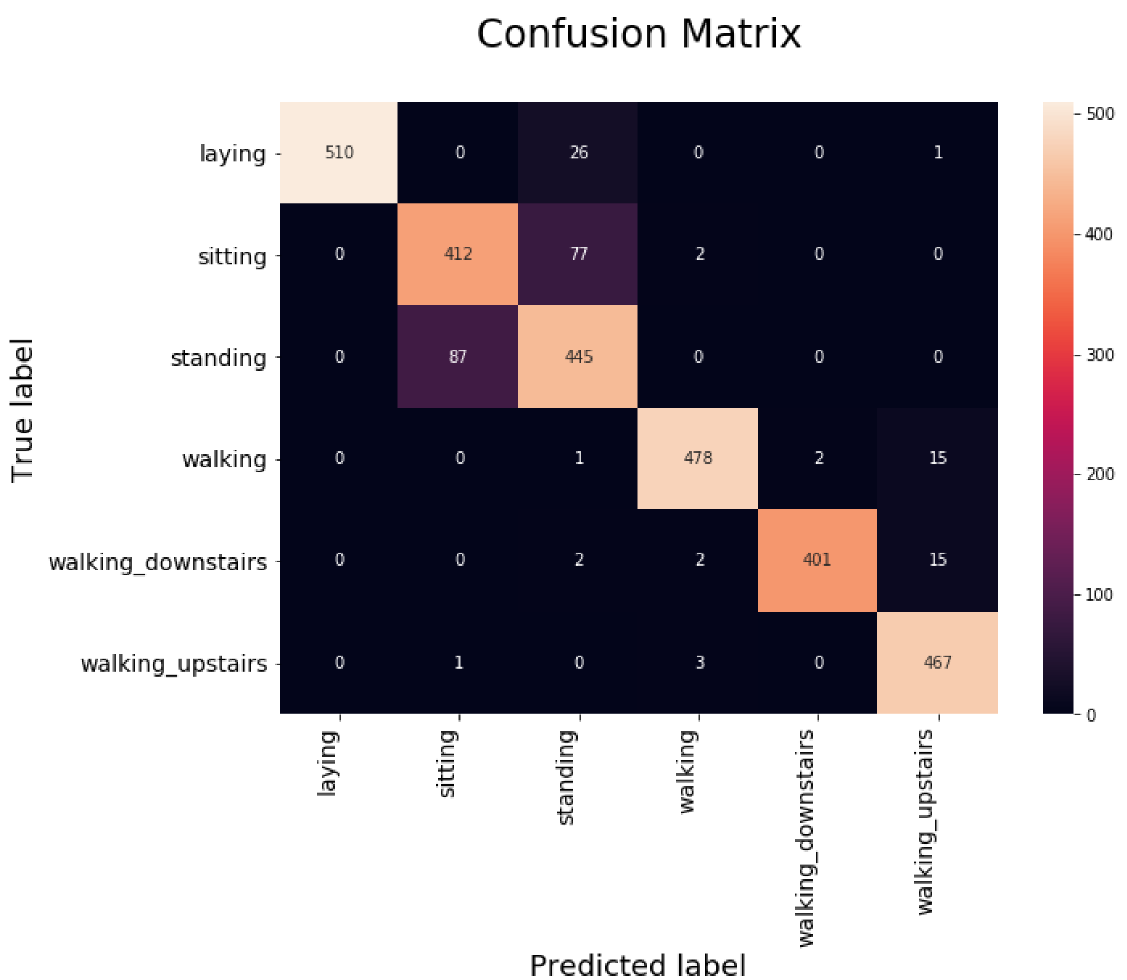
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 64)                18944
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 390
=================================================================
Total params: 19,334
Trainable params: 19,334
Non-trainable params: 0
_________________________________________________________________
None
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 47s 6ms/step - loss: 1.2746 - acc: 0.4457 - val_lo
Epoch 2/30
7352/7352 [==============================] - 46s 6ms/step - loss: 0.9587 - acc: 0.6020 - val_lo
Epoch 3/30
7352/7352 [==============================] - 46s 6ms/step - loss: 1.0225 - acc: 0.5890 - val_lo
Epoch 4/30
7352/7352 [==============================] - 46s 6ms/step - loss: 0.7561 - acc: 0.6812 - val_lo
Epoch 5/30
7352/7352 [==============================] - 46s 6ms/step - loss: 0.6203 - acc: 0.7402 - val_lo
Epoch 6/30
7352/7352 [==============================] - 58s 8ms/step - loss: 0.4874 - acc: 0.8249 - val_lo
Epoch 7/30
7352/7352 [==============================] - 48s 7ms/step - loss: 0.3588 - acc: 0.8905 - val_lo
Epoch 8/30
7352/7352 [==============================] - 52s 7ms/step - loss: 0.2826 - acc: 0.9042 - val_lo
Epoch 9/30
7352/7352 [==============================] - 49s 7ms/step - loss: 0.2855 - acc: 0.9033 - val_lo
Epoch 10/30
7352/7352 [==============================] - 48s 7ms/step - loss: 0.2367 - acc: 0.9197 - val_lo
Epoch 11/30
7352/7352 [==============================] - 48s 6ms/step - loss: 0.2891 - acc: 0.9064 - val_lo
Epoch 12/30
7352/7352 [==============================] - 48s 7ms/step - loss: 0.2101 - acc: 0.9327 - val_lo
Epoch 13/30
7352/7352 [==============================] - 49s 7ms/step - loss: 0.1883 - acc: 0.9309 - val_lo
Epoch 14/30
7352/7352 [==============================] - 48s 6ms/step - loss: 0.1781 - acc: 0.9354 - val_lo
Epoch 15/30
7352/7352 [==============================] - 47s 6ms/step - loss: 0.1812 - acc: 0.9344 - val_lo
```

