In [1]:

```python
# Importing all the necessary packages and libraries

import pandas as pd
import numpy as np
import tensorflow as tf
np.random.seed(42)
tf.set_random_seed(42)

from keras import backend as K
from keras.models import Sequential
from keras.layers import LSTM, Conv1D, MaxPooling1D, Flatten, BatchNormalization
from keras.layers.core import Dense, Dropout
from keras.regularizers import l1, l2, l1_l2
from sklearn.metrics import accuracy_score
```

```
C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\tensorflow\python\framework\dtypes.py:516: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\tensorflow\python\framework\dtypes.py:517: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\tensorflow\python\framework\dtypes.py:518: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\tensorflow\python\framework\dtypes.py:519: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\tensorflow\python\framework\dtypes.py:520: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:541: FutureWarning: Passing (type, 1) or '1t
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:542: FutureWarning: Passing (type, 1) or '1t
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:543: FutureWarning: Passing (type, 1) or '1t
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:544: FutureWarning: Passing (type, 1) or '1t
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:545: FutureWarning: Passing (type, 1) or '1t
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:550: FutureWarning: Passing (type, 1) or '1t
```

```
packages\tensorboard\compat\tensorflow_stub\dtypes.py:550: FutureWarning: Passing (type, 1) or '1t
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
Using TensorFlow backend.
```

In [30]:

```
# Labelling the 6 classes
ACTIVITIES = {
    0: 'WALKING',
    1: 'WALKING_UPSTAIRS',
    2: 'WALKING_DOWNSTAIRS',
    3: 'SITTING',
    4: 'STANDING',
    5: 'LAYING',
}

# Function for Confusion Matrix
def confusion_matrix2(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'])
```

In [3]:

```
DATADIR = 'UCI_HAR_Dataset'

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 [4]:

```
# Function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Function to load the signals data
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()
        )
    return np.transpose(signals_data, (1, 2, 0))


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

    return y.values

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 [5]:

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

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

In [6]:

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

# Loading the Train and Test Data
X_train, X_test, Y_train, Y_test = load_data()

y_train_dif, y_test_dif = pd.Series(Y_train).map(dict(zip(range(1,7), [1]*3+[0]*3))).values,
pd.Series(Y_test).map(dict(zip(range(1,7), [1]*3+[0]*3))).values

# Dynamic class data
X_train_Dynamic, X_test_Dynamic = X_train[y_train_dif==1], X_test[y_test_dif==1]
Y_train_Dynamic, Y_test_Dynamic = Y_train[y_train_dif==1], Y_test[y_test_dif==1]

# Static class data
X_train_Static, X_test_Static = X_train[y_train_dif==0], X_test[y_test_dif==0]
Y_train_Static, Y_test_Static = Y_train[y_train_dif==0], Y_test[y_test_dif==0]

y_train_dif, y_test_dif = pd.get_dummies(y_train_dif).values,pd.get_dummies(y_test_dif).values
Y_train_Dynamic, Y_test_Dynamic = pd.get_dummies(Y_train_Dynamic).values, pd.get_dummies(Y_test_Dyn
amic).values
Y_train_Static, Y_test_Static = pd.get_dummies(Y_train_Static).values, pd.get_dummies(Y_test_Static
).values

timesteps = len(X_train[0])
input_dim = len(X_train[0][0])

print("Time steps : ", timesteps)
print("Input dimensions : ", input_dim)
print("Len of X_train : ", len(X_train))
```

```
Time steps :  128
Input dimensions :  9
Len of X_train :  7352
```

Divide and Conquer CNN Model

2- class Classifier

In [7]:

```
model = Sequential()

model.add(Conv1D(16, 3, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.
0001), input_shape=(timesteps, input_dim)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(16, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.001)
))
```

```
model.add(BatchNormalization())
model.add(Dropout(0.65))
model.add(Dense(2, activation='softmax'))

model.summary()
```

```
WARNING:tensorflow:From C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\keras\backend\tensorflow_backend.py:4070: The name tf.nn.max_pool is deprecated. Please u
se tf.nn.max_pool2d instead.

