Configuration Manual: Human Activity Recognition using Deep Learning Approach

MSc Research Project
Data Analytics

Laiba Rehman
Student ID: x20144032

School of Computing
National College of Ireland

Supervisor: Mohammed Hasanuzzaman

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Laiba Rehman……………………………………………………………………

Student ID: X20144032…………………………………………………………………………..……

Programme: Data Analytics…………… **Year:** 2021-2022..

Module: M.Sc. Research Project……………………………………………………….………

Lecturer: …………………………………………………………………………………………….………

Submission Due Date: 31/01/2022……………………………………………………………………….………

Project Title: Human Activity Recognition using Deep Learning Approach….………

Word Count: 1057… **Page Count:** 19…………………….…….………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Laiba………………………………………………………………………………

Date: 31/01/2022…………………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual: Human Activity Recognition using a Deep Learning Approach

Laiba Rehman
x20144032

1 Introduction

The configuration manual provides an overview of the hardware, software, and programming required to complete the MSc Research Project "Human Activity Recognition Using a Deep Learning Approach." Additionally, it discusses the specifics of the needed libraries. The last portion of this document contains code and major output for all execution, results, and assessment procedures.

2 Hardware Requirement

For environmental setup, an HP laptop with a 64-bit OS system is utilized.

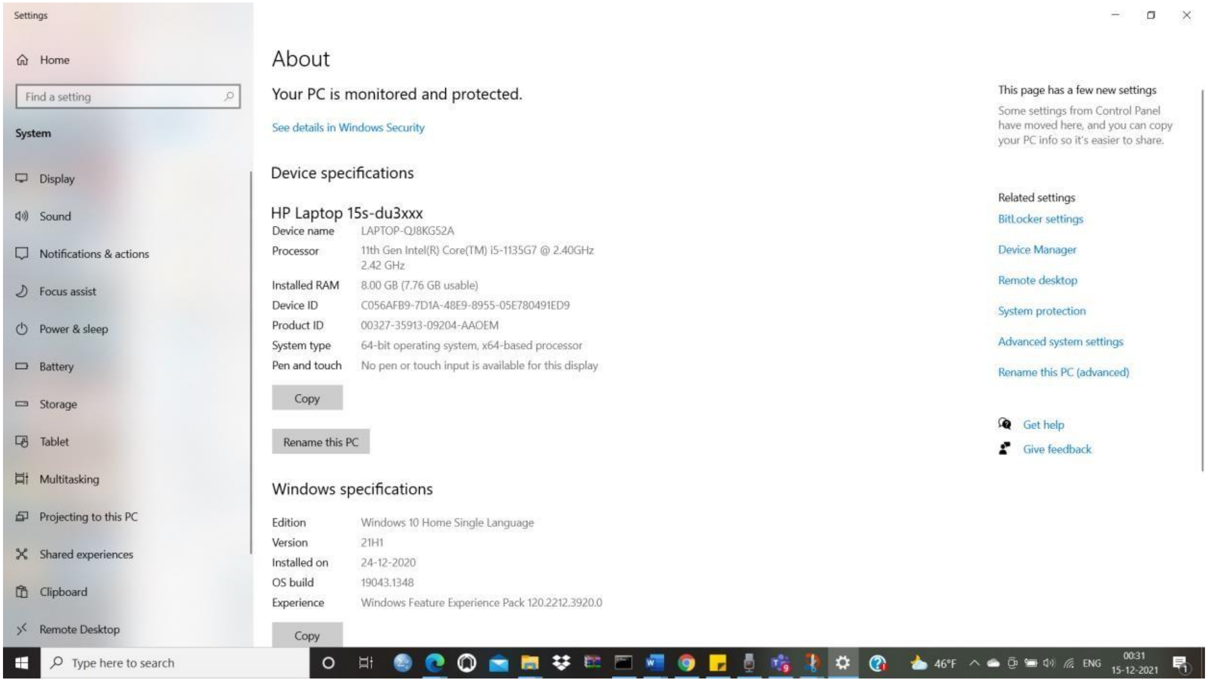


Fig 1 Device and Windows Specifications

The above-mentioned configuration is the one on which the scripts were executed, but the requirements were more than these. There were certain limitations that were observed during executing the models in the project. The limitations include high amount of time taken during training each model and different errors that are encountered during executing the hyper parameter tuning of the projects with the help of hyperas libraries.

This indicated us that the code should be updated in different period of time in order to support updated libraries and methods that reduce the issues that are encountered now. The training time can be reduced by executing all the models in a better hardware system that has at least 16 GB of RAM with high end processor and graphics.

