In [1]:
```
# Importing Libraries
```

In [2]:
```
import pandas as pd
import numpy as np
```

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',
}

# 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 [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}_{subse
t}.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 signa
ls)
    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_dummie
s.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 [22]:
```
# Initializing parameters
epochs = 20
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 [17]:
```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

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

In [18]:
```
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 [ ]:
```
#http://maxpumperla.com/hyperas/
'''
from __future__ import print_function
import numpy as np

from hyperopt import Trials, STATUS_OK, tpe
from keras.datasets import mnist
from keras.layers.core import Dense, Dropout, Activation
from keras.models import Sequential
from keras.utils import np_utils

from hyperas import optim
from hyperas.distributions import choice, uniform


def create_model(X_train, y_train, X_test, y_test):

    epochs = 8
    batch_size = 32
    timesteps = x_train.shape[1]
    input_dim = len(x_train[0][0])
    n_classes = 6

    model = Sequential()

    model.add(LSTM({{choice([64,32, 16])}}, return_sequences = True, input_sha
pe = (timesteps, input_dim)))
    model.add(Dropout({{uniform(0, 1)}}))

    model.add(LSTM({{choice([32, 16])}}))
    model.add(Dropout({{uniform(0, 1)}}))

    model.add(Dense(n_classes, activation='sigmoid'))

    print(model.summary())

    model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optim
izer='rmsprop')

    result = model.fit(X_train, y_train, batch_size = batch_size, epochs=epoch
s, verbose=2, validation_split=0.01)

    validation_acc = np.amax(result.history['val_acc'])

    print('Best validation acc of epoch:', validation_acc)

    return {'loss': -validation_acc, 'status': STATUS_OK, 'model': model}'''
```

In [ ]:
```
'''best_run, best_model = optim.minimize(model=create_model, data=load_data(),
algo=tpe.suggest, max_evals=4, trials=Trials())
X_train, y_train, X_test, y_test = load_data()

score = best_model.evaluate(X_test, y_test)

print('---------------------')
print('|      Accuracy      |')
print('---------------------')
acc = np.round((score[1]*100), 2)
print(str(acc)+"%\n")

print('----------------------------------')
print('|      Best Hyper-Parameters      |')
print('----------------------------------')
print(best_run)
print("\n\n")

true_labels = [np.argmax(i)+1 for i in y_test]
predicted_probs = best_model.predict(X_test)
predicted_labels = [np.argmax(i)+1 for i in predicted_probs]
print_confusionMatrix(true_labels, predicted_labels)'''
```

Model1: 1 LSTM with 32 hidden unit , rmsprop optimizer

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

```
WARNING: Logging before flag parsing goes to stderr.
W0626 22:24:19.299101  6868 deprecation_wrapper.py:119] From C:\Users\Raftaar
Singh\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:74: The
name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_
graph instead.

W0626 22:24:19.305118  6868 deprecation_wrapper.py:119] From C:\Users\Raftaar
Singh\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:517: Th
e name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder inst
ead.

W0626 22:24:19.308441  6868 deprecation_wrapper.py:119] From C:\Users\Raftaar
Singh\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:4138: T
he name tf.random_uniform is deprecated. Please use tf.random.uniform instea
d.

W0626 22:24:19.699645  6868 deprecation_wrapper.py:119] From C:\Users\Raftaar
Singh\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:133: Th
e name tf.placeholder_with_default is deprecated. Please use tf.compat.v1.pla
ceholder_with_default instead.

W0626 22:24:19.712710  6868 deprecation.py:506] From C:\Users\Raftaar Singh\A
naconda3\lib\site-packages\keras\backend\tensorflow_backend.py:3445: calling
dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and
will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - k
eep_prob`.

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 32)                5376
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 198
=================================================================
Total params: 5,574
Trainable params: 5,574
Non-trainable params: 0
_________________________________________________________________
```

In [20]:
```
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

```
W0626 22:24:23.850056  6868 deprecation_wrapper.py:119] From C:\Users\Raftaar
Singh\Anaconda3\lib\site-packages\keras\optimizers.py:790: The name tf.train.
Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

W0626 22:24:23.889933  6868 deprecation_wrapper.py:119] From C:\Users\Raftaar
Singh\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:3295: T
he name tf.log is deprecated. Please use tf.math.log instead.
```