WARNING:tensorflow:Large dropout rate: 0.65 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate
instead of keep_prob. Please ensure that this is intended.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 126, 16)           448
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 63, 16)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1008)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1008)              0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                16144
_________________________________________________________________
batch_normalization_1 (Batch (None, 16)                64
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 34
=================================================================
Total params: 16,690
Trainable params: 16,658
Non-trainable params: 32
_________________________________________________________________
```

In [8]:

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

# Training the model
model.fit(X_train,
          y_train_dif,
          batch_size=8,
          validation_data=(X_test, y_test_dif),
          epochs=20)
```

```
WARNING:tensorflow:From C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\tensorflow\python\ops\math_grad.py:1250: add_dispatch_support.<locals>.wrapper (from
tensorflow.python.ops.array_ops) is deprecated 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
WARNING:tensorflow:From C:\Users\Karan Kapadia\Anaconda3\envs\workspace\lib\site-
packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables is deprecated. Plea
se use tf.compat.v1.global_variables instead.

Train on 7352 samples, validate on 2947 samples
Epoch 1/20
7352/7352 [==============================] - 9s 1ms/step - loss: 0.3187 - accuracy: 0.8904 - val_l
oss: 0.0747 - val_accuracy: 0.9844
Epoch 2/20
7352/7352 [==============================] - 6s 834us/step - loss: 0.1425 - accuracy: 0.9652 - val
_loss: 0.0580 - val_accuracy: 0.9939
Epoch 3/20
7352/7352 [==============================] - 6s 850us/step - loss: 0.1350 - accuracy: 0.9693 - val
_loss: 0.0535 - val_accuracy: 0.9949
Epoch 4/20
7352/7352 [==============================] - 6s 843us/step - loss: 0.1417 - accuracy: 0.9706 - val
_loss: 0.0566 - val_accuracy: 0.9942
Epoch 5/20
7352/7352 [==============================] - 6s 780us/step - loss: 0.1061 - accuracy: 0.9818 - val
```

```
_loss: 0.0527 - val_accuracy: 0.9922
Epoch 6/20
7352/7352 [==============================] - 6s 780us/step - loss: 0.0961 - accuracy: 0.9835 - val
_loss: 0.0506 - val_accuracy: 0.9922
Epoch 7/20
7352/7352 [==============================] - 6s 781us/step - loss: 0.0892 - accuracy: 0.9837 - val
_loss: 0.0446 - val_accuracy: 0.9963
Epoch 8/20
7352/7352 [==============================] - 6s 840us/step - loss: 0.0971 - accuracy: 0.9819 - val
_loss: 0.0545 - val_accuracy: 0.9881
Epoch 9/20
7352/7352 [==============================] - 6s 852us/step - loss: 0.0866 - accuracy: 0.9856 - val
_loss: 0.0383 - val_accuracy: 0.9983
Epoch 10/20
7352/7352 [==============================] - 6s 842us/step - loss: 0.0993 - accuracy: 0.9801 - val
_loss: 0.0490 - val_accuracy: 0.9898
Epoch 11/20
7352/7352 [==============================] - 6s 876us/step - loss: 0.0837 - accuracy: 0.9879 - val
_loss: 0.0393 - val_accuracy: 0.9963
Epoch 12/20
7352/7352 [==============================] - 6s 795us/step - loss: 0.0788 - accuracy: 0.9874 - val
_loss: 0.0336 - val_accuracy: 0.9990
Epoch 13/20
7352/7352 [==============================] - 6s 777us/step - loss: 0.1005 - accuracy: 0.9808 - val
_loss: 0.0399 - val_accuracy: 0.9969
Epoch 14/20
7352/7352 [==============================] - 6s 789us/step - loss: 0.0772 - accuracy: 0.9883 - val
_loss: 0.0357 - val_accuracy: 0.9983
Epoch 15/20
7352/7352 [==============================] - 6s 810us/step - loss: 0.0737 - accuracy: 0.9874 - val
_loss: 0.0290 - val_accuracy: 0.9993
Epoch 16/20
7352/7352 [==============================] - 6s 831us/step - loss: 0.0677 - accuracy: 0.9897 - val
_loss: 0.0288 - val_accuracy: 0.9980
Epoch 17/20
7352/7352 [==============================] - 6s 778us/step - loss: 0.0681 - accuracy: 0.9882 - val
_loss: 0.0237 - val_accuracy: 0.9997
Epoch 18/20
7352/7352 [==============================] - 6s 779us/step - loss: 0.0649 - accuracy: 0.9890 - val
_loss: 0.0238 - val_accuracy: 0.9993
Epoch 19/20
7352/7352 [==============================] - 6s 772us/step - loss: 0.0478 - accuracy: 0.9913 - val
_loss: 0.0204 - val_accuracy: 0.9997
Epoch 20/20
7352/7352 [==============================] - 6s 779us/step - loss: 0.0447 - accuracy: 0.9922 - val
_loss: 0.0158 - val_accuracy: 1.0000
```