3 Software Requirements

Jupyter Notebook was used to write and run these scripts. Jupyter Notebook is an Integrated Development Environment (IDE) for writing Python scripts. Because the data was captured in a.csv file, we kept it on the system, as the jupyter notebook can access files and run applications directly on the system. To execute jupyter notebook, we must first open a command prompt in the same directory and then pre-install all the Python libraries and additional deep learning frameworks such as TensorFlow and Keras, as well as sklearn.

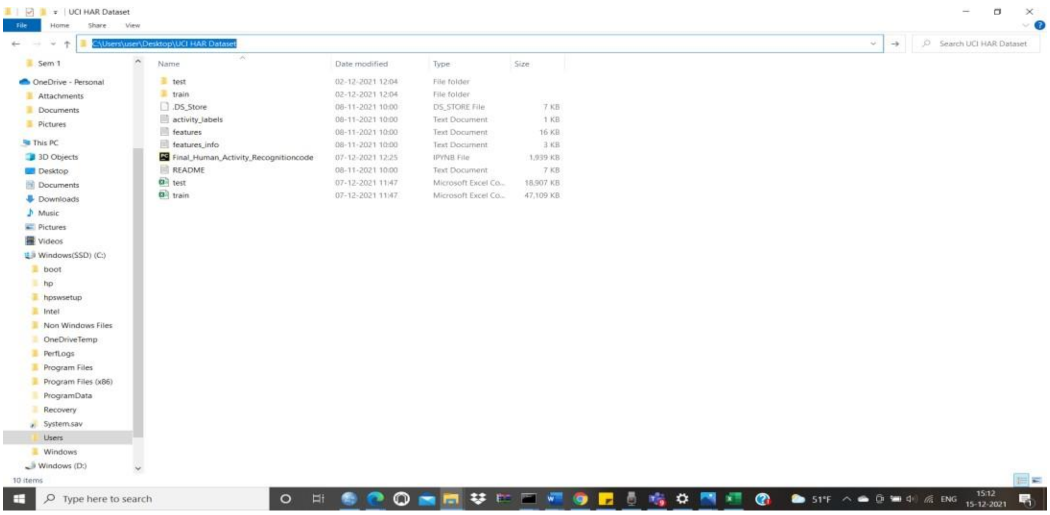


Fig: Directory path

The Figure below shows all the libraries imported in our code.

```
In [1]: import numpy as np
        import pandas as pd
        import warnings
        warnings.filterwarnings("ignore")

        import matplotlib.pyplot as plt
        import seaborn as sns

        sns.set_style('whitegrid')


        from sklearn.manifold import TSNE
        import matplotlib.pyplot as plt

        import itertools
        from sklearn import metrics
        from sklearn.metrics import confusion_matrix

        from numpy import mean
        from numpy import std
        from numpy import dstack
        from pandas import read_csv
        from matplotlib import pyplot
        from sklearn.preprocessing import StandardScaler
        from keras.models import Sequential
        from keras.layers import Dense
        from keras.layers import Flatten
        from keras.layers import Dropout
        from keras.layers.convolutional import Conv1D
        from keras.layers.convolutional import MaxPooling1D
        from keras.utils import to_categorical
        from keras.models import Sequential
        from keras.layers import LSTM
        from keras.layers.core import Dense, Dropout
        from keras.models import load_model
        # Importing tensorflow
        np.random.seed(42)
        import tensorflow as tf
        tf.set_random_seed(42)
```

```python
from sklearn import metrics
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score,accuracy_score,classification_report

from hyperopt import Trials, STATUS_OK, tpe
from hyperas import optim
from hyperas.distributions import choice, uniform
from hyperas.utils import eval_hyperopt_space

from keras.regularizers import l2, l1
import keras
from keras.layers import BatchNormalization
import os
os.environ['PYTHONHASHSEED'] = '0'

from keras import backend as K

import math
from keras.callbacks import LearningRateScheduler
from sklearn.base import BaseEstimator, TransformerMixin
import pickle
```

Obtain the train data

```python
In [3]: # get the data from txt files to pandas dataffame
X_train = pd.read_csv('C:/Users/user/Desktop/UCI HAR Dataset/train/X_train.txt', delim_whitespace=True, header=None)
```

Fig 2: Obtain the train data from the text files to pandas dataframe

Obtain the test data

```python
In [6]: # get the data from txt files to pandas dataffame
X_test = pd.read_csv('C:/Users/user/Desktop/UCI HAR Dataset/test/X_test.txt', delim_whitespace=True, header=None)
```