In [21]:
```
# Training the model
model.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)
```

```
W0626 22:24:25.472001  6868 deprecation.py:323] From C:\Users\Raftaar Singh\A
naconda3\lib\site-packages\tensorflow\python\ops\math_grad.py:1250: add_dispa
tch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprec
ated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

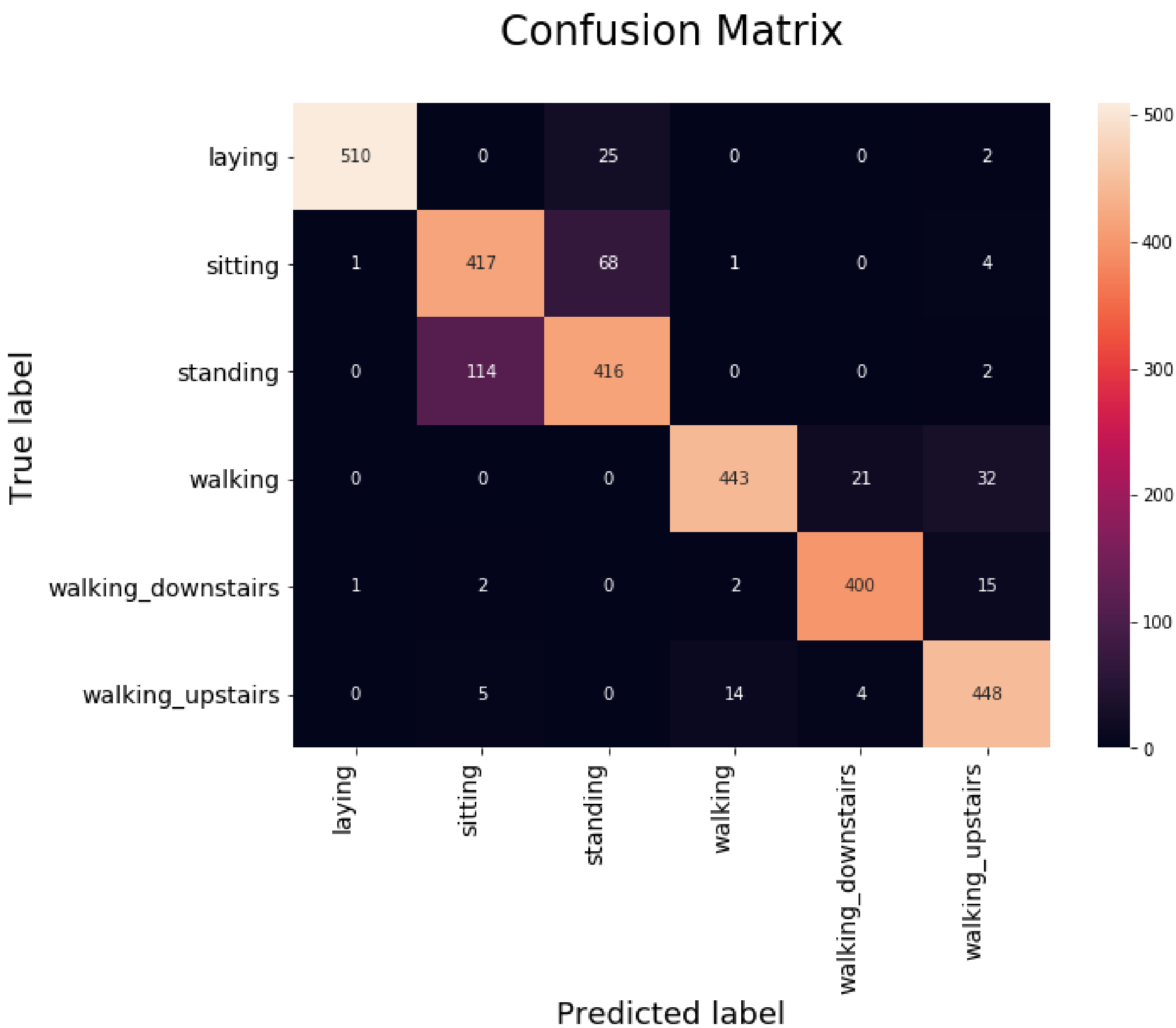
```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 77s 10ms/step - loss: 1.3207 - a
cc: 0.4329 - val_loss: 1.1474 - val_acc: 0.4706
Epoch 2/30
7352/7352 [==============================] - 72s 10ms/step - loss: 0.9877 - a
cc: 0.5702 - val_loss: 1.0286 - val_acc: 0.5046
Epoch 3/30
7352/7352 [==============================] - 74s 10ms/step - loss: 0.7947 - a
cc: 0.6477 - val_loss: 0.7681 - val_acc: 0.6074
Epoch 4/30
7352/7352 [==============================] - 73s 10ms/step - loss: 0.6952 - a
cc: 0.6578 - val_loss: 0.7221 - val_acc: 0.6060
Epoch 5/30
7352/7352 [==============================] - 73s 10ms/step - loss: 0.6491 - a
cc: 0.6802 - val_loss: 0.7290 - val_acc: 0.6169
Epoch 6/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.6374 - ac
c: 0.6857 - val_loss: 1.2811 - val_acc: 0.5877
Epoch 7/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.6218 - ac
c: 0.7231 - val_loss: 0.6598 - val_acc: 0.7194
Epoch 8/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.5557 - ac
c: 0.7578 - val_loss: 0.7062 - val_acc: 0.7333
Epoch 9/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.5000 - ac
c: 0.7933 - val_loss: 0.6462 - val_acc: 0.7618
Epoch 10/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.4565 - ac
c: 0.8069 - val_loss: 0.5785 - val_acc: 0.7737
Epoch 11/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.4371 - ac
c: 0.8171 - val_loss: 0.5561 - val_acc: 0.7788
Epoch 12/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.3866 - ac
c: 0.8437 - val_loss: 0.5607 - val_acc: 0.8273
Epoch 13/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.3693 - ac
c: 0.8819 - val_loss: 0.5133 - val_acc: 0.8687
Epoch 14/30
7352/7352 [==============================] - 66s 9ms/step - loss: 0.3033 - ac
c: 0.9106 - val_loss: 0.5015 - val_acc: 0.8799
Epoch 15/30
7352/7352 [==============================] - 66s 9ms/step - loss: 0.2572 - ac
c: 0.9226 - val_loss: 0.4646 - val_acc: 0.8823
Epoch 16/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.2539 - ac
c: 0.9232 - val_loss: 0.5301 - val_acc: 0.8826
Epoch 17/30
7352/7352 [==============================] - 61s 8ms/step - loss: 0.2158 - ac
c: 0.9331 - val_loss: 0.5799 - val_acc: 0.8612
Epoch 18/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.2564 - ac
c: 0.9244 - val_loss: 0.5184 - val_acc: 0.8680
Epoch 19/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.2175 - ac
```

```
c: 0.9319 - val_loss: 0.5122 - val_acc: 0.8870
Epoch 20/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.2374 - ac
c: 0.9321 - val_loss: 0.5969 - val_acc: 0.8711
Epoch 21/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.2009 - ac
c: 0.9392 - val_loss: 0.6558 - val_acc: 0.8714
Epoch 22/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.2106 - ac
c: 0.9387 - val_loss: 0.5078 - val_acc: 0.8782
Epoch 23/30
7352/7352 [==============================] - 62s 8ms/step - loss: 0.1870 - ac
c: 0.9411 - val_loss: 0.4839 - val_acc: 0.8812
Epoch 24/30
7352/7352 [==============================] - 61s 8ms/step - loss: 0.1881 - ac
c: 0.9399 - val_loss: 0.6952 - val_acc: 0.8721
Epoch 25/30
7352/7352 [==============================] - 62s 8ms/step - loss: 0.2000 - ac
c: 0.9391 - val_loss: 0.5929 - val_acc: 0.8856
Epoch 26/30
7352/7352 [==============================] - 61s 8ms/step - loss: 0.1904 - ac
c: 0.9412 - val_loss: 0.5378 - val_acc: 0.8823
Epoch 27/30
7352/7352 [==============================] - 61s 8ms/step - loss: 0.2367 - ac
c: 0.9358 - val_loss: 0.5179 - val_acc: 0.8697
Epoch 28/30
7352/7352 [==============================] - 61s 8ms/step - loss: 0.1854 - ac
c: 0.9468 - val_loss: 0.5344 - val_acc: 0.8931
Epoch 29/30
7352/7352 [==============================] - 61s 8ms/step - loss: 0.2251 - ac
c: 0.9402 - val_loss: 0.4379 - val_acc: 0.8918
Epoch 30/30
7352/7352 [==============================] - 61s 8ms/step - loss: 0.1702 - ac
c: 0.9457 - val_loss: 0.4320 - val_acc: 0.8938
```

Out[21]:
```
<keras.callbacks.History at 0x1c74c006a90>
```

In [23]:
```
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_te
st), axis=1)])

# Code for drawing seaborn heatmaps
class_names = ['laying','sitting','standing','walking','walking_downstairs','w
alking_upstairs']
df_heatmap = pd.DataFrame(confusion_matrix(Y_true, Y_predictions), index=class
_names, columns=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='r
ight', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=90, ha=
'right', fontsize=14)
plt.ylabel('True label',size=18)
plt.xlabel('Predicted label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```

```
Test Score: 0.431993
Test Accuracy: 89.379030%
```



- With a simple 1 layer architecture we got 89.34% accuracy and a loss of 0.43
- We can further imporve the performace with Hyperparameter tuning

Model2: 1 LSTM with 64 hodden unit , adam optimizer

In [24]:
```
# Initiliazing the sequential model
model2 = Sequential()
# Configuring the parameters
model2.add(LSTM(64, 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'))
model2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_2 (LSTM)                (None, 64)                18944     
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 390       
=================================================================
Total params: 19,334
Trainable params: 19,334
Non-trainable params: 0
_________________________________________________________________
```

In [25]:
```
# Compiling the model
model2.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Training the model
model2.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)
```