Out[8]:

```
<keras.callbacks.callbacks.History at 0x18f9bfd84c8>
```

In [9]:

```
# Confusion Matrix
print(confusion_matrix2(y_test_dif, model.predict(X_test), {0: 'Static', 1: 'Dynamic',}))
```

```
Pred     Dynamic  Static
True
Dynamic     1387       0
Static         0    1560
```

In [10]:

```
score = model.evaluate(X_test, y_test_dif)
print(score)
```

```
2947/2947 [==============================] - 0s 74us/step
[0.01575663369774697, 1.0]
```

In [11]:

```
model.save('class_model.h5')
```

Observations

- 2- class classifer has 100 % validation accuracy.
- Which means that our model can perfectly distinguish static and dynamic activities.

Model for Dynamic Class

In [12]:

```
model = Sequential()

model.add(Conv1D(64, 3, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.
0001), input_shape=(timesteps, input_dim)))
model.add(Conv1D(32, 3, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.
001), input_shape=(timesteps, input_dim)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dropout(0.6))
model.add(Dense(32, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.001)
))
model.add(BatchNormalization())
model.add(Dropout(0.6))
model.add(Dense(3, activation='softmax'))

model.summary()
```

```
WARNING:tensorflow:Large dropout rate: 0.6 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate
instead of keep_prob. Please ensure that this is intended.
WARNING:tensorflow:Large dropout rate: 0.6 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate
instead of keep_prob. Please ensure that this is intended.
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_2 (Conv1D)            (None, 126, 64)           1792      
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 124, 32)           6176      
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 62, 32)            0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 1984)              0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 1984)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 32)                63520     
_________________________________________________________________
batch_normalization_2 (Batch (None, 32)                128       
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 99        
=================================================================
Total params: 71,715
Trainable params: 71,651
Non-trainable params: 64
_________________________________________________________________
```

In [13]:

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

# Training the model
model.fit(X_train_Dynamic,
          Y_train_Dynamic,
          batch_size=8,
          validation_data=(X_test_Dynamic, Y_test_Dynamic),
```