Fig 3: Obtain the test data from the text files to pandas dataframe

Data Preprocessing

1. Check for Duplicates

```python
In [12]: print('No of duplicates in train: {}'.format(sum(train.duplicated())))
print('No of duplicates in test : {}'.format(sum(test.duplicated())))
```

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

Fig 4

2. Checking for NaN/null values

```python
In [13]: print('We have {} NaN/Null values in train'.format(train.isnull().values.sum()))
print('We have {} NaN/Null values in test'.format(test.isnull().values.sum()))
```

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

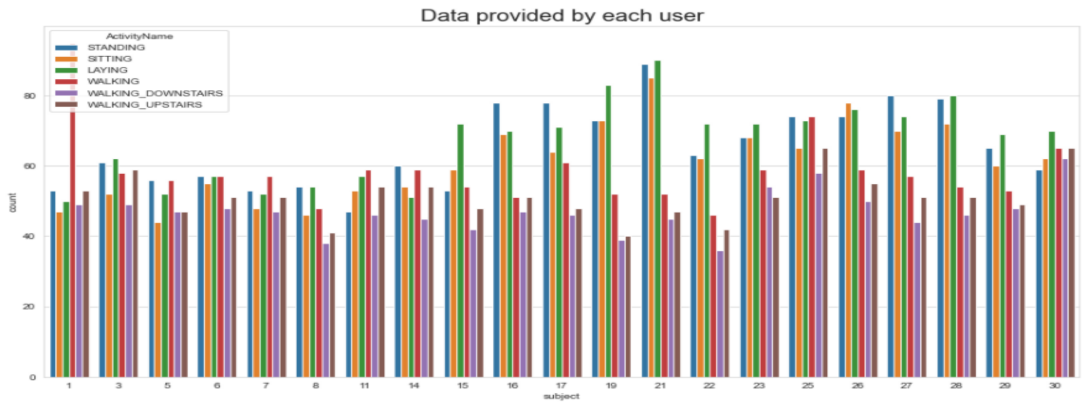
Fig 5

3. Check for data imbalance

In [14]:
```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style('whitegrid')
```

In [15]:
```
plt.figure(figsize=(16,8))
plt.title('Data provided by each user', fontsize=20)
sns.countplot(x='subject',hue='ActivityName', data = train)
plt.show()
```



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

Fig 6

Observation

The data was nearly balanced.

4. Changing feature names

In [17]:
```
columns = train.columns

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

train.columns = columns
test.columns = columns

test.columns
```

Out[17]:
```
Index(['tBodyAcc_mean_X', 'tBodyAcc_mean_Y', 'tBodyAcc_mean_Z',
       'tBodyAcc_std_X', 'tBodyAcc_std_Y', 'tBodyAcc_std_Z', 'tBodyAcc_mad_X',
       'tBodyAcc_mad_Y', 'tBodyAcc_mad_Z', 'tBodyAcc_max_X',
       ...
       'angletBodyAccMeangravity', 'angletBodyAccJerkMeangravityMean',
       'angletBodyGyroMeangravityMean', 'angletBodyGyroJerkMeangravityMean',
       'angleXgravityMean', 'angleYgravityMean', 'angleZgravityMean',
       'subject', 'Activity', 'ActivityName'],
      dtype='object', length=564)
```

Fig 7

5. Save this dataframe in a csv files

In [18]:
```
train.to_csv('C:/Users/user/Desktop/UCI HAR Dataset/train.csv', index=False)
test.to_csv('C:/Users/user/Desktop/UCI HAR Dataset/test.csv', index=False)
```

Fig 8

Apply t-sne on the data

```
In [24]: import numpy as np
         from sklearn.manifold import TSNE
         import matplotlib.pyplot as plt
         import seaborn as sns
```

```
In [25]: # performs t-sne with different perplexity values and their repective plots..

         def perform_tsne(X_data, y_data, perplexities, n_iter=1000, img_name_prefix='t-sne'):

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

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

                 # draw the plot in appropriate place in the grid
                 sns.lmplot(data=df, x='x', y='y', hue='label', fit_reg=False, size=8,\
                            palette="Set1",markers=['^','v','s','o', '1','2'])
                 plt.title("perplexity : {} and max_iter : {}".format(perplexity, n_iter))
                 img_name = img_name_prefix + '_perp_{}_iter_{}.png'.format(perplexity, n_iter)
                 print('saving this plot as image in present working directory...')
                 plt.savefig(img_name)
                 plt.show()
                 print('Done')
```

```
In [26]: X_pre_tsne = train.drop(['subject', 'Activity','ActivityName'], axis=1)
         y_pre_tsne = train['ActivityName']
         perform_tsne(X_data = X_pre_tsne,y_data=y_pre_tsne, perplexities =[2,5,10,20,50])
```

Fig 9

We have visualized all the activities with the help of the t-Distributed Stochastic Neighbour Embedding that will convert all the data from high dimensional to 2-dimensional space. We will use different perplexity values in order to separate the data in order to understand how distinct the classes are from each other. The results are shown in the report.

