Importing Libraries

In [1]:

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from sklearn.model_selection import train_test_split
from tqdm import tqdm

from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense,Dropout,CuDNNLSTM,BatchNormalization
from keras.layers import Conv1D,MaxPooling1D,Flatten
from keras.layers.embeddings import Embedding
import numpy as np
```

```
Using TensorFlow backend.
```

In [5]:

```
import pickle

def dump_file(filename, mode, data):
    '''
    Save model on the disk
    '''
    pickle.dump(data, open(filename, mode))
```

Load the Signals (Input Data)

In [13]:

```
import numpy as np
import pandas as pd

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

```
No of Features: 561
```

In [8]:

```
# 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 [9]:

```
# 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',
}
```

In [10]:

```
# 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).to_numpy()
        )

    # 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 [14]:

```
from sklearn.preprocessing import StandardScaler

def load_y_static_dynamic(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.htm
l)
        """
        filename = f'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
        y = _read_csv(filename)[0]
        y[y<=3] = 0
        y[y>3] = 1
        return pd.get_dummies(y).to_numpy()

def load_data_static_dynamic():
    '''
    Load train, test data and scale the data as well
    '''
    X_train_2c, X_val_2c = load_signals('train'), load_signals('test')
    Y_train_2c, Y_val_2c = load_y_static_dynamic('train'), load_y_static_dynamic('test'
)

    # fit and transform data
    Scale = fit(X_train_2c)
    dump_file('Scale_2class.p','wb', Scale)
    X_train_2c = transform(X_train_2c, Scale)
    X_val_2c = transform(X_val_2c, Scale)

    return X_train_2c, Y_train_2c, X_val_2c, Y_val_2c

def transform(X, scale):
    '''
    Transform the data
    '''
    temp_X1 = X.reshape((X.shape[0] * X.shape[1], X.shape[2]))
    temp_X1 = scale.transform(temp_X1)
    return temp_X1.reshape(X.shape)

def fit(X):
    '''
    Fit data for scaling
    '''
    # remove overlaping
    remove = int(X.shape[1] / 2)
    temp_X = X[:, -remove:, :]
    # flatten data
    temp_X = temp_X.reshape((temp_X.shape[0] * temp_X.shape[1], temp_X.shape[2]))
    scale = StandardScaler()
    scale.fit(temp_X)
    return scale
```

In [15]:

```
X_train_2c, Y_train_2c, X_val_2c,  Y_val_2c = load_data_static_dynamic()
```

In [16]:

```
print(X_train_2c.shape)
print(X_val_2c.shape)
print(Y_train_2c.shape)
print(Y_val_2c.shape)
```

```
(7352, 128, 9)
(2947, 128, 9)
(7352, 2)
(2947, 2)
```

Fucntion for Confusion Matrix

In [17]:

```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Utility function to print the confusion matrix
def confusion_matrix_cnn(Y_true, Y_pred,activities):
    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'])
    return confusion_matrix(Y_true, Y_pred)
```

Loading the Output labels by spliting into Static and Dynamic

1. walking, up, down -- dynamic
2. sitting standing lying -- static

In [20]:

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

In [21]:

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

In [22]:

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

In [23]:

```
# Importing Libraries
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout
from keras.layers.normalization import BatchNormalization
```

In [24]:

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

In [25]:

```
# update LSTM layers
n_hidden_1 = 32
n_hidden_2 = 16
```

Model for classifying data into Static and Dynamic activities

In [1]:

```
# https://github.com/mayank171986/Human-Activity-Detection/blob/master/human-activity-d
etection.ipynb
```

In [26]:

```
np.random.seed(42)
tf.set_random_seed(42)
# Start Session
sess = tf.Session(graph=tf.get_default_graph())
K.set_session(sess)
# Create model
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer='he_un
iform',input_shape=(128,9)))
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer='he_un
iform'))
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()
```

```
_________________________________________________________________
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
_________________________________________________________________
```

In [28]:

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train_2c,Y_train_2c, epochs=20, batch_size=16,validation_data=(X_val_2c, Y_
val_2c), verbose=1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/20
7352/7352 [==============================] - 65s 9ms/step - loss: 0.0360 -
acc: 0.9869 - val_loss: 0.0746 - val_acc: 0.9783
Epoch 2/20
7352/7352 [==============================] - 63s 9ms/step - loss: 0.0042 -
acc: 0.9989 - val_loss: 0.0294 - val_acc: 0.9936
Epoch 3/20
7352/7352 [==============================] - 64s 9ms/step - loss: 6.4344e-
04 - acc: 0.9999 - val_loss: 0.0125 - val_acc: 0.9973
Epoch 4/20
7352/7352 [==============================] - 65s 9ms/step - loss: 1.2485e-
04 - acc: 1.0000 - val_loss: 0.0175 - val_acc: 0.9966
Epoch 5/20
7352/7352 [==============================] - 67s 9ms/step - loss: 1.4892e-
04 - acc: 1.0000 - val_loss: 0.0207 - val_acc: 0.9963
Epoch 6/20
7352/7352 [==============================] - 51s 7ms/step - loss: 0.0072 -
acc: 0.9984 - val_loss: 0.0100 - val_acc: 0.9986
Epoch 7/20
7352/7352 [==============================] - 8s 1ms/step - loss: 2.0327e-0
5 - acc: 1.0000 - val_loss: 0.0091 - val_acc: 0.9990
Epoch 8/20
7352/7352 [==============================] - 9s 1ms/step - loss: 1.3286e-0
6 - acc: 1.0000 - val_loss: 0.0091 - val_acc: 0.9990
Epoch 9/20
7352/7352 [==============================] - 8s 1ms/step - loss: 1.9687e-0
6 - acc: 1.0000 - val_loss: 0.0089 - val_acc: 0.9990
Epoch 10/20
7352/7352 [==============================] - 8s 1ms/step - loss: 1.9564e-0
6 - acc: 1.0000 - val_loss: 0.0090 - val_acc: 0.9990
Epoch 11/20
7352/7352 [==============================] - 8s 1ms/step - loss: 3.8553e-0
6 - acc: 1.0000 - val_loss: 0.0082 - val_acc: 0.9990
Epoch 12/20
7352/7352 [==============================] - 8s 1ms/step - loss: 9.0197e-0
7 - acc: 1.0000 - val_loss: 0.0086 - val_acc: 0.9990
Epoch 13/20
7352/7352 [==============================] - 8s 1ms/step - loss: 0.0075 -
acc: 0.9989 - val_loss: 0.0026 - val_acc: 0.9986
Epoch 14/20
7352/7352 [==============================] - 8s 1ms/step - loss: 3.0977e-0
4 - acc: 0.9997 - val_loss: 0.0033 - val_acc: 0.9990
Epoch 15/20
7352/7352 [==============================] - 8s 1ms/step - loss: 2.0082e-0
5 - acc: 1.0000 - val_loss: 0.0032 - val_acc: 0.9990
Epoch 16/20
7352/7352 [==============================] - 8s 1ms/step - loss: 0.0011 -
acc: 0.9997 - val_loss: 0.0060 - val_acc: 0.9986
Epoch 17/20
7352/7352 [==============================] - 8s 1ms/step - loss: 1.3465e-0
7 - acc: 1.0000 - val_loss: 0.0060 - val_acc: 0.9986
Epoch 18/20
7352/7352 [==============================] - 9s 1ms/step - loss: 8.3655e-0
7 - acc: 1.0000 - val_loss: 0.0057 - val_acc: 0.9986
Epoch 19/20
7352/7352 [==============================] - 9s 1ms/step - loss: 1.3062e-0
7 - acc: 1.0000 - val_loss: 0.0057 - val_acc: 0.9986
Epoch 20/20
7352/7352 [==============================] - 8s 1ms/step - loss: 1.3181e-0
7 - acc: 1.0000 - val_loss: 0.0057 - val_acc: 0.9986
```

Out[28]:

```
<keras.callbacks.History at 0x1d3a1e890f0>
```

In [29]:

```
_,acc_val = model.evaluate(X_val_2c,Y_val_2c,verbose=0)
_,acc_train = model.evaluate(X_train_2c,Y_train_2c,verbose=0)
print('Train_accuracy',acc_train,'test_accuracy',acc_val)
```

```
Train_accuracy 1.0 test_accuracy 0.998642687478792
```

Save the 2 class classification model

In [30]:

```
##saving model
model.save('final_model_2class.h5')
```

Classificaton of Static activities

In [33]:

```
def load_y_static(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.htm
l)
        """
        filename = f'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
        y = _read_csv(filename)[0]
        y_subset = y>3
        y = y[y_subset]
        return pd.get_dummies(y).to_numpy(),y_subset

def load_data_static():
    '''
    Load train, test data and scale the data as well
    '''
    Y_train_s, y_train_sub = load_y_static('train')
    Y_val_s, y_test_sub = load_y_static('test')

    X_train_s, X_val_s = load_signals('train'), load_signals('test')
    X_train_s = X_train_s[y_train_sub]
    X_val_s = X_val_s[y_test_sub]