```
        epochs=49)
```

```
Train on 3285 samples, validate on 1387 samples
Epoch 1/49
3285/3285 [==============================] - 3s 1ms/step - loss: 1.5175 - accuracy: 0.4514 - val_l
oss: 1.0317 - val_accuracy: 0.6410
Epoch 2/49
3285/3285 [==============================] - 3s 894us/step - loss: 0.7712 - accuracy: 0.7486 - val
_loss: 0.6647 - val_accuracy: 0.8169
Epoch 3/49
3285/3285 [==============================] - 3s 908us/step - loss: 0.5382 - accuracy: 0.8633 - val
_loss: 0.6644 - val_accuracy: 0.7830
Epoch 4/49
3285/3285 [==============================] - 3s 891us/step - loss: 0.4462 - accuracy: 0.8992 - val
_loss: 0.4809 - val_accuracy: 0.8782
Epoch 5/49
3285/3285 [==============================] - 3s 867us/step - loss: 0.3918 - accuracy: 0.9199 - val
_loss: 0.3956 - val_accuracy: 0.9005
Epoch 6/49
3285/3285 [==============================] - 3s 850us/step - loss: 0.3862 - accuracy: 0.9285 - val
_loss: 0.5752 - val_accuracy: 0.8659
Epoch 7/49
3285/3285 [==============================] - 3s 849us/step - loss: 0.3321 - accuracy: 0.9467 - val
_loss: 0.3282 - val_accuracy: 0.9402
Epoch 8/49
3285/3285 [==============================] - 3s 869us/step - loss: 0.3094 - accuracy: 0.9540 - val
_loss: 0.3538 - val_accuracy: 0.9358
Epoch 9/49
3285/3285 [==============================] - 3s 841us/step - loss: 0.2876 - accuracy: 0.9592 - val
_loss: 0.3773 - val_accuracy: 0.9250
Epoch 10/49
3285/3285 [==============================] - 3s 887us/step - loss: 0.3026 - accuracy: 0.9565 - val
_loss: 0.3909 - val_accuracy: 0.9466
Epoch 11/49
3285/3285 [==============================] - 3s 910us/step - loss: 0.3115 - accuracy: 0.9507 - val
_loss: 0.5067 - val_accuracy: 0.8767
Epoch 12/49
3285/3285 [==============================] - 3s 903us/step - loss: 0.2994 - accuracy: 0.9556 - val
_loss: 0.2647 - val_accuracy: 0.9567
Epoch 13/49
3285/3285 [==============================] - 3s 863us/step - loss: 0.3018 - accuracy: 0.9534 - val
_loss: 0.2676 - val_accuracy: 0.9575
Epoch 14/49
3285/3285 [==============================] - 3s 845us/step - loss: 0.2704 - accuracy: 0.9616 - val
_loss: 1.2183 - val_accuracy: 0.6979
Epoch 15/49
3285/3285 [==============================] - 3s 870us/step - loss: 0.2856 - accuracy: 0.9613 - val
_loss: 0.3476 - val_accuracy: 0.9430
Epoch 16/49
3285/3285 [==============================] - 3s 887us/step - loss: 0.2892 - accuracy: 0.9571 - val
_loss: 0.3635 - val_accuracy: 0.9308
Epoch 17/49
3285/3285 [==============================] - 3s 830us/step - loss: 0.2732 - accuracy: 0.9623 - val
_loss: 0.2850 - val_accuracy: 0.9553
Epoch 18/49
3285/3285 [==============================] - 3s 857us/step - loss: 0.2661 - accuracy: 0.9619 - val
_loss: 0.3080 - val_accuracy: 0.9531
Epoch 19/49
3285/3285 [==============================] - 3s 905us/step - loss: 0.2626 - accuracy: 0.9623 - val
_loss: 0.2695 - val_accuracy: 0.9654
Epoch 20/49
3285/3285 [==============================] - 3s 895us/step - loss: 0.2981 - accuracy: 0.9546 - val
_loss: 0.2530 - val_accuracy: 0.9676
Epoch 21/49
3285/3285 [==============================] - 3s 847us/step - loss: 0.2415 - accuracy: 0.9723 - val
_loss: 0.4838 - val_accuracy: 0.9041
Epoch 22/49
3285/3285 [==============================] - 3s 837us/step - loss: 0.2403 - accuracy: 0.9686 - val
_loss: 0.3731 - val_accuracy: 0.9438
Epoch 23/49
3285/3285 [==============================] - 3s 842us/step - loss: 0.2152 - accuracy: 0.9763 - val
_loss: 0.6386 - val_accuracy: 0.8508
Epoch 24/49
3285/3285 [==============================] - 3s 835us/step - loss: 0.4051 - accuracy: 0.9151 - val
_loss: 0.2754 - val_accuracy: 0.9719
Epoch 25/49
3285/3285 [==============================] - 3s 835us/step - loss: 0.3170 - accuracy: 0.9364 - val
```