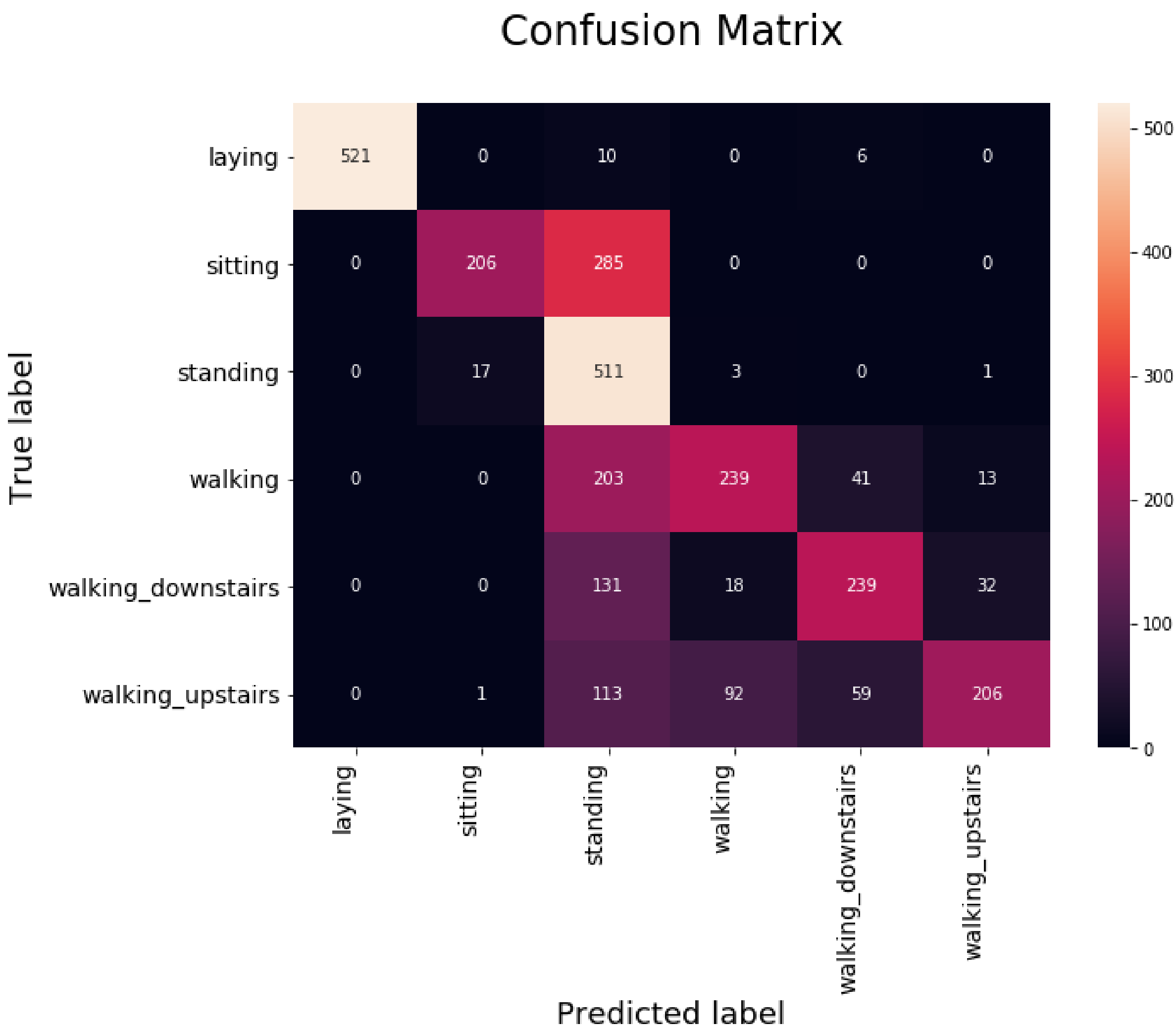
```
Train on 7352 samples, validate on 2947 samples
Epoch 1/20
7352/7352 [==============================] - 73s 10ms/step - loss: 1.4055 - a
cc: 0.3928 - val_loss: 1.2250 - val_acc: 0.4326
Epoch 2/20
7352/7352 [==============================] - 73s 10ms/step - loss: 1.2155 - a
cc: 0.4698 - val_loss: 1.2551 - val_acc: 0.5151
Epoch 3/20
7352/7352 [==============================] - 71s 10ms/step - loss: 1.0704 - a
cc: 0.5320 - val_loss: 1.0345 - val_acc: 0.4930
Epoch 4/20
7352/7352 [==============================] - 75s 10ms/step - loss: 1.0918 - a
cc: 0.5222 - val_loss: 1.2427 - val_acc: 0.5134
Epoch 5/20
7352/7352 [==============================] - 74s 10ms/step - loss: 1.1104 - a
cc: 0.5144 - val_loss: 0.8929 - val_acc: 0.5870
Epoch 6/20
7352/7352 [==============================] - 73s 10ms/step - loss: 0.8779 - a
cc: 0.6009 - val_loss: 0.8901 - val_acc: 0.5657
Epoch 7/20
7352/7352 [==============================] - 80s 11ms/step - loss: 0.8381 - a
cc: 0.6034 - val_loss: 0.9905 - val_acc: 0.5704
Epoch 8/20
7352/7352 [==============================] - 69s 9ms/step - loss: 0.8075 - ac
c: 0.6260 - val_loss: 0.9050 - val_acc: 0.5585
Epoch 9/20
7352/7352 [==============================] - 71s 10ms/step - loss: 0.8471 - a
cc: 0.6119 - val_loss: 0.8664 - val_acc: 0.5864
Epoch 10/20
7352/7352 [==============================] - 82s 11ms/step - loss: 0.7705 - a
cc: 0.6383 - val_loss: 0.7785 - val_acc: 0.6115
Epoch 11/20
7352/7352 [==============================] - 79s 11ms/step - loss: 0.7098 - a
cc: 0.6536 - val_loss: 0.7785 - val_acc: 0.5959
Epoch 12/20
7352/7352 [==============================] - 79s 11ms/step - loss: 0.6937 - a
cc: 0.6585 - val_loss: 0.8031 - val_acc: 0.6084
Epoch 13/20
7352/7352 [==============================] - 78s 11ms/step - loss: 0.8204 - a
cc: 0.6266 - val_loss: 0.7719 - val_acc: 0.6006
Epoch 14/20
7352/7352 [==============================] - 71s 10ms/step - loss: 1.1724 - a
cc: 0.4165 - val_loss: 0.9000 - val_acc: 0.5765
Epoch 15/20
7352/7352 [==============================] - 72s 10ms/step - loss: 0.9995 - a
cc: 0.4483 - val_loss: 0.9984 - val_acc: 0.4279
Epoch 16/20
7352/7352 [==============================] - 79s 11ms/step - loss: 0.9568 - a
cc: 0.5365 - val_loss: 0.9455 - val_acc: 0.6325
Epoch 17/20
7352/7352 [==============================] - 77s 10ms/step - loss: 0.7482 - a
cc: 0.6825 - val_loss: 0.6589 - val_acc: 0.7078
Epoch 18/20
7352/7352 [==============================] - 82s 11ms/step - loss: 0.7057 - a
cc: 0.7252 - val_loss: 0.7243 - val_acc: 0.7068
Epoch 19/20
7352/7352 [==============================] - 78s 11ms/step - loss: 0.6257 - a
```

```
cc: 0.7709 - val_loss: 0.5719 - val_acc: 0.7954
Epoch 20/20
7352/7352 [==============================] - 76s 10ms/step - loss: 0.8550 - a
cc: 0.6759 - val_loss: 0.9165 - val_acc: 0.6522
```

Out[25]:
```
<keras.callbacks.History at 0x1c752728470>
```

In [26]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Final evaluation of the model
scores = model2.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(model2.predict(X_t
est), axis=1)])