Building Models

a) LSTM

In LSTM model, we will build a single layer and multilayer LSTM model and also a model with regularization technique in order to see if there is an improvement in accuracy compared to the baseline model.

LSTM Base Model

```
In [49]: # 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:tensorflow:From C:\Users\user\Anaconda3\envs\venv\lib\site-packages\ten
sorflow\python\ops\resource_variable_ops.py:435: colocate_with (from tensorflo
w.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 32)                5376
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 198
=================================================================
Total params: 5,574
Trainable params: 5,574
Non-trainable params: 0
_________________________________________________________________
```

Fig 10

In [50]:
```python
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

In [51]:
```python
# Training the model
model.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)
```

```
WARNING:tensorflow:From C:\Users\user\Anaconda3\envs\venv\lib\site-packages\ten
sorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_
ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 31s 4ms/step - loss: 1.2941 - accu
racy: 0.4523 - val_loss: 1.1074 - val_accuracy: 0.4924
Epoch 2/30
```

Fig 11 Compiling and Training the model

Multi layer LSTM

In [94]:
```python
# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(32,return_sequences=True,input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model.add(Dropout(0.5))

model.add(LSTM(28,input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model.add(Dropout(0.6))
# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 128, 32)           5376
_________________________________________________________________
dropout_2 (Dropout)          (None, 128, 32)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 28)                6832
_________________________________________________________________
dropout_3 (Dropout)          (None, 28)                0
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 174
=================================================================
Total params: 12,382
Trainable params: 12,382
Non-trainable params: 0
_________________________________________________________________
```

Fig 12

```
In [95]: # Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

In [96]: # 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 [==============================] - 67s 9ms/step - loss: 1.2287 - accu
racy: 0.5173 - val_loss: 0.8923 - val_accuracy: 0.6193
Epoch 2/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.8551 - accu
racy: 0.6317 - val_loss: 0.8429 - val_accuracy: 0.6383
Epoch 3/30
7352/7352 [==============================] - 66s 9ms/step - loss: 0.7493 - accu
racy: 0.6536 - val_loss: 0.7709 - val_accuracy: 0.6325
Epoch 4/30
7352/7352 [==============================] - 62s 8ms/step - loss: 0.7210 - accu
racy: 0.6835 - val_loss: 0.7076 - val_accuracy: 0.6960
Epoch 5/30
7352/7352 [==============================] - 59s 8ms/step - loss: 0.6750 - accu
```

Fig 13: Compiling and Training the model

By making a comparison of above shown, 2 layer LSTM model is giving similar score as 1 layer LSTM model which we trained.

```
In [100]: from keras.regularizers import l2

In [101]: # Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(32,recurrent_regularizer=l2(0.003),return_sequences=True,input_sh
# Adding a dropout layer
model.add(Dropout(0.5))

model.add(LSTM(28,input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model.add(Dropout(0.6))
# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))
model.summary()

Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 128, 32)           5376
_________________________________________________________________
dropout_4 (Dropout)          (None, 128, 32)           0
_________________________________________________________________
lstm_5 (LSTM)                (None, 28)                6832
_________________________________________________________________
dropout_5 (Dropout)          (None, 28)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 174
=================================================================
Total params: 12,382
Trainable params: 12,382
Non-trainable params: 0
```

Fig 14: LSTM model with regularization technique

Hyperparameter Tuning Using Hyperas:

```
In [105]: # Importing tensorflow
          np.random.seed(36)
          import tensorflow as tf
          tf.set_random_seed(36)
```

```
In [106]: # Importing libraries
          from keras.models import Sequential
          from keras.layers import LSTM
          from keras.layers.core import Dense, Dropout
          from hyperopt import Trials, STATUS_OK, tpe
          from hyperas import optim
          from hyperas.distributions import choice, uniform
          from hyperas.utils import eval_hyperopt_space
```

Fig 15: Importing libraries of hyperas