```
Epoch 16/30
7352/7352 [==============================] - 48s 6ms/step - loss: 0.1701 - acc: 0.9414 - val_lo
Epoch 17/30
7352/7352 [==============================] - 48s 6ms/step - loss: 0.1603 - acc: 0.9446 - val_lo
Epoch 18/30
7352/7352 [==============================] - 48s 7ms/step - loss: 0.1494 - acc: 0.9460 - val_lo
Epoch 19/30
7352/7352 [==============================] - 47s 6ms/step - loss: 0.1555 - acc: 0.9445 - val_lo
Epoch 20/30
7352/7352 [==============================] - 47s 6ms/step - loss: 0.1413 - acc: 0.9498 - val_lo
Epoch 21/30
7352/7352 [==============================] - 47s 6ms/step - loss: 0.1674 - acc: 0.9444 - val_lo
Epoch 22/30
7352/7352 [==============================] - 47s 6ms/step - loss: 0.1550 - acc: 0.9430 - val_lo
Epoch 23/30
7352/7352 [==============================] - 47s 6ms/step - loss: 0.1551 - acc: 0.9450 - val_lo
Epoch 24/30
7352/7352 [==============================] - 46s 6ms/step - loss: 0.1679 - acc: 0.9440 - val_lo
Epoch 25/30
7352/7352 [==============================] - 47s 6ms/step - loss: 0.1543 - acc: 0.9472 - val_lo
Epoch 26/30
7352/7352 [==============================] - 47s 6ms/step - loss: 0.1457 - acc: 0.9459 - val_lo
Epoch 27/30
7352/7352 [==============================] - 47s 6ms/step - loss: 0.1383 - acc: 0.9476 - val_lo
Epoch 28/30
7352/7352 [==============================] - 46s 6ms/step - loss: 0.1412 - acc: 0.9508 - val_lo
Epoch 29/30
7352/7352 [==============================] - 47s 6ms/step - loss: 0.1496 - acc: 0.9464 - val_lo
Epoch 30/30
7352/7352 [==============================] - 47s 6ms/step - loss: 0.1439 - acc: 0.9490 - val_lo
```

In [24]:
```
# Final evaluation of the model
scores3 = model3.evaluate(X_test, Y_test, verbose=0)
print("Test Score: %f" % (scores3[0]))
print("Test Accuracy: %f%%" % (scores3[1]*100))

# Confusion Matrix
Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_test, axis=1)])
Y_predictions = pd.Series([ACTIVITIES[y] for y in np.argmax(model3.predict(X_test), a

# Code for drawing seaborn heatmaps
class_names = ['laying','sitting','standing','walking','walking_downstairs','walking_
df_heatmap = pd.DataFrame(confusion_matrix(Y_true, Y_predictions), index=class_names,
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
```

```
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', 
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=90, ha='right', 
plt.ylabel('True label',size=18)
plt.xlabel('Predicted label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```

```
Test Score: 0.299268
Test Accuracy: 92.059722%
```