```
3285/3285 [==============================] - 3s 835us/step - loss: 0.3170 - accuracy: 0.9504 - val
_loss: 0.2864 - val_accuracy: 0.9603
Epoch 26/49
3285/3285 [==============================] - 3s 835us/step - loss: 0.3125 - accuracy: 0.9452 - val
_loss: 0.2343 - val_accuracy: 0.9676
Epoch 27/49
3285/3285 [==============================] - 3s 850us/step - loss: 0.2933 - accuracy: 0.9470 - val
_loss: 0.5349 - val_accuracy: 0.8774
Epoch 28/49
3285/3285 [==============================] - 3s 833us/step - loss: 0.2960 - accuracy: 0.9522 - val
_loss: 0.2474 - val_accuracy: 0.9697
Epoch 29/49
3285/3285 [==============================] - 3s 835us/step - loss: 0.2408 - accuracy: 0.9732 - val
_loss: 0.2108 - val_accuracy: 0.9726
Epoch 30/49
3285/3285 [==============================] - 3s 837us/step - loss: 0.2347 - accuracy: 0.9693 - val
_loss: 0.4800 - val_accuracy: 0.9257
Epoch 31/49
3285/3285 [==============================] - 3s 830us/step - loss: 0.2160 - accuracy: 0.9775 - val
_loss: 0.3916 - val_accuracy: 0.9452
Epoch 32/49
3285/3285 [==============================] - 3s 838us/step - loss: 0.2481 - accuracy: 0.9665 - val
_loss: 0.4799 - val_accuracy: 0.9178
Epoch 33/49
3285/3285 [==============================] - 3s 834us/step - loss: 0.2298 - accuracy: 0.9671 - val
_loss: 0.5751 - val_accuracy: 0.9229
Epoch 34/49
3285/3285 [==============================] - 3s 840us/step - loss: 0.2154 - accuracy: 0.9717 - val
_loss: 0.4753 - val_accuracy: 0.8875
Epoch 35/49
3285/3285 [==============================] - 3s 848us/step - loss: 0.2081 - accuracy: 0.9738 - val
_loss: 0.2402 - val_accuracy: 0.9712
Epoch 36/49
3285/3285 [==============================] - 3s 839us/step - loss: 0.2451 - accuracy: 0.9656 - val
_loss: 0.4011 - val_accuracy: 0.9366
Epoch 37/49
3285/3285 [==============================] - 3s 840us/step - loss: 0.2244 - accuracy: 0.9699 - val
_loss: 0.3332 - val_accuracy: 0.9567
Epoch 38/49
3285/3285 [==============================] - 3s 905us/step - loss: 0.2290 - accuracy: 0.9717 - val
_loss: 0.3263 - val_accuracy: 0.9488
Epoch 39/49
3285/3285 [==============================] - 3s 889us/step - loss: 0.2029 - accuracy: 0.9781 - val
_loss: 0.2532 - val_accuracy: 0.9668
Epoch 40/49
3285/3285 [==============================] - 3s 851us/step - loss: 0.2285 - accuracy: 0.9723 - val
_loss: 0.6764 - val_accuracy: 0.9041
Epoch 41/49
3285/3285 [==============================] - 3s 845us/step - loss: 0.2213 - accuracy: 0.9677 - val
_loss: 0.4815 - val_accuracy: 0.8695
Epoch 42/49
3285/3285 [==============================] - 3s 939us/step - loss: 0.2149 - accuracy: 0.9699 - val
_loss: 0.4441 - val_accuracy: 0.9250
Epoch 43/49
3285/3285 [==============================] - 3s 905us/step - loss: 0.1943 - accuracy: 0.9793 - val
_loss: 0.2159 - val_accuracy: 0.9791
Epoch 44/49
3285/3285 [==============================] - 3s 933us/step - loss: 0.2093 - accuracy: 0.9699 - val
_loss: 0.3177 - val_accuracy: 0.9394
Epoch 45/49
3285/3285 [==============================] - 3s 874us/step - loss: 0.2375 - accuracy: 0.9641 - val
_loss: 0.1956 - val_accuracy: 0.9805
Epoch 46/49
3285/3285 [==============================] - 3s 868us/step - loss: 0.2063 - accuracy: 0.9760 - val
_loss: 0.3386 - val_accuracy: 0.9474
Epoch 47/49
3285/3285 [==============================] - 3s 898us/step - loss: 0.2126 - accuracy: 0.9753 - val
_loss: 0.2572 - val_accuracy: 0.9733
Epoch 48/49
3285/3285 [==============================] - 3s 903us/step - loss: 0.2132 - accuracy: 0.9732 - val
_loss: 0.3720 - val_accuracy: 0.9430
Epoch 49/49
3285/3285 [==============================] - 3s 919us/step - loss: 0.1997 - accuracy: 0.9793 - val
_loss: 0.2354 - val_accuracy: 0.9784
```

Out[13]:

```
<keras.callbacks.callbacks.History at 0x18ff9e906c8>
```

In [14]:

```
# Confusion Matrix
print(confusion_matrix2(Y_test_Dynamic, model.predict(X_test_Dynamic), {0: 'Walking', 1: 'Walking Upstairs', 2: 'Walking Downstairs',}))
```

```
Pred                  Walking  Walking Downstairs  Walking Upstairs
True
Walking                   492                   3                 1
Walking Downstairs          2                 418                 0
Walking Upstairs            0                  24               447
```

In [15]:

```
score = model.evaluate(X_test_Dynamic, Y_test_Dynamic)
print(score)
```

```
1387/1387 [==============================] - 0s 99us/step
[0.23537334086938821, 0.9783706068992615]
```

In [16]:

```
model.save('Dynamic_class_model.h5')
```

Observations

- Dynamic class model has 97.83% validation accuracy.
- Our Dynamic class model also performs very good but it is having some issues while identifying walking upstairs and walking downstairs.

Model for Static class

In [17]:

```
model = Sequential()

model.add(Conv1D(32, 5, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.001), input_shape=(timesteps, input_dim)))
model.add(Conv1D(16, 3, activation='relu', kernel_initializer='he_normal', kernel_regularizer=l2(0.01)))
model.add(Dropout(0.45))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(64, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(3, activation='softmax'))

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_4 (Conv1D)            (None, 124, 32)           1472
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 122, 16)           1552
_________________________________________________________________
dropout_5 (Dropout)          (None, 122, 16)           0
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 61, 16)            0
_________________________________________________________________
flatten_3 (Flatten)          (None, 976)               0
_________________________________________________________________
dense_5 (Dense)              (None, 64)                62528
_________________________________________________________________
dense_6 (Dense)              (None, 3)                 195
```

```
=================================================================
Total params: 65,747
Trainable params: 65,747
Non-trainable params: 0
_________________________________________________________________
```

In [18]:

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

# Training the model
model.fit(X_train_Static,
          Y_train_Static,
          batch_size=64,
          validation_data=(X_test_Static, Y_test_Static),
          epochs=30)
```

```
Train on 4067 samples, validate on 1560 samples
Epoch 1/30
4067/4067 [==============================] - 1s 189us/step - loss: 0.5892 - accuracy: 0.8879 - val
_loss: 0.5801 - val_accuracy: 0.8692
Epoch 2/30
4067/4067 [==============================] - 0s 84us/step - loss: 0.4342 - accuracy: 0.9095 - val_
loss: 0.6168 - val_accuracy: 0.8449
Epoch 3/30
4067/4067 [==============================] - 0s 87us/step - loss: 0.3644 - accuracy: 0.9169 - val_
loss: 0.5273 - val_accuracy: 0.8744
Epoch 4/30
4067/4067 [==============================] - 0s 93us/step - loss: 0.3208 - accuracy: 0.9159 - val_
loss: 0.5319 - val_accuracy: 0.8859
Epoch 5/30
4067/4067 [==============================] - 0s 85us/step - loss: 0.2917 - accuracy: 0.9230 - val_
loss: 0.5039 - val_accuracy: 0.8750
Epoch 6/30
4067/4067 [==============================] - 0s 83us/step - loss: 0.2709 - accuracy: 0.9216 - val_
loss: 0.4587 - val_accuracy: 0.8615
Epoch 7/30
4067/4067 [==============================] - 0s 87us/step - loss: 0.2524 - accuracy: 0.9265 - val_
loss: 0.5580 - val_accuracy: 0.8635
Epoch 8/30
4067/4067 [==============================] - 0s 86us/step - loss: 0.2461 - accuracy: 0.9248 - val_
loss: 0.6068 - val_accuracy: 0.8692
Epoch 9/30
4067/4067 [==============================] - 0s 97us/step - loss: 0.2337 - accuracy: 0.9292 - val_
loss: 0.5964 - val_accuracy: 0.8654
Epoch 10/30
4067/4067 [==============================] - 0s 86us/step - loss: 0.2276 - accuracy: 0.9292 - val_
loss: 0.6412 - val_accuracy: 0.8436
Epoch 11/30
4067/4067 [==============================] - 0s 100us/step - loss: 0.2175 - accuracy: 0.9343 - val
_loss: 0.5408 - val_accuracy: 0.8487
Epoch 12/30
4067/4067 [==============================] - 0s 94us/step - loss: 0.2122 - accuracy: 0.9348 - val_
loss: 0.6241 - val_accuracy: 0.8583
Epoch 13/30
4067/4067 [==============================] - 0s 88us/step - loss: 0.2135 - accuracy: 0.9319 - val_
loss: 0.6989 - val_accuracy: 0.8821
Epoch 14/30
4067/4067 [==============================] - 0s 95us/step - loss: 0.2003 - accuracy: 0.9371 - val_
loss: 0.6297 - val_accuracy: 0.8801
Epoch 15/30
4067/4067 [==============================] - 0s 90us/step - loss: 0.2082 - accuracy: 0.9368 - val_
loss: 0.5916 - val_accuracy: 0.8423
Epoch 16/30
4067/4067 [==============================] - 0s 98us/step - loss: 0.1944 - accuracy: 0.9412 - val_
loss: 0.5883 - val_accuracy: 0.8571
Epoch 17/30
4067/4067 [==============================] - 0s 87us/step - loss: 0.1889 - accuracy: 0.9398 - val_
loss: 0.8149 - val_accuracy: 0.8654
Epoch 18/30
4067/4067 [==============================] - 0s 93us/step - loss: 0.1917 - accuracy: 0.9390 - val_
loss: 0.6948 - val_accuracy: 0.8641
```

```
Epoch 19/30
4067/4067 [==============================] - 0s 95us/step - loss: 0.1809 - accuracy: 0.9430 - val_
loss: 0.6399 - val_accuracy: 0.8788
Epoch 20/30
4067/4067 [==============================] - 0s 88us/step - loss: 0.1828 - accuracy: 0.9439 - val_
loss: 0.6057 - val_accuracy: 0.8590
Epoch 21/30
4067/4067 [==============================] - 0s 88us/step - loss: 0.1812 - accuracy: 0.9432 - val_
loss: 0.6477 - val_accuracy: 0.8628
Epoch 22/30
4067/4067 [==============================] - 0s 90us/step - loss: 0.1860 - accuracy: 0.9380 - val_
loss: 0.6618 - val_accuracy: 0.8833
Epoch 23/30
4067/4067 [==============================] - 0s 88us/step - loss: 0.1698 - accuracy: 0.9486 - val_
loss: 0.6620 - val_accuracy: 0.8910
Epoch 24/30
4067/4067 [==============================] - 0s 99us/step - loss: 0.1818 - accuracy: 0.9415 - val_
loss: 0.7011 - val_accuracy: 0.8769
Epoch 25/30
4067/4067 [==============================] - 0s 88us/step - loss: 0.1738 - accuracy: 0.9444 - val_
loss: 0.7368 - val_accuracy: 0.8859
Epoch 26/30
4067/4067 [==============================] - 0s 89us/step - loss: 0.1627 - accuracy: 0.9469 - val_
loss: 0.7011 - val_accuracy: 0.8558
Epoch 27/30
4067/4067 [==============================] - 0s 91us/step - loss: 0.1661 - accuracy: 0.9476 - val_
loss: 0.7004 - val_accuracy: 0.8788
Epoch 28/30
4067/4067 [==============================] - 0s 98us/step - loss: 0.1630 - accuracy: 0.9511 - val_
loss: 0.7455 - val_accuracy: 0.8808
Epoch 29/30
4067/4067 [==============================] - 0s 88us/step - loss: 0.1591 - accuracy: 0.9491 - val_
loss: 0.6871 - val_accuracy: 0.8654
Epoch 30/30
4067/4067 [==============================] - 0s 89us/step - loss: 0.1616 - accuracy: 0.9503 - val_
loss: 0.7667 - val_accuracy: 0.8891
```