```
In [43]: X_train, Y_train, X_val, Y_val = data()
         trials = Trials()
         best_run, best_model, space = optim.minimize(model=model,
                                               data=data,
                                               algo=tpe.suggest,
                                               max_evals=15,
                                               trials=trials,notebook_name = 'Human Activ
                                              return_space = True)
```

```
>>> Imports:
#coding=utf-8

try:
    from keras.models import Sequential
except:
    pass

try:
    from keras.layers import LSTM
except:
    pass

try:
    from keras.layers.core import Dense, Dropout
except:
    pass

try:
```

Fig 16

b) Convolution Neural Network

In CNN model, we will build a baseline model and a model using L2 regularized parameter and compare the accuracy of the model.

Using CNN

```
In [109]: import os
          os.environ['PYTHONHASHSEED'] = '0'
          import numpy as np
          import tensorflow as tf
          import random as rn
          np.random.seed(36)
          rn.seed(36)
          tf.set_random_seed(36)
          # Force TensorFlow to use single thread.
          # Multiple threads are a potential source of non-reproducible results.
          # For further details, see: https://stackoverflow.com/questions/42022950/
          session_conf = tf.ConfigProto(intra_op_parallelism_threads=1,
                                        inter_op_parallelism_threads=1)

          from keras import backend as K

          # The below tf.set_random_seed() will make random number generation
          # in the TensorFlow backend have a well-defined initial state.
          # For further details, see:
          # https://www.tensorflow.org/api_docs/python/tf/set_random_seed

          tf.set_random_seed(36)

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

Fig 17

Base Model

In [115]:
```
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer
model.add(Dropout(0.6))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(6, activation='softmax'))
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 126, 32)           896
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 124, 32)           3104
_________________________________________________________________
dropout_6 (Dropout)          (None, 124, 32)           0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 62, 32)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1984)              0
_________________________________________________________________
dense_4 (Dense)              (None, 50)                99250
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 306
=================================================================
Total params: 103,556
Trainable params: 103,556
Non-trainable params: 0
_________________________________________________________________
```

Fig 18 CNN Base Model

In [116]:
```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [117]:
```
model.fit(X_train_sc,Y_train, epochs=30, batch_size=16,validation_data=(X_val_sc, Y_val), verbose=1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 8s 1ms/step - loss: 0.4746 - accuracy: 0.8270 - val_loss: 0.3
0.8836
Epoch 2/30
7352/7352 [==============================] - 7s 1ms/step - loss: 0.1416 - accuracy: 0.9410 - val_loss: 0.2
0.9091
Epoch 3/30
7352/7352 [==============================] - 7s 989us/step - loss: 0.1241 - accuracy: 0.9487 - val_loss: 0
y: 0.9169
Epoch 4/30
7352/7352 [==============================] - 7s 1ms/step - loss: 0.1037 - accuracy: 0.9553 - val_loss: 0.2
0.9186
Epoch 5/30
7352/7352 [==============================] - 7s 985us/step - loss: 0.0941 - accuracy: 0.9558 - val_loss: 0
y: 0.9260
Epoch 6/30
7352/7352 [==============================] - 7s 989us/step - loss: 0.0831 - accuracy: 0.9600 - val_loss: 0
y: 0.9260
```

it is giving some good score in train as well as test but it is overfitting so much. i will try some regularization in below models.

In [122]:
```
test_cnn = np.argmax(Y_val, axis=1)
```

In [123]:
```
pred_cnn=model.predict_classes(X_val_sc)
print(classification_report(test_cnn, pred_cnn))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       496
           1       0.96      0.91      0.94       471
           2       0.92      1.00      0.96       420
           3       0.82      0.81      0.82       491
           4       0.84      0.82      0.83       532
           5       1.00      1.00      1.00       537

    accuracy                           0.92      2947
   macro avg       0.92      0.92      0.92      2947
weighted avg       0.92      0.92      0.92      2947
```

Fig 19 Compiling and Training the model

```
In [126]: from keras.regularizers import l2,l1
          import keras
          from keras.layers import BatchNormalization
```