5 (5) Model having 2 LSTM layer with 32 LSTM Units and 'rmsprop' as an optimizer

In [25]:
```
# Initiliazing the sequential model
model4 = Sequential()
```

```
# Configuring the parameters
model4.add(LSTM(32,return_sequences=True, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model4.add(Dropout(0.5))

# Configuring the parameters
model4.add(LSTM(32))
# Adding a dropout layer
model4.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model4.add(Dense(n_classes, activation='sigmoid'))
print(model4.summary())

# Compiling the model
model4.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['accuracy

# Training the model
history4 = model4.fit(X_train,Y_train,batch_size=batch_size,validation_data=(X_test, Y
```

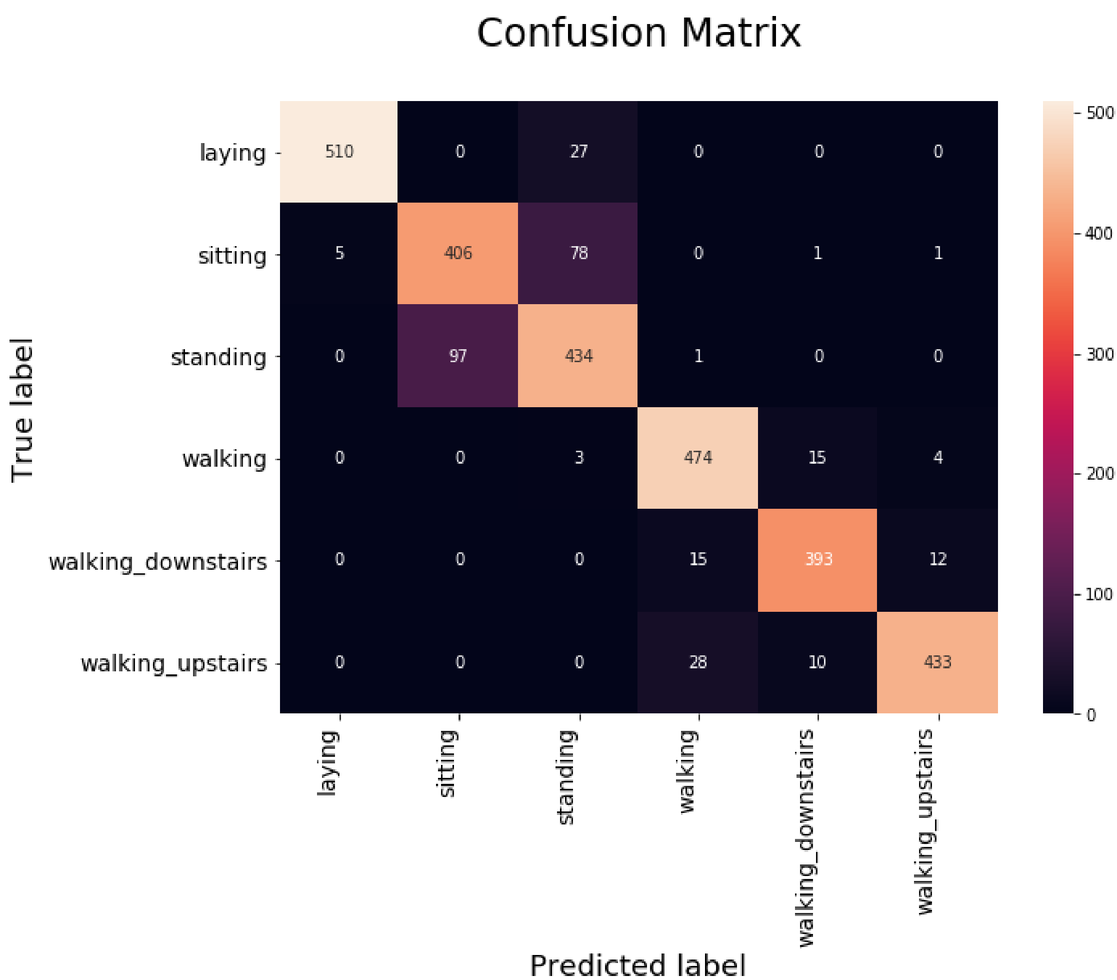
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 128, 32)           5376
_________________________________________________________________
dropout_5 (Dropout)          (None, 128, 32)           0
_________________________________________________________________
lstm_6 (LSTM)                (None, 32)                8320
_________________________________________________________________
dropout_6 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 198
=================================================================
Total params: 13,894
Trainable params: 13,894
Non-trainable params: 0
_________________________________________________________________
None
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 86s 12ms/step - loss: 1.2107 - acc: 0.5061 - val_l
Epoch 2/30
7352/7352 [==============================] - 84s 11ms/step - loss: 0.7991 - acc: 0.6766 - val_l
Epoch 3/30
7352/7352 [==============================] - 83s 11ms/step - loss: 0.6209 - acc: 0.7542 - val_l
Epoch 4/30
7352/7352 [==============================] - 83s 11ms/step - loss: 0.4968 - acc: 0.7802 - val_l
Epoch 5/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.4323 - acc: 0.8009 - val_l
```