    # fit and transform data
    Scale = None
    Scale = fit(X_train_s)
    dump_file('Scale_static.p','wb', Scale)
    X_train_s = transform(X_train_s, Scale)
    X_val_s = transform(X_val_s, Scale)

    return X_train_s, Y_train_s, X_val_s, Y_val_s
```

In [34]:

```
X_train_s, Y_train_s, X_val_s,  Y_val_s = load_data_static()
```

In [35]:

```
print('X Shape of train data',X_train_s.shape, 'Y shape', Y_train_s.shape)
print('X Shape of val data',X_val_s.shape,'Y shape',Y_val_s.shape)
```

```
X Shape of train data (4067, 128, 9) Y shape (4067, 3)
X Shape of val data (1560, 128, 9) Y shape (1560, 3)
```

Model for Static Activities

In [36]:

```
# Clear session
K.clear_session()
# Random seed
np.random.seed(42)
tf.set_random_seed(42)
# Start session
sess = tf.Session(graph=tf.get_default_graph())
K.set_session(sess)
# Define the model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=7, activation='relu',kernel_initializer='he_un
iform',input_shape=(128,9)))
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer='he_un
iform'))
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()
```

```
_________________________________________________________________
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
_________________________________________________________________
```

In [37]:

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train_s,Y_train_s, epochs=20, batch_size=32,validation_data=(X_val_s, Y_val
_s), verbose=1)
# K.clear_session()
```

```
Train on 4067 samples, validate on 1560 samples
Epoch 1/20
4067/4067 [==============================] - 8s 2ms/step - loss: 0.4232 -
acc: 0.8480 - val_loss: 0.3000 - val_acc: 0.8731
Epoch 2/20
4067/4067 [==============================] - 6s 2ms/step - loss: 0.1956 -
acc: 0.9184 - val_loss: 0.2661 - val_acc: 0.8936
Epoch 3/20
4067/4067 [==============================] - 12s 3ms/step - loss: 0.1658 -
acc: 0.9312 - val_loss: 0.2919 - val_acc: 0.8705
Epoch 4/20
4067/4067 [==============================] - 7s 2ms/step - loss: 0.1749 -
acc: 0.9289 - val_loss: 0.3007 - val_acc: 0.8962
Epoch 5/20
4067/4067 [==============================] - 14s 3ms/step - loss: 0.1388 -
acc: 0.9454 - val_loss: 0.2798 - val_acc: 0.8878
Epoch 6/20
4067/4067 [==============================] - 12s 3ms/step - loss: 0.1358 -
acc: 0.9452 - val_loss: 0.2305 - val_acc: 0.9276
Epoch 7/20
4067/4067 [==============================] - 24s 6ms/step - loss: 0.1374 -
acc: 0.9471 - val_loss: 0.2337 - val_acc: 0.9141
Epoch 8/20
4067/4067 [==============================] - 22s 6ms/step - loss: 0.1198 -
acc: 0.9503 - val_loss: 0.2678 - val_acc: 0.8994
Epoch 9/20
4067/4067 [==============================] - 22s 5ms/step - loss: 0.1001 -
acc: 0.9582 - val_loss: 0.2387 - val_acc: 0.9321
Epoch 10/20
4067/4067 [==============================] - 23s 6ms/step - loss: 0.1193 -
acc: 0.9555 - val_loss: 0.2570 - val_acc: 0.9231
Epoch 11/20
4067/4067 [==============================] - 23s 6ms/step - loss: 0.1138 -
acc: 0.9582 - val_loss: 0.2125 - val_acc: 0.9333
Epoch 12/20
4067/4067 [==============================] - 18s 4ms/step - loss: 0.0933 -
acc: 0.9656 - val_loss: 0.2326 - val_acc: 0.9263
Epoch 13/20
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0910 -
acc: 0.9671 - val_loss: 0.1895 - val_acc: 0.9429
Epoch 14/20
4067/4067 [==============================] - 5s 1ms/step - loss: 0.0890 -
acc: 0.9666 - val_loss: 0.2075 - val_acc: 0.9359
Epoch 15/20
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0680 -
acc: 0.9749 - val_loss: 0.1895 - val_acc: 0.9391
Epoch 16/20
4067/4067 [==============================] - 10s 2ms/step - loss: 0.0830 -
acc: 0.9685 - val_loss: 0.1985 - val_acc: 0.9365
Epoch 17/20
4067/4067 [==============================] - 28s 7ms/step - loss: 0.0578 -
acc: 0.9752 - val_loss: 0.1765 - val_acc: 0.9423
Epoch 18/20
4067/4067 [==============================] - 21s 5ms/step - loss: 0.0552 -
acc: 0.9806 - val_loss: 0.2349 - val_acc: 0.9147
Epoch 19/20
4067/4067 [==============================] - 6s 1ms/step - loss: 0.0877 -
acc: 0.9698 - val_loss: 0.2149 - val_acc: 0.9340
Epoch 20/20
4067/4067 [==============================] - 8s 2ms/step - loss: 0.0459 -
acc: 0.9808 - val_loss: 0.1936 - val_acc: 0.9346
```