```
In [127]: model = Sequential()
          model.add(Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer
                           kernel_regularizer=l2(0.1),input_shape=(128,9)))
          model.add(Conv1D(filters=16, kernel_size=3, activation='relu',kernel_regularizer
          model.add(Dropout(0.65))
          model.add(MaxPooling1D(pool_size=2))
          model.add(Flatten())
          model.add(Dense(32, activation='relu'))
          model.add(Dense(6, activation='softmax'))
          model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_3 (Conv1D)            (None, 126, 32)           896
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 124, 16)           1552
_________________________________________________________________
dropout_7 (Dropout)          (None, 124, 16)           0
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 62, 16)            0
_________________________________________________________________
flatten_2 (Flatten)          (None, 992)               0
_________________________________________________________________
dense_6 (Dense)              (None, 32)                31776
_________________________________________________________________
dense_7 (Dense)              (None, 6)                 198
=================================================================
Total params: 34,422
Trainable params: 34,422
Non-trainable params: 0
_________________________________________________________________
```

Fig 20 CNN model with regularization technique

```
In [128]: import math
          adam = keras.optimizers.Adam(lr=0.001)
          rmsprop = keras.optimizers.RMSprop(lr=0.001)
          def step_decay(epoch):
              return float(0.001 * math.pow(0.6, math.floor((1+epoch)/10)))
          from keras.callbacks import LearningRateScheduler
          lrate = LearningRateScheduler(step_decay)
          callbacks_list = [lrate]

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

```
In [129]: model.fit(X_train_sc,Y_train, epochs=30, batch_size=16,validation_data=(X_val_sc, Y_val), verbose=1)

          Train on 7352 samples, validate on 2947 samples
          Epoch 1/30
          7352/7352 [==============================] - 7s 935us/step - loss: 3.9366 - accuracy: 0.8009 - val_lo
          y: 0.8795
          Epoch 2/30
          7352/7352 [==============================] - 6s 787us/step - loss: 0.6804 - accuracy: 0.9202 - val_lo
          y: 0.8663
          Epoch 3/30
          7352/7352 [==============================] - 6s 784us/step - loss: 0.3624 - accuracy: 0.9282 - val_lo
          y: 0.8436
          Epoch 4/30
          7352/7352 [==============================] - 6s 808us/step - loss: 0.2917 - accuracy: 0.9342 - val_lo
          y: 0.8931
          Epoch 5/30
          7352/7352 [==============================] - 6s 773us/step - loss: 0.2628 - accuracy: 0.9380 - val_lo
          y: 0.8663
          Epoch 6/30
          7352/7352 [==============================] - 6s 814us/step - loss: 0.2459 - accuracy: 0.9357 - val_lo
          y: 0.8873
```

```
In [130]: test_cnn2 = np.argmax(Y_val, axis=1)

          pred_cnn2=model.predict_classes(X_val_sc)
          print(classification_report(test_cnn2, pred_cnn2))

                        precision    recall  f1-score   support

                     0       0.94      0.98      0.96       496
```

Fig 21 Compiling and Training the model

```
X_train, Y_train, X_val, Y_val = data_scaled()
trials = Trials()
best_run, best_model, space = optim.minimize(model=model_cnn,
                                      data=data_scaled,
                                      algo=tpe.suggest,
                                      max_evals=100,
                                      trials=trials,notebook_name = 'HAR',
                                      return_space = True)
```

```
from hyperas.utils import eval_hyperopt_space
total_trials = dict()
total_list = []
for t, trial in enumerate(trials):
        vals = trial.get('misc').get('vals')
        z = eval_hyperopt_space(space, vals)
        total_trials['M'+str(t+1)] = z
```

Fig 22 **Hyperparameter tuning using Hyperas on CNN model**

c) Divide-And-Conquer based CNN model

In divide and conquer based approach we will build two models using three classes which will divide the model into Static and dynamic activities and build CNN model on each of them. We will also perform both base line and model with regularized parameter to see if there is an improving accuracy in both the models.

Divide and Conquer-Based:

```
In [141]: import os
os.environ['PYTHONHASHSEED'] = '0'
import numpy as np
import tensorflow as tf
import random as rn
np.random.seed(0)
rn.seed(0)
tf.set_random_seed(0)
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1,
                              inter_op_parallelism_threads=1)

from keras import backend as K

# The below tf.set_random_seed() will make random number generation
# in the TensorFlow backend have a well-defined initial state.
# For further details, see:
# https://www.tensorflow.org/api_docs/python/tf/set_random_seed

tf.set_random_seed(0)

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

Fig 23

Model for classifying data into Static and Dynamic activities

In [148]:
```
K.clear_session()
np.random.seed(0)
tf.set_random_seed(0)
sess = tf.Session(graph=tf.get_default_graph())
K.set_session(sess)
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer
model.add(Dropout(0.6))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(2, activation='softmax'))
model.summary()
```