```
Epoch 6/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.4538 - acc: 0.8247 - val_
Epoch 7/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.3524 - acc: 0.8785 - val_
Epoch 8/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.3199 - acc: 0.9115 - val_
Epoch 9/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.2609 - acc: 0.9285 - val_
Epoch 10/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.2290 - acc: 0.9339 - val_
Epoch 11/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.2160 - acc: 0.9353 - val_
Epoch 12/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.2236 - acc: 0.9321 - val_
Epoch 13/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1718 - acc: 0.9455 - val_
Epoch 14/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1740 - acc: 0.9359 - val_
Epoch 15/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1693 - acc: 0.9423 - val_
Epoch 16/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1813 - acc: 0.9459 - val_
Epoch 17/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1687 - acc: 0.9472 - val_
Epoch 18/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1557 - acc: 0.9474 - val_
Epoch 19/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1479 - acc: 0.9471 - val_
Epoch 20/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1479 - acc: 0.9493 - val_
Epoch 21/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1465 - acc: 0.9509 - val_
Epoch 22/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1508 - acc: 0.9508 - val_
Epoch 23/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1512 - acc: 0.9489 - val_
Epoch 24/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1434 - acc: 0.9513 - val_
Epoch 25/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1805 - acc: 0.9414 - val_
Epoch 26/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1453 - acc: 0.9528 - val_
Epoch 27/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1385 - acc: 0.9520 - val_
Epoch 28/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1420 - acc: 0.9533 - val_
Epoch 29/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1288 - acc: 0.9547 - val_
```

```
Epoch 30/30
7352/7352 [==============================] - 82s 11ms/step - loss: 0.1291 - acc: 0.9532 - val_l
```

```
In [26]: # Final evaluation of the model
         scores4 = model4.evaluate(X_test, Y_test, verbose=0)
         print("Test Score: %f" % (scores4[0]))
         print("Test Accuracy: %f%%" % (scores4[1]*100))

         # Confusion Matrix
         Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_test, axis=1)])
         Y_predictions = pd.Series([ACTIVITIES[y] for y in np.argmax(model4.predict(X_test), a

         # Code for drawing seaborn heatmaps
         class_names = ['laying','sitting','standing','walking','walking_downstairs','walking_u
         df_heatmap = pd.DataFrame(confusion_matrix(Y_true, Y_predictions), index=class_names,
         fig = plt.figure(figsize=(10,7))
         heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

         # Setting tick labels for heatmap
         heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', f
         heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=90, ha='right',
         plt.ylabel('True label',size=18)
         plt.xlabel('Predicted label',size=18)
         plt.title("Confusion Matrix\n",size=24)
         plt.show()
```

```
Test Score: 0.545492
Test Accuracy: 89.921955%
```



6 (6) Model having 2 LSTM layer with 64 LSTM Units and 'rmsprop' as an optimizer

In [27]:
```
# Initiliazing the sequential model
model5 = Sequential()
# Configuring the parameters
model5.add(LSTM(64,return_sequences=True, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model5.add(Dropout(0.7))

# Configuring the parameters
model5.add(LSTM(64))
# Adding a dropout layer
model5.add(Dropout(0.7))
# Adding a dense output layer with sigmoid activation
model5.add(Dense(n_classes, activation='sigmoid'))
```

```
print(model5.summary())