Out[37]:

```
<keras.callbacks.History at 0x1d3a3588c18>
```

In [38]:

```python
_,acc_val = model.evaluate(X_val_s, Y_val_s,verbose=0)
_,acc_train = model.evaluate(X_train_s,Y_train_s,verbose=0)
print('Train_accuracy',acc_train,'test_accuracy',acc_val)
```

```
Train_accuracy 0.9857388738627981 test_accuracy 0.9346153846153846
```

In [39]:

```python
##saving model
model.save('final_model_static.h5')
```

In [42]:

```python
def load_y_dynamic(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = f'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
    y = _read_csv(filename)[0]
    y_subset = y<=3
    y = y[y_subset]
    return pd.get_dummies(y).to_numpy(),y_subset

def load_data_dynamic():
    '''
    Load train, test data and scale the data as well
    '''
    Y_train_d, y_train_sub = load_y_dynamic('train')
    Y_val_d, y_test_sub = load_y_dynamic('test')

    X_train_d, X_val_d = load_signals('train'), load_signals('test')
    X_train_d = X_train_d[y_train_sub]
    X_val_d = X_val_d[y_test_sub]

    # fit and transform data
    Scale = None
    Scale = fit(X_train_d)
    dump_file('Scale_dynamic.p','wb', Scale)
    X_train_d = transform(X_train_d, Scale)
    X_val_d = transform(X_val_d, Scale)

    return X_train_d, Y_train_d, X_val_d, Y_val_d
```

In [43]:

```python
X_train_d, Y_train_d, X_val_d,  Y_val_d = load_data_dynamic()
```

In [44]:

```
print('Train X shape',X_train_d.shape,'Test X shape',X_val_d.shape)
print('Train Y shape',Y_train_d.shape,'Test Y shape',Y_val_d.shape)
```

```
Train X shape (3285, 128, 9) Test X shape (1387, 128, 9)
Train Y shape (3285, 3) Test Y shape (1387, 3)
```

In [45]:

```
print('Train X shape',X_train_d.shape,'Test X shape',X_val_d.shape)
print('Train Y shape',Y_train_d.shape,'Test Y shape',Y_val_d.shape)
```

```
Train X shape (3285, 128, 9) Test X shape (1387, 128, 9)
Train Y shape (3285, 3) Test Y shape (1387, 3)
```

Model for Dynamic

In [47]:

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train_d,Y_train_d, epochs=20, batch_size=32,validation_data=(X_val_d, Y_val
_d), verbose=1)
# K.clear_session()
```