# Code for drawing seaborn heatmaps
class_names = ['laying','sitting','standing','walking','walking_downstairs','w
alking_upstairs']
df_heatmap = pd.DataFrame(confusion_matrix(Y_true, Y_predictions), index=class
_names, columns=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='r
ight', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=90, ha=
'right', fontsize=14)
plt.ylabel('True label',size=18)
plt.xlabel('Predicted label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```

```
Test Score: 0.916506
Test Accuracy: 65.218867%
```



it is seen that when i decrease opochs value from 30 to 20, accuracy decrease drastically.

loss also increase significantly in 1 LSTM model

Model3: 1 LSTM with 64 hodden unit , rmsprop optimizer

In [27]:
```
# 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'))
model3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_3 (LSTM)                (None, 64)                18944     
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 390       
=================================================================
Total params: 19,334
Trainable params: 19,334
Non-trainable params: 0
_________________________________________________________________
```

In [28]:
```
# Compiling the model
model3.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

# Training the model
model3.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)
```

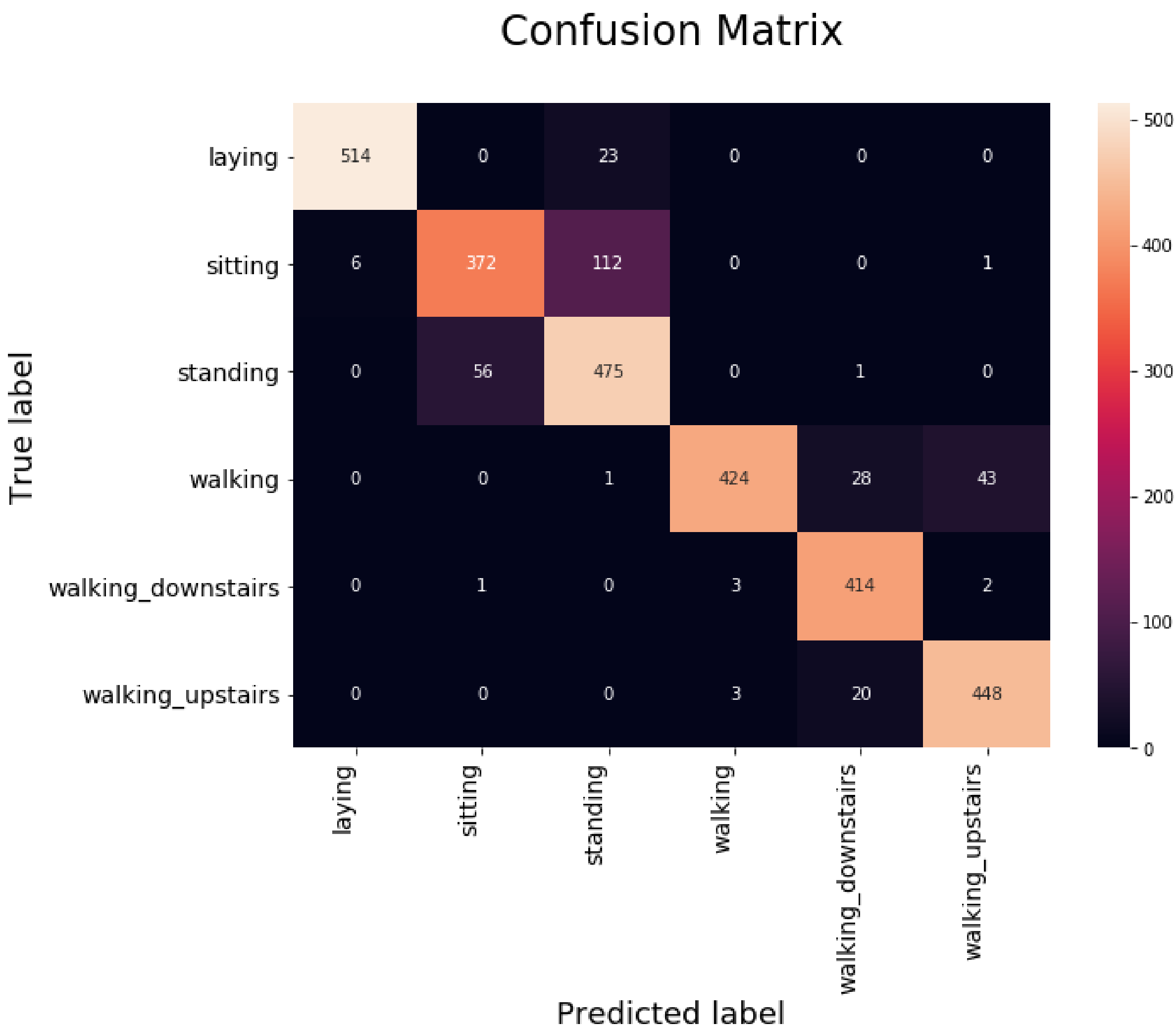
```
Train on 7352 samples, validate on 2947 samples
Epoch 1/20
7352/7352 [==============================] - 76s 10ms/step - loss: 1.2690 - a
cc: 0.4430 - val_loss: 1.0997 - val_acc: 0.5324
Epoch 2/20
7352/7352 [==============================] - 69s 9ms/step - loss: 0.9841 - ac
c: 0.5680 - val_loss: 0.8458 - val_acc: 0.6491
Epoch 3/20
7352/7352 [==============================] - 70s 10ms/step - loss: 0.8499 - a
cc: 0.6334 - val_loss: 1.0590 - val_acc: 0.5127
Epoch 4/20
7352/7352 [==============================] - 72s 10ms/step - loss: 0.6973 - a
cc: 0.7001 - val_loss: 0.7259 - val_acc: 0.7038
Epoch 5/20
7352/7352 [==============================] - 75s 10ms/step - loss: 0.5915 - a
cc: 0.7455 - val_loss: 0.5523 - val_acc: 0.7608
Epoch 6/20
7352/7352 [==============================] - 71s 10ms/step - loss: 0.4976 - a
cc: 0.8048 - val_loss: 0.5066 - val_acc: 0.8001
Epoch 7/20
7352/7352 [==============================] - 70s 10ms/step - loss: 0.3891 - a
cc: 0.8637 - val_loss: 0.4102 - val_acc: 0.8578
Epoch 8/20
7352/7352 [==============================] - 70s 10ms/step - loss: 0.2971 - a
cc: 0.9040 - val_loss: 0.8256 - val_acc: 0.7893
Epoch 9/20
7352/7352 [==============================] - 70s 10ms/step - loss: 0.2370 - a
cc: 0.9204 - val_loss: 0.4101 - val_acc: 0.8697
Epoch 10/20
7352/7352 [==============================] - 71s 10ms/step - loss: 0.2206 - a
cc: 0.9301 - val_loss: 0.5412 - val_acc: 0.8744
Epoch 11/20
7352/7352 [==============================] - 71s 10ms/step - loss: 0.2309 - a
cc: 0.9233 - val_loss: 0.6137 - val_acc: 0.8578
Epoch 12/20
7352/7352 [==============================] - 70s 10ms/step - loss: 0.1880 - a
cc: 0.9354 - val_loss: 0.5275 - val_acc: 0.8795
Epoch 13/20
7352/7352 [==============================] - 71s 10ms/step - loss: 0.2023 - a
cc: 0.9335 - val_loss: 0.4805 - val_acc: 0.8826
Epoch 14/20
7352/7352 [==============================] - 71s 10ms/step - loss: 0.1785 - a
cc: 0.9372 - val_loss: 0.3575 - val_acc: 0.8819
Epoch 15/20
7352/7352 [==============================] - 71s 10ms/step - loss: 0.1892 - a
cc: 0.9378 - val_loss: 0.6877 - val_acc: 0.8541
Epoch 16/20
7352/7352 [==============================] - 70s 10ms/step - loss: 0.1690 - a
cc: 0.9433 - val_loss: 0.4625 - val_acc: 0.8884
Epoch 17/20
7352/7352 [==============================] - 71s 10ms/step - loss: 0.1721 - a
cc: 0.9400 - val_loss: 0.6151 - val_acc: 0.8711
Epoch 18/20
7352/7352 [==============================] - 71s 10ms/step - loss: 0.1677 - a
cc: 0.9396 - val_loss: 0.3984 - val_acc: 0.8945
Epoch 19/20
7352/7352 [==============================] - 71s 10ms/step - loss: 0.1567 - a
```

```
cc: 0.9444 - val_loss: 0.7661 - val_acc: 0.8548
Epoch 20/20
7352/7352 [==============================] - 70s 10ms/step - loss: 0.1481 - a
cc: 0.9464 - val_loss: 0.4669 - val_acc: 0.8982
```