# Compiling the model
model5.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['accuracy

# Training the model
history5 = model5.fit(X_train,Y_train,batch_size=batch_size,validation_data=(X_test, Y
```

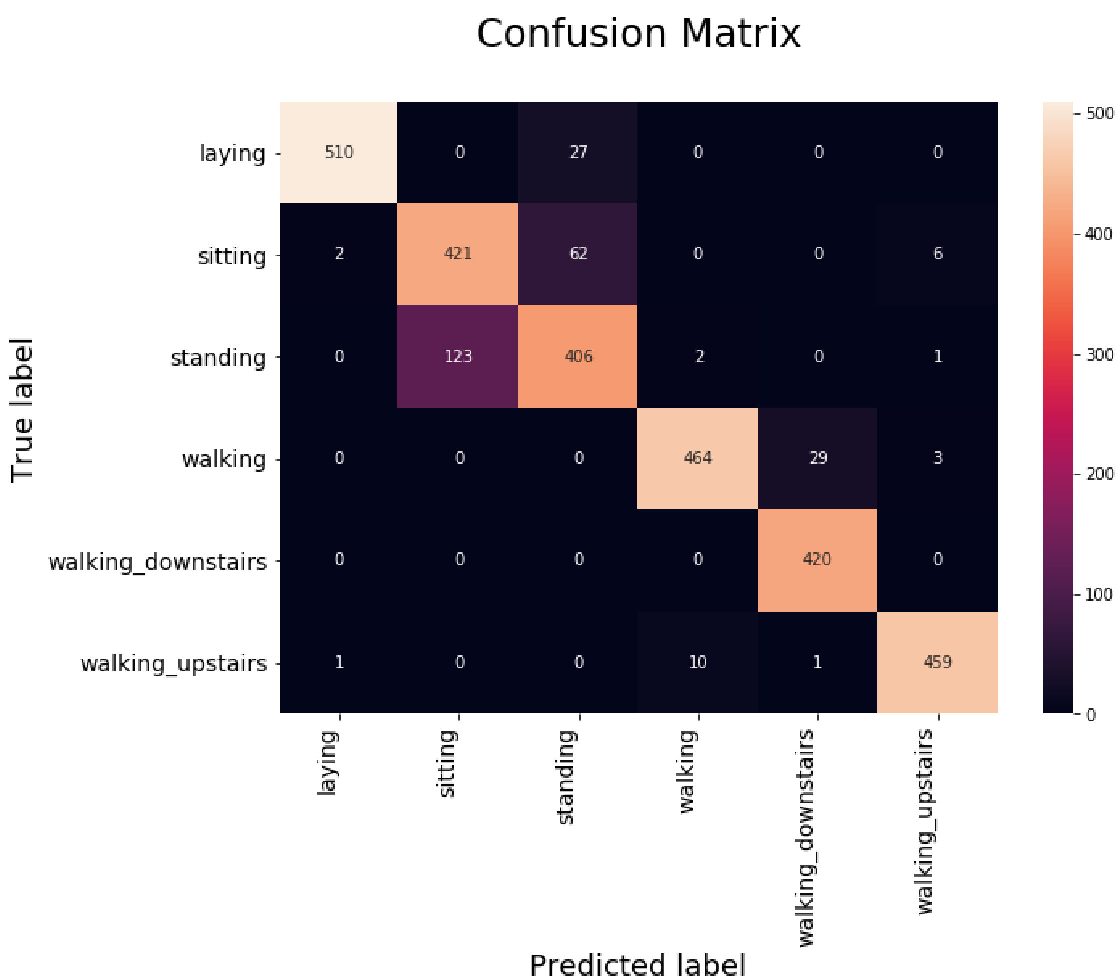
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_7 (LSTM)                (None, 128, 64)           18944
_________________________________________________________________
dropout_7 (Dropout)          (None, 128, 64)           0
_________________________________________________________________
lstm_8 (LSTM)                (None, 64)                33024
_________________________________________________________________
dropout_8 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 6)                 390
=================================================================
Total params: 52,358
Trainable params: 52,358
Non-trainable params: 0
_________________________________________________________________
None
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 110s 15ms/step - loss: 1.1611 - acc: 0.4890 - val_
Epoch 2/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.8021 - acc: 0.6549 - val_
Epoch 3/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.7392 - acc: 0.6670 - val_
Epoch 4/30
7352/7352 [==============================] - 109s 15ms/step - loss: 0.6489 - acc: 0.7338 - val_
Epoch 5/30
7352/7352 [==============================] - 107s 15ms/step - loss: 0.5545 - acc: 0.7625 - val_
Epoch 6/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.4380 - acc: 0.8164 - val_
Epoch 7/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.3323 - acc: 0.8966 - val_
Epoch 8/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.2380 - acc: 0.9301 - val_
Epoch 9/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.2048 - acc: 0.9370 - val_
Epoch 10/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1991 - acc: 0.9389 - val_
Epoch 11/30
```

```
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1775 - acc: 0.9429 - val_
Epoch 12/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1724 - acc: 0.9436 - val_
Epoch 13/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1628 - acc: 0.9453 - val_
Epoch 14/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1754 - acc: 0.9440 - val_
Epoch 15/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1556 - acc: 0.9465 - val_
Epoch 16/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1733 - acc: 0.9452 - val_
Epoch 17/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1737 - acc: 0.9448 - val_
Epoch 18/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1698 - acc: 0.9442 - val_
Epoch 19/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1469 - acc: 0.9512 - val_
Epoch 20/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1492 - acc: 0.9461 - val_
Epoch 21/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1532 - acc: 0.9490 - val_
Epoch 22/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1758 - acc: 0.9436 - val_
Epoch 23/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1528 - acc: 0.9489 - val_
Epoch 24/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1589 - acc: 0.9438 - val_
Epoch 25/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1522 - acc: 0.9437 - val_
Epoch 26/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1363 - acc: 0.9495 - val_
Epoch 27/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1480 - acc: 0.9459 - val_
Epoch 28/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1347 - acc: 0.9495 - val_
Epoch 29/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1489 - acc: 0.9474 - val_
Epoch 30/30
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1491 - acc: 0.9486 - val_
```