```
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, 50)                99250
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 102
=================================================================
Total params: 103,352
Trainable params: 103,352
Non-trainable params: 0
```

Fig 24

Classificaton of Static activities

In [157]:
```
##data preparation
def data_scaled_static():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    # Data directory
    DATADIR = 'C:/Users/user/Desktop/UCI HAR Dataset'
    # 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"
        ]
    from sklearn.base import BaseEstimator, TransformerMixin
    class scaling_tseries_data(BaseEstimator, TransformerMixin):
        from sklearn.preprocessing import StandardScaler
        def __init__(self):
            self.scale = None

        def transform(self, X):
            temp_X1 = X.reshape((X.shape[0] * X.shape[1], X.shape[2]))
            temp_X1 = self.scale.transform(temp_X1)
            return temp_X1.reshape(X.shape)
```

Fig 25

Baseline Model

```
In [161]: np.random.seed(0)
tf.set_random_seed(0)
sess = tf.Session(graph=tf.get_default_graph())
K.set_session(sess)
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=7, activation='relu',kernel_initializer
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer
model.add(Dropout(0.6))
model.add(MaxPooling1D(pool_size=3))
model.add(Flatten())
model.add(Dense(30, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_3 (Conv1D)            (None, 122, 64)           4096
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 120, 32)           6176
_________________________________________________________________
dropout_2 (Dropout)          (None, 120, 32)           0
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 40, 32)            0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1280)              0
_________________________________________________________________
dense_3 (Dense)              (None, 30)                38430
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 93
=================================================================
Total params: 48,795
Trainable params: 48,795
Non-trainable params: 0
_________________________________________________________________
```

Fig 26: Applying CNN model on Static Activities

```
In [21]: from hyperas.utils import eval_hyperopt_space
total_trials = dict()
total_list = []
for t, trial in enumerate(trials):
        vals = trial.get('misc').get('vals')
        z = eval_hyperopt_space(space, vals)
        total_trials['M'+str(t+1)] = z


#best Hyper params from hyperas
best_params = eval_hyperopt_space(space, best_run)
best_params

Out[21]: {'Dense': 64,
 'Dense_1': 64,
 'Dropout': 0.45377377480700615,
 'choiceval': 'rmsprop',
```

Fig 27: Hyperparameter Tuning using Hyperas

Classification of Dynamic activities :

In [165]:
```
##data preparation
def data_scaled_dynamic():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    # Data directory
    DATADIR = 'C:/Users/user/Desktop/UCI HAR Dataset'
    # 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"
        ]
    from sklearn.base import BaseEstimator, TransformerMixin
    class scaling_tseries_data(BaseEstimator, TransformerMixin):
        from sklearn.preprocessing import StandardScaler
        def __init__(self):
            self.scale = None

        def transform(self, X):
            temp_X1 = X.reshape((X.shape[0] * X.shape[1], X.shape[2]))
            temp_X1 = self.scale.transform(temp_X1)
            return temp_X1.reshape(X.shape)
```

Fig 28

Baseline Model

In [168]:
```
np.random.seed(0)
tf.set_random_seed(0)
sess = tf.Session(graph=tf.get_default_graph())
K.set_session(sess)
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=7, activation='relu',kernel_initializer
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer
model.add(Dropout(0.6))
model.add(MaxPooling1D(pool_size=3))
model.add(Flatten())
model.add(Dense(30, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 122, 64)           4096
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 120, 32)           6176
_________________________________________________________________
dropout_1 (Dropout)          (None, 120, 32)           0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 40, 32)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1280)              0
_________________________________________________________________
dense_1 (Dense)              (None, 30)                38430
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 93
=================================================================
Total params: 48,795
Trainable params: 48,795
Non-trainable params: 0
_________________________________________________________________
```

Fig 29 **Applying CNN model on Dynamic Activities**

```
In [10]: X_train_d, Y_train_d, X_val_d, Y_val_d = data_scaled_dynamic()
         trials = Trials()
         best_run, best_model, space = optim.minimize(model=model_cnn,
                                               data=data_scaled_dynamic,
                                               algo=tpe.suggest,
                                               max_evals=120,rseed = 0,
                                               trials=trials,notebook_name='Human Activity Detection',
                                               return_space = True)