Out[28]:
```
<keras.callbacks.History at 0x1c75a136f60>
```

In [29]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Final evaluation of the model
scores = model3.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(model3.predict(X_t
est), axis=1)])

# Code for drawing seaborn heatmaps
class_names = ['laying','sitting','standing','walking','walking_downstairs','w
alking_upstairs']
df_heatmap = pd.DataFrame(confusion_matrix(Y_true, Y_predictions), index=class
_names, columns=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='r
ight', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=90, ha=
'right', fontsize=14)
plt.ylabel('True label',size=18)
plt.xlabel('Predicted label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```

```
Test Score: 0.466954
Test Accuracy: 89.820156%
```



RMSProp optimizer is suitable for this problem. accuracy again reach to 89%

loss also decrease from 0.9 to 0.46

let check with 2 LSTM network...

Model4: 2 LSTM with 32 hidden unit , adam optimizer

In [30]:
```
# Initiliazing the sequential model
model4 = Sequential()
# Configuring the parameters
model4.add(LSTM(32,return_sequences=True, input_shape=(timesteps, input_dim)))
model4.add(Dropout(0.5))

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_4 (LSTM)                (None, 128, 32)           5376      
_________________________________________________________________
dropout_4 (Dropout)          (None, 128, 32)           0         
_________________________________________________________________
lstm_5 (LSTM)                (None, 32)                8320      
_________________________________________________________________
dropout_5 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 198       
=================================================================
Total params: 13,894
Trainable params: 13,894
Non-trainable params: 0
_________________________________________________________________
None
```

In [31]:
```
# Compiling the model
model4.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Training the model
model4.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)
```

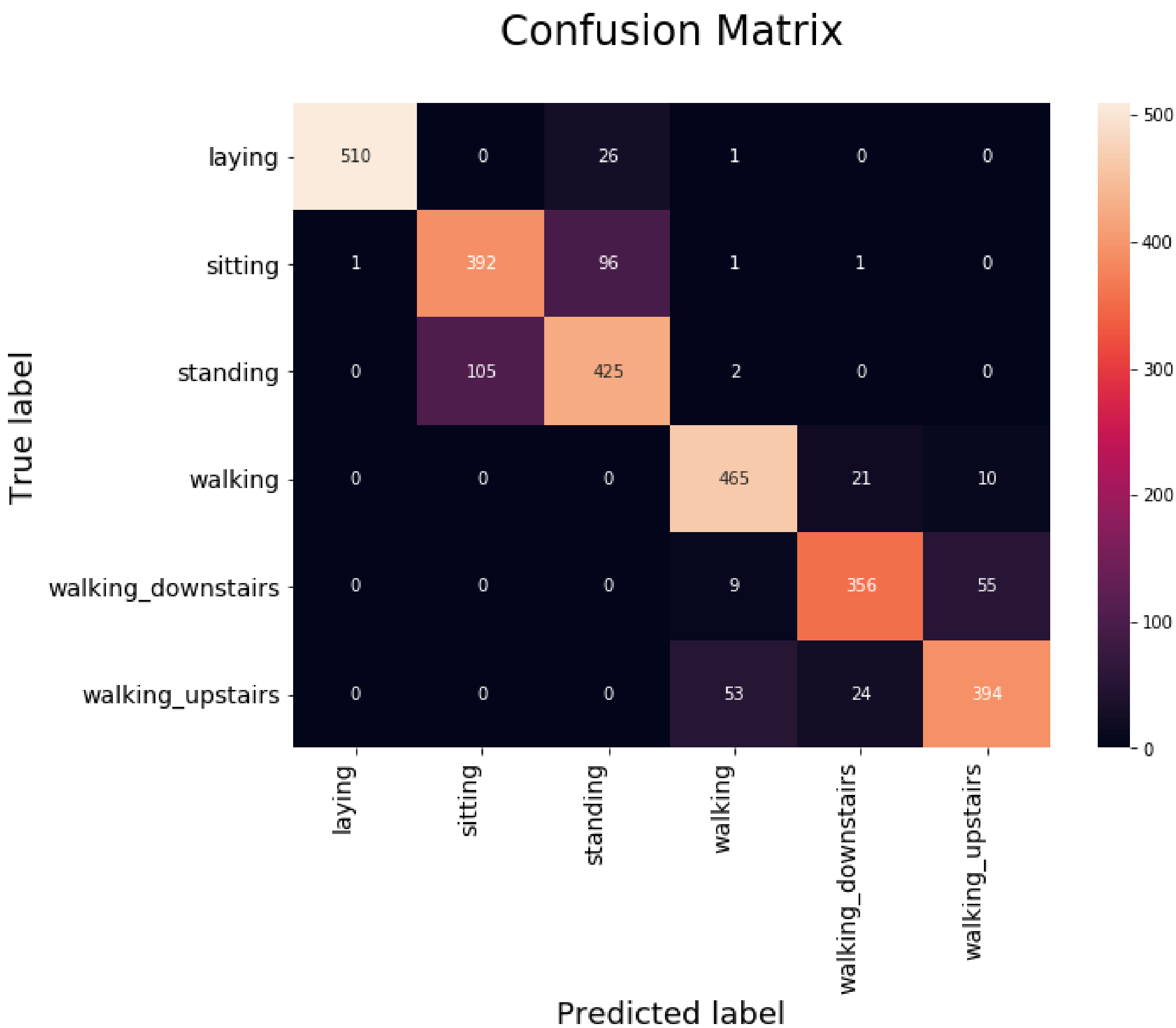
```
Train on 7352 samples, validate on 2947 samples
Epoch 1/20
7352/7352 [==============================] - 137s 19ms/step - loss: 1.2312 -
acc: 0.4869 - val_loss: 0.9302 - val_acc: 0.6210
Epoch 2/20
7352/7352 [==============================] - 130s 18ms/step - loss: 0.8336 -
acc: 0.6080 - val_loss: 0.7768 - val_acc: 0.6206
Epoch 3/20
7352/7352 [==============================] - 5732s 780ms/step - loss: 0.7795
- acc: 0.6174 - val_loss: 0.7749 - val_acc: 0.6430
Epoch 4/20
7352/7352 [==============================] - 129s 18ms/step - loss: 0.8037 -
acc: 0.6049 - val_loss: 0.7544 - val_acc: 0.5969
Epoch 5/20
7352/7352 [==============================] - 1588s 216ms/step - loss: 0.7207
- acc: 0.6138 - val_loss: 0.6889 - val_acc: 0.6125
Epoch 6/20
7352/7352 [==============================] - 120s 16ms/step - loss: 0.7499 -
acc: 0.5962 - val_loss: 0.7753 - val_acc: 0.5419
Epoch 7/20
7352/7352 [==============================] - 134s 18ms/step - loss: 0.7406 -
acc: 0.5967 - val_loss: 0.7400 - val_acc: 0.6111
Epoch 8/20
7352/7352 [==============================] - 144s 20ms/step - loss: 0.7154 -
acc: 0.6372 - val_loss: 0.7954 - val_acc: 0.5948
Epoch 9/20
7352/7352 [==============================] - 169s 23ms/step - loss: 0.6815 -
acc: 0.6446 - val_loss: 0.7485 - val_acc: 0.6162
Epoch 10/20
7352/7352 [==============================] - 159s 22ms/step - loss: 0.6546 -
acc: 0.6613 - val_loss: 0.9102 - val_acc: 0.5823
Epoch 11/20
7352/7352 [==============================] - 174s 24ms/step - loss: 0.7418 -
acc: 0.6239 - val_loss: 0.9104 - val_acc: 0.4601
Epoch 12/20
7352/7352 [==============================] - 197s 27ms/step - loss: 0.7262 -
acc: 0.6288 - val_loss: 0.7161 - val_acc: 0.6162
Epoch 13/20
7352/7352 [==============================] - 157s 21ms/step - loss: 0.6820 -
acc: 0.6522 - val_loss: 0.7361 - val_acc: 0.6152
Epoch 14/20
7352/7352 [==============================] - 158s 22ms/step - loss: 0.6417 -
acc: 0.6636 - val_loss: 0.7225 - val_acc: 0.6223
Epoch 15/20
7352/7352 [==============================] - 128s 17ms/step - loss: 0.6037 -
acc: 0.6904 - val_loss: 0.7110 - val_acc: 0.6301
Epoch 16/20
7352/7352 [==============================] - 134s 18ms/step - loss: 0.5680 -
acc: 0.7130 - val_loss: 0.6425 - val_acc: 0.6603
Epoch 17/20
7352/7352 [==============================] - 121s 17ms/step - loss: 0.5061 -
acc: 0.7669 - val_loss: 0.5578 - val_acc: 0.7482
Epoch 18/20
7352/7352 [==============================] - 148s 20ms/step - loss: 0.3826 -
acc: 0.8464 - val_loss: 0.6141 - val_acc: 0.8079
Epoch 19/20
7352/7352 [==============================] - 190s 26ms/step - loss: 0.3548 -
```

```
acc: 0.8902 - val_loss: 0.4817 - val_acc: 0.8697
Epoch 20/20
7352/7352 [==============================] - 126s 17ms/step - loss: 0.3011 -
acc: 0.9100 - val_loss: 0.4616 - val_acc: 0.8626
```