In [28]:
```
# Final evaluation of the model
scores5 = model5.evaluate(X_test, Y_test, verbose=0)
print("Test Score: %f" % (scores5[0]))
print("Test Accuracy: %f%%" % (scores5[1]*100))

# Confusion Matrix
Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_test, axis=1)])
```

```
Y_predictions = pd.Series([ACTIVITIES[y] for y in np.argmax(model5.predict(X_test), a

# Code for drawing seaborn heatmaps
class_names = ['laying','sitting','standing','walking','walking_downstairs','walking_u
df_heatmap = pd.DataFrame(confusion_matrix(Y_true, Y_predictions), index=class_names,
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right',
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=90, ha='right',
plt.ylabel('True label',size=18)
plt.xlabel('Predicted label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```

```
Test Score: 0.412691
Test Accuracy: 90.939939%
```



7 CONCLUSION

8 (a). Procedure Followed :

STEP 1 :- Load the data and split into training_data and test_data
STEP 2:-Try out different LSTM architectures
STEP 3:- Find test score and accuracy for each model
STEP 4:- Draw confusion matrix using seaborn heatmap for each model

9 (b). Table (Model performances) :

In [29]:
```
# Creating table using PrettyTable library
from prettytable import PrettyTable

# Names of models
names =['1 LSTM layer with 32 LSTM Units(Optimizer-->rmsprop)','1 LSTM layer with 48 L
        '1 LSTM layer with 48 LSTM Units(Optimizer-->rmsprop)','1 LSTM layer with 64 L
        '2 LSTM layer with 32 LSTM Units(Optimizer-->rmsprop)','2 LSTM layer with 64 L

# Training accuracies
train_acc = [history.history['acc'][29],history1.history['acc'][29],history2.history[
             history3.history['acc'][29],history4.history['acc'][29],history5.history

# Test accuracies
test_acc =[scores[1],scores1[1],scores2[1],scores3[1],scores4[1],scores5[1]]

numbering = [1,2,3,4,5,6]

# Initializing prettytable
ptable = PrettyTable()

# Adding columns
ptable.add_column("S.NO.",numbering)
ptable.add_column("MODEL",names)
ptable.add_column("Training Accuracy",train_acc)
ptable.add_column("Test Accuracy",test_acc)

# Printing the Table
print(ptable)
```

```
+-------+------------------------------------------------------+--------------------+---------
| S.NO. |                        MODEL                         | Training Accuracy  |   Test Ac
+-------+------------------------------------------------------+--------------------+---------
|   1   | 1 LSTM layer with 32 LSTM Units(Optimizer-->rmsprop) | 0.9411044613710555 | 0.882931
```

2	1 LSTM layer with 48 LSTM Units(Optimizer-->adam)	0.9272306855277476	0.871394
3	1 LSTM layer with 48 LSTM Units(Optimizer-->rmsprop)	0.9470892274211099	0.904988
4	1 LSTM layer with 64 LSTM Units(Optimizer-->rmsprop)	0.948993471164309	0.920597
5	2 LSTM layer with 32 LSTM Units(Optimizer-->rmsprop)	0.9532100108813928	0.899219
6	2 LSTM layer with 64 LSTM Units(Optimizer-->rmsprop)	0.9485854189336235	0.909399