>>> Imports:
#coding=utf-8

try:
    import os
except:
    pass

try:
    import numpy as np
except:
    pass

try:
    import tensorflow as tf
except:
    pass

try:
```

```
In [11]: from hyperas.utils import eval_hyperopt_space
         total_trials = dict()
         for t, trial in enumerate(trials):
                 vals = trial.get('misc').get('vals')
                 z = eval_hyperopt_space(space, vals)
                 total_trials['M'+str(t+1)] = z
         #best Hyper params from hyperas
         best_params = eval_hyperopt_space(space, best_run)
         best_params
```

Fig 30 Hyperparameter Tuning using Hyperas

Final Prediction

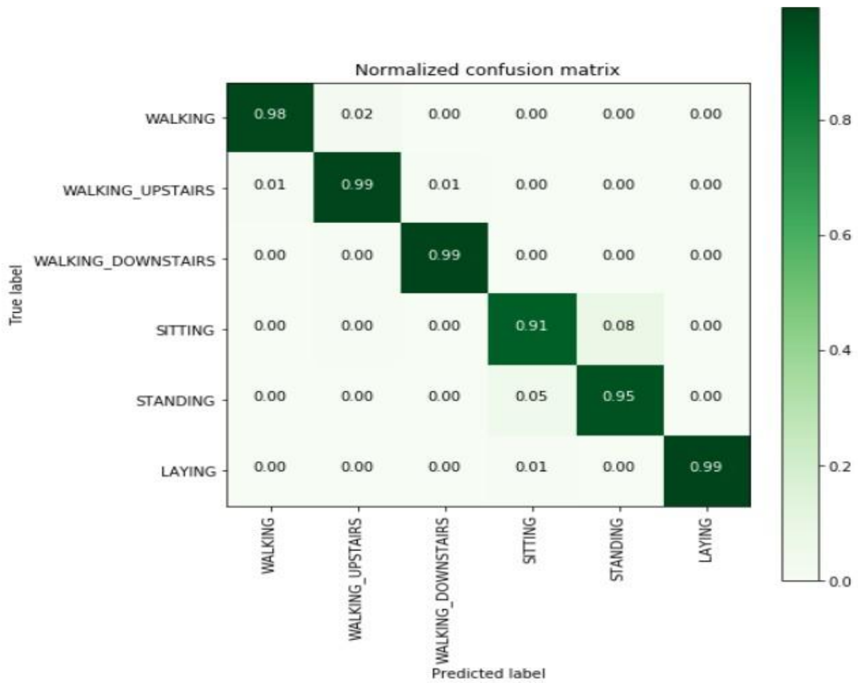
Final prediction pipeline

```
In [159]: ##loading keras models and picle files for scaling data
          from keras.models import load_model
          import pickle
          model_2class = load_model('final_model_2class.h5')
          model_dynamic = load_model('final_model_dynamic.h5')
          model_static = load_model('final_model_static.h5')
          scale_2class = pickle.load(open('Scale_2class.p','rb'))
          scale_static = pickle.load(open('Scale_static.p','rb'))
          scale_dynamic = pickle.load(open('Scale_dynamic.p','rb'))
```

```
In [162]: ##scaling the data
          def transform_data(X,scale):
              X_temp = X.reshape((X.shape[0] * X.shape[1], X.shape[2]))
              X_temp = scale.transform(X_temp)
              return X_temp.reshape(X.shape)
```

```
In [169]: #predicting output activity
          def predict_activity(X):
              ##predicting whether dynamic or static
              predict_2class = model_2class.predict(transform_data(X,scale_2class))
              Y_pred_2class =  np.argmax(predict_2class, axis=1)
              #static data filter
              X_static = X[Y_pred_2class==1]
              #dynamic data filter
              X_dynamic = X[Y_pred_2class==0]
              #predicting static activities
              predict_static = model_static.predict(transform_data(X_static,scale_static))
              predict_static = np.argmax(predict_static,axis=1)
              #adding 4 because need to get inal prediction lable as output
              predict_static = predict_static + 4
              #predicting dynamic activites
              predict_dynamic = model_dynamic.predict(transform_data(X_dynamic,scale_dynamic))
              predict_dynamic = np.argmax(predict_dynamic,axis=1)
              #adding 1 because need to get inal prediction lable as output
              predict_dynamic = predict_dynamic + 1
              ##appending final output to one list in the same sequence of input data
```

Fig 31:



Divide and Conquer approch with CNN is giving good result with final test accuracy of ~0.97. and train accuracy ~0.98.

Fig 32

The Classification accuracies given by the CNN model on classifying both Static and dynamic activities are fitted into a final pipeline in order to give the combined accuracy in classifying all the 6 human activities. The combined CNN model after building different CNN models based on divide and conquer based approach are evaluated based on confusion Matrix where the final pipeline model is classified as both Static and dynamic activities in a single model.