Out[31]:
```
<keras.callbacks.History at 0x1c75d23aef0>
```

In [32]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Final evaluation of the model
scores = model4.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(model4.predict(X_t
est), axis=1)])

# Code for drawing seaborn heatmaps
class_names = ['laying','sitting','standing','walking','walking_downstairs','w
alking_upstairs']
df_heatmap = pd.DataFrame(confusion_matrix(Y_true, Y_predictions), index=class
_names, columns=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='r
ight', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=90, ha=
'right', fontsize=14)
plt.ylabel('True label',size=18)
plt.xlabel('Predicted label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```

```
Test Score: 0.461573
Test Accuracy: 86.257211%
```



Model5: 2 LSTM with 64 hidden unit , adam optimizer

In [33]:
```
# Initiliazing the sequential model
model5 = Sequential()
# Configuring the parameters
model5.add(LSTM(64,return_sequences=True, input_shape=(timesteps, input_dim)))
model5.add(Dropout(0.5))

# Configuring the parameters
model5.add(LSTM(32))
model5.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model5.add(Dense(n_classes, activation='sigmoid'))
print(model4.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 128, 32)           5376
_________________________________________________________________
dropout_4 (Dropout)          (None, 128, 32)           0
_________________________________________________________________
lstm_5 (LSTM)                (None, 32)                8320
_________________________________________________________________
dropout_5 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 198
=================================================================
Total params: 13,894
Trainable params: 13,894
Non-trainable params: 0
_________________________________________________________________
None
```

In [34]:
```
# Compiling the model
model5.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Training the model
model5.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)
```