```
Train on 3285 samples, validate on 1387 samples
Epoch 1/20
3285/3285 [==============================] - 7s 2ms/step - loss: 0.8125 -
acc: 0.7260 - val_loss: 0.4112 - val_acc: 0.8659
Epoch 2/20
3285/3285 [==============================] - 4s 1ms/step - loss: 0.1114 -
acc: 0.9623 - val_loss: 0.2984 - val_acc: 0.8904
Epoch 3/20
3285/3285 [==============================] - 6s 2ms/step - loss: 0.0435 -
acc: 0.9857 - val_loss: 0.1599 - val_acc: 0.9430
Epoch 4/20
3285/3285 [==============================] - 13s 4ms/step - loss: 0.0211 -
acc: 0.9936 - val_loss: 0.1888 - val_acc: 0.9221
Epoch 5/20
3285/3285 [==============================] - 6s 2ms/step - loss: 0.0131 -
acc: 0.9951 - val_loss: 0.1602 - val_acc: 0.9402
Epoch 6/20
3285/3285 [==============================] - 13s 4ms/step - loss: 0.0122 -
acc: 0.9951 - val_loss: 0.0874 - val_acc: 0.9683
Epoch 7/20
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0033 -
acc: 0.9991 - val_loss: 0.0939 - val_acc: 0.9661
Epoch 8/20
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0034 -
acc: 0.9985 - val_loss: 0.1045 - val_acc: 0.9704
Epoch 9/20
3285/3285 [==============================] - 4s 1ms/step - loss: 0.0019 -
acc: 0.9994 - val_loss: 0.1290 - val_acc: 0.9596
Epoch 10/20
3285/3285 [==============================] - 4s 1ms/step - loss: 0.0032 -
acc: 0.9988 - val_loss: 0.1004 - val_acc: 0.9654
Epoch 11/20
3285/3285 [==============================] - 5s 1ms/step - loss: 0.0013 -
acc: 0.9997 - val_loss: 0.0822 - val_acc: 0.9726
Epoch 12/20
3285/3285 [==============================] - 5s 1ms/step - loss: 5.3048e-0
4 - acc: 1.0000 - val_loss: 0.1107 - val_acc: 0.9690
Epoch 13/20
3285/3285 [==============================] - 4s 1ms/step - loss: 0.0069 -
acc: 0.9973 - val_loss: 0.0966 - val_acc: 0.9704
Epoch 14/20
3285/3285 [==============================] - 4s 1ms/step - loss: 0.0024 -
acc: 0.9994 - val_loss: 0.0753 - val_acc: 0.9798
Epoch 15/20
3285/3285 [==============================] - 6s 2ms/step - loss: 0.0172 -
acc: 0.9942 - val_loss: 0.1140 - val_acc: 0.9596
Epoch 16/20
3285/3285 [==============================] - 10s 3ms/step - loss: 0.0081 -
acc: 0.9973 - val_loss: 0.1101 - val_acc: 0.9668
Epoch 17/20
3285/3285 [==============================] - 12s 4ms/step - loss: 0.0017 -
acc: 0.9997 - val_loss: 0.1460 - val_acc: 0.9697
Epoch 18/20
3285/3285 [==============================] - 6s 2ms/step - loss: 0.0032 -
acc: 0.9985 - val_loss: 0.0696 - val_acc: 0.9712
Epoch 19/20
3285/3285 [==============================] - 11s 3ms/step - loss: 4.2772e-
04 - acc: 1.0000 - val_loss: 0.0678 - val_acc: 0.9769
Epoch 20/20
3285/3285 [==============================] - 7s 2ms/step - loss: 0.0034 -
acc: 0.9994 - val_loss: 0.1442 - val_acc: 0.9676
```

```
Out[47]:

<keras.callbacks.History at 0x1d3a3a42eb8>
```

Output For Dynamic Activities

In [48]:

```
_,acc_val = model.evaluate(X_val_d, Y_val_d,verbose=0)
_,acc_train = model.evaluate(X_train_d,Y_train_d,verbose=0)
print('Train_accuracy',acc_train,'test_accuracy',acc_val)
```

```
Train_accuracy 1.0 test_accuracy 0.9675558759913482
```

In [49]:

```
##saving model
model.save('final_model_dynamic.h5')
```

In [50]:

```
def load_y_whole_data(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 y

def load_whole_data():
    '''
    Load and split whole data
    '''
    X_train, X_val = load_signals('train'), load_signals('test')
    Y_train, Y_val = load_y_whole_data('train'), load_y_whole_data('test')

    return X_train, Y_train, X_val, Y_val
```

In [51]:

```
X_train, Y_train, X_val, Y_val = load_whole_data()
```

In [52]:

```
print('shape of train X',X_train.shape, 'shape of train Y',Y_train.shape)
print('shape of test X', X_val.shape, 'shape of test Y', Y_val.shape)
```

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

Final Prediction pipeline

In [53]:

```
#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 [54]:

```
##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)
```

Evaluate Predictions

In [55]:

```
#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 final 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 final prediction lable as output
    predict_dynamic = predict_dynamic + 1
    ##appending final output to one list in the same sequence of input data
    i,j = 0,0
    final_pred = []
    for mask in Y_pred_2class:
        if mask == 1:
            final_pred.append(predict_static[i])
            i = i + 1
        else:
            final_pred.append(predict_dynamic[j])
            j = j + 1
    return final_pred
```

In [56]:

```
##predicting
final_pred_val = predict_activity(X_val)
final_pred_train = predict_activity(X_train)
```

In [57]:

```
##accuracy of train and test
from sklearn.metrics import accuracy_score
print('Accuracy of train data',accuracy_score(Y_train,final_pred_train))
print('Accuracy of validation data',accuracy_score(Y_val,final_pred_val))
```

```
Accuracy of train data 0.9921109902067464
Accuracy of validation data 0.9487614523243977
```

- Best test accuracy obtained is 94.87% using divide and conquer.