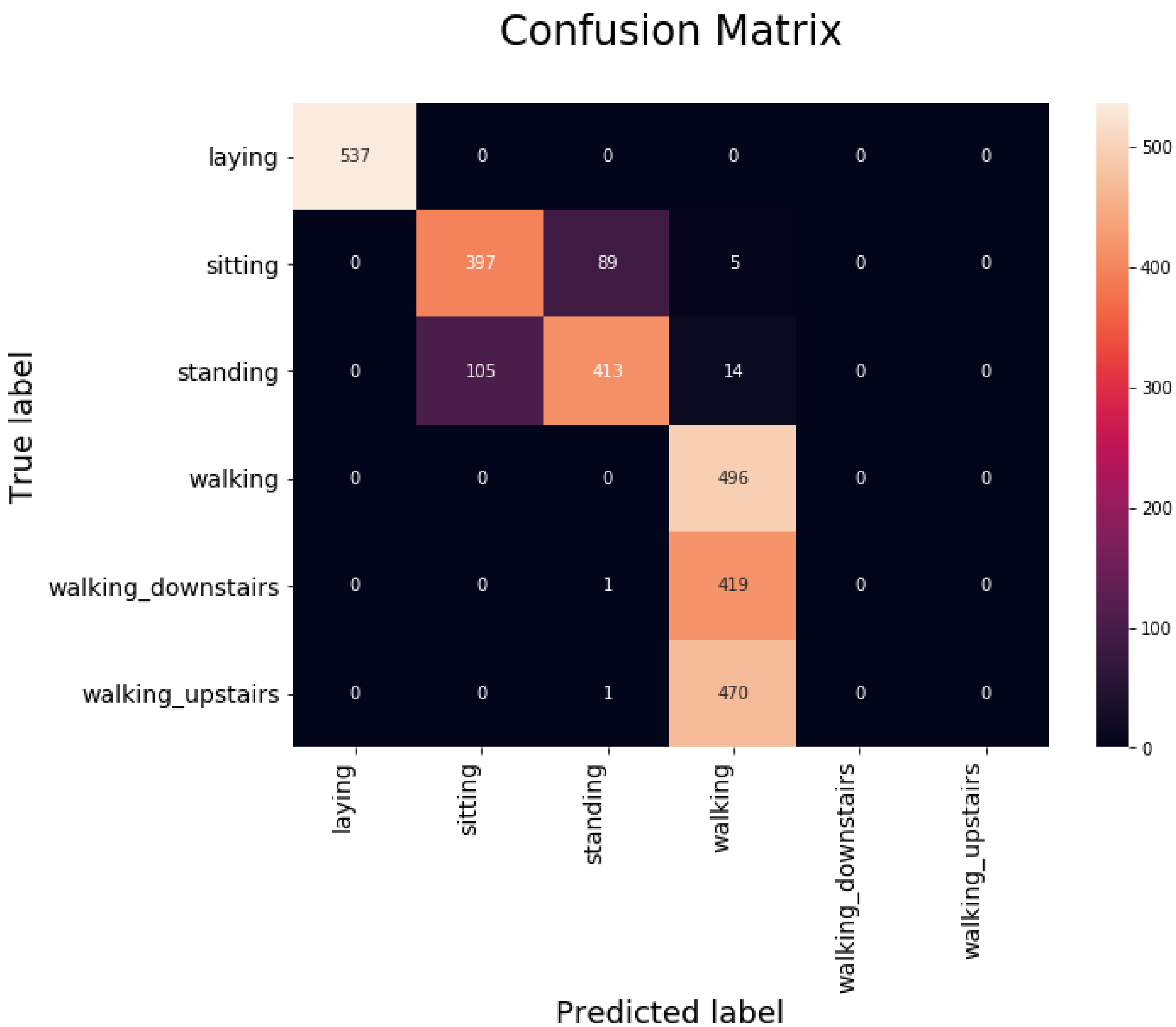
```
Train on 7352 samples, validate on 2947 samples
Epoch 1/20
7352/7352 [==============================] - 134s 18ms/step - loss: 1.4015 -
acc: 0.4169 - val_loss: 1.2836 - val_acc: 0.4564
Epoch 2/20
7352/7352 [==============================] - 128s 17ms/step - loss: 1.2390 -
acc: 0.4407 - val_loss: 1.2479 - val_acc: 0.4164
Epoch 3/20
7352/7352 [==============================] - 125s 17ms/step - loss: 1.2551 -
acc: 0.4348 - val_loss: 1.4182 - val_acc: 0.3488
Epoch 4/20
7352/7352 [==============================] - 120s 16ms/step - loss: 1.1706 -
acc: 0.4937 - val_loss: 1.0925 - val_acc: 0.5643
Epoch 5/20
7352/7352 [==============================] - 127s 17ms/step - loss: 0.9595 -
acc: 0.5817 - val_loss: 1.3757 - val_acc: 0.4147
Epoch 6/20
7352/7352 [==============================] - 154s 21ms/step - loss: 1.0993 -
acc: 0.5267 - val_loss: 0.9664 - val_acc: 0.5711
Epoch 7/20
7352/7352 [==============================] - 146s 20ms/step - loss: 0.9192 -
acc: 0.6043 - val_loss: 1.1931 - val_acc: 0.4676
Epoch 8/20
7352/7352 [==============================] - 125s 17ms/step - loss: 1.1554 -
acc: 0.5014 - val_loss: 1.2935 - val_acc: 0.4113
Epoch 9/20
7352/7352 [==============================] - 128s 17ms/step - loss: 0.9674 -
acc: 0.5543 - val_loss: 0.8394 - val_acc: 0.5911
Epoch 10/20
7352/7352 [==============================] - 132s 18ms/step - loss: 1.1149 -
acc: 0.5064 - val_loss: 1.0591 - val_acc: 0.4662
Epoch 11/20
7352/7352 [==============================] - 131s 18ms/step - loss: 0.9692 -
acc: 0.5234 - val_loss: 0.9272 - val_acc: 0.5253
Epoch 12/20
7352/7352 [==============================] - 120s 16ms/step - loss: 1.3288 -
acc: 0.4253 - val_loss: 1.2216 - val_acc: 0.4201
Epoch 13/20
7352/7352 [==============================] - 203s 28ms/step - loss: 1.1431 -
acc: 0.4916 - val_loss: 0.9540 - val_acc: 0.5684
Epoch 14/20
7352/7352 [==============================] - 172s 23ms/step - loss: 0.8517 -
acc: 0.5885 - val_loss: 0.8382 - val_acc: 0.6037
Epoch 15/20
7352/7352 [==============================] - 134s 18ms/step - loss: 0.9328 -
acc: 0.5692 - val_loss: 0.8958 - val_acc: 0.5755
Epoch 16/20
7352/7352 [==============================] - 119s 16ms/step - loss: 0.8119 -
acc: 0.6144 - val_loss: 0.7843 - val_acc: 0.6084
Epoch 17/20
7352/7352 [==============================] - 116s 16ms/step - loss: 0.7306 -
acc: 0.6371 - val_loss: 0.7516 - val_acc: 0.6138
Epoch 18/20
7352/7352 [==============================] - 125s 17ms/step - loss: 0.6908 -
acc: 0.6525 - val_loss: 0.7487 - val_acc: 0.6216
Epoch 19/20
7352/7352 [==============================] - 122s 17ms/step - loss: 0.6674 -
```

```
acc: 0.6508 - val_loss: 0.7072 - val_acc: 0.6315
Epoch 20/20
7352/7352 [==============================] - 144s 20ms/step - loss: 0.6570 -
acc: 0.6555 - val_loss: 0.7070 - val_acc: 0.6254
```

Out[34]:
```
<keras.callbacks.History at 0x1c762bd1ef0>
```

In [35]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Final evaluation of the model
scores = model5.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(model5.predict(X_t
est), axis=1)])

# Code for drawing seaborn heatmaps
class_names = ['laying','sitting','standing','walking','walking_downstairs','w
alking_upstairs']
df_heatmap = pd.DataFrame(confusion_matrix(Y_true, Y_predictions), index=class
_names, columns=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='r
ight', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=90, ha=
'right', fontsize=14)
plt.ylabel('True label',size=18)
plt.xlabel('Predicted label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```

```
Test Score: 0.707013
Test Accuracy: 62.538174%
```



Model6: 2 LSTM with 64 hidden unit , rmsprop optimizer

In [36]:
```
# Initiliazing the sequential model
model6 = Sequential()
# Configuring the parameters
model6.add(LSTM(64,return_sequences=True, input_shape=(timesteps, input_dim)))
model6.add(Dropout(0.5))

# Configuring the parameters
model6.add(LSTM(32))
model6.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model6.add(Dense(n_classes, activation='sigmoid'))
print(model4.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_4 (LSTM)                (None, 128, 32)           5376      
_________________________________________________________________
dropout_4 (Dropout)          (None, 128, 32)           0         
_________________________________________________________________
lstm_5 (LSTM)                (None, 32)                8320      
_________________________________________________________________
dropout_5 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 198       
=================================================================
Total params: 13,894
Trainable params: 13,894
Non-trainable params: 0
_________________________________________________________________
None
```

In [37]:
```
# Compiling the model
model6.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

# Training the model
model6.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 1/20
7352/7352 [==============================] - 134s 18ms/step - loss: 1.2233 -
acc: 0.4961 - val_loss: 0.9743 - val_acc: 0.5986
Epoch 2/20
7352/7352 [==============================] - 130s 18ms/step - loss: 0.8135 -
acc: 0.6517 - val_loss: 0.7748 - val_acc: 0.7197
Epoch 3/20
7352/7352 [==============================] - 128s 17ms/step - loss: 0.6675 -
acc: 0.7274 - val_loss: 0.6315 - val_acc: 0.7913
Epoch 4/20
7352/7352 [==============================] - 135s 18ms/step - loss: 0.4925 -
acc: 0.8443 - val_loss: 0.5876 - val_acc: 0.8005
Epoch 5/20
7352/7352 [==============================] - 121s 16ms/step - loss: 0.3329 -
acc: 0.9048 - val_loss: 0.4312 - val_acc: 0.8663
Epoch 6/20
7352/7352 [==============================] - 125s 17ms/step - loss: 0.2665 -
acc: 0.9253 - val_loss: 0.3789 - val_acc: 0.8873
Epoch 7/20
7352/7352 [==============================] - 122s 17ms/step - loss: 0.2311 -
acc: 0.9271 - val_loss: 0.4710 - val_acc: 0.8690
Epoch 8/20
7352/7352 [==============================] - 214s 29ms/step - loss: 0.1946 -
acc: 0.9334 - val_loss: 0.3532 - val_acc: 0.8972
Epoch 9/20
7352/7352 [==============================] - 173s 24ms/step - loss: 0.2094 -
acc: 0.9380 - val_loss: 0.3589 - val_acc: 0.9033
Epoch 10/20
7352/7352 [==============================] - 124s 17ms/step - loss: 0.1785 -
acc: 0.9415 - val_loss: 0.2869 - val_acc: 0.9070
Epoch 11/20
7352/7352 [==============================] - 144s 20ms/step - loss: 0.1875 -
acc: 0.9418 - val_loss: 0.3843 - val_acc: 0.9016
Epoch 12/20
7352/7352 [==============================] - 156s 21ms/step - loss: 0.1964 -
acc: 0.9399 - val_loss: 0.3523 - val_acc: 0.8860
Epoch 13/20
7352/7352 [==============================] - 231s 31ms/step - loss: 0.1772 -
acc: 0.9382 - val_loss: 0.4540 - val_acc: 0.8839
Epoch 14/20
7352/7352 [==============================] - 156s 21ms/step - loss: 0.1703 -
acc: 0.9455 - val_loss: 0.3913 - val_acc: 0.8928
Epoch 15/20
7352/7352 [==============================] - 143s 19ms/step - loss: 0.1718 -
acc: 0.9446 - val_loss: 0.3111 - val_acc: 0.9152
Epoch 16/20
7352/7352 [==============================] - 142s 19ms/step - loss: 0.1520 -
acc: 0.9490 - val_loss: 0.3391 - val_acc: 0.9128
Epoch 17/20
7352/7352 [==============================] - 144s 20ms/step - loss: 0.1565 -
acc: 0.9472 - val_loss: 0.4047 - val_acc: 0.9060
Epoch 18/20
7352/7352 [==============================] - 132s 18ms/step - loss: 0.1499 -
acc: 0.9470 - val_loss: 0.3015 - val_acc: 0.8992
Epoch 19/20
7352/7352 [==============================] - 124s 17ms/step - loss: 0.1385 -
```

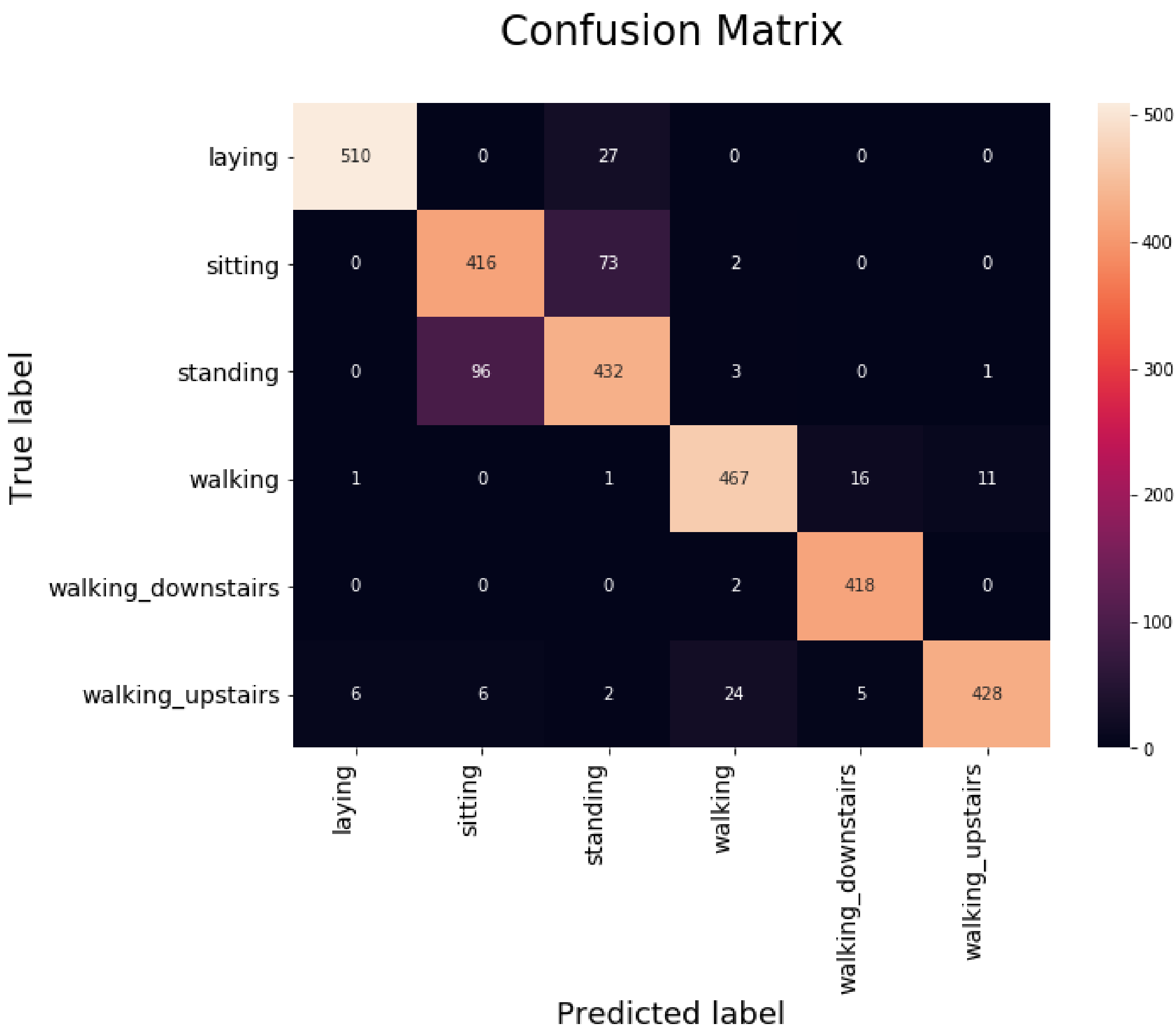
```
acc: 0.9498 - val_loss: 0.3873 - val_acc: 0.9057
Epoch 20/20
7352/7352 [==============================] - 145s 20ms/step - loss: 0.1550 -
acc: 0.9510 - val_loss: 0.4510 - val_acc: 0.9063
```

Out[37]:

```
<keras.callbacks.History at 0x1c7526f1550>
```

In [38]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Final evaluation of the model
scores = model6.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(model6.predict(X_t
est), axis=1)])

# Code for drawing seaborn heatmaps
class_names = ['laying','sitting','standing','walking','walking_downstairs','w
alking_upstairs']
df_heatmap = pd.DataFrame(confusion_matrix(Y_true, Y_predictions), index=class
_names, columns=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='r
ight', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=90, ha=
'right', fontsize=14)
plt.ylabel('True label',size=18)
plt.xlabel('Predicted label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```

```
Test Score: 0.451065
Test Accuracy: 90.634544%
```



Observation:

```
test accuracy reach upto 90.0 % with 2 LSTM Layer and rmsprop optimizer.

loss is 0.45.
```

Overall Observation:

For this problem, require optimal parameters are........

2 LSTM model,

rmsprop optimizer,

64 hidden unit

In [ ]:

```

```