Out[18]:

```
<keras.callbacks.callbacks.History at 0x18ffb985948>
```

In [19]:

```
# Confusion Matrix
print(confusion_matrix2(Y_test_Static, model.predict(X_test_Static), {0: 'Laying', 1: 'Sitting', 2:
'Standing',}))
```

```
Pred      Laying  Sitting  Standing
True
Laying       400       91         0
Sitting       55      477         0
Standing       0       27       510
```

In [20]:

```
score = model.evaluate(X_test_Static, Y_test_Static)
print(score)
```

```
1560/1560 [==============================] - 0s 91us/step
[0.7667308768209739, 0.889102578163147]
```

In [21]:

```
model.save('Static_class_model.h5')
```

Observations

- Static class model has 88.91% validation accuracy.
- Our Static class model performs good but it is having issues while identifying Laying and Sitting.

Final Model

In [22]:

```
from keras.models import load_model
from scipy.ndimage import gaussian_filter

class PredictActivity:
    def __init__(self):
        self.binary_model = None
        self.dynamic_model = None
        self.static_model = None

    def loadModels(self, binModelPath, dynamicModelpath, staticModelPath):
        self.binary_model = load_model(binModelPath)
        self.dynamic_model = load_model(dynamicModelpath)
        self.static_model = load_model(staticModelPath)

    def predict(self, X):
        y_bin = np.argmax(self.binary_model.predict(X), axis=1)

        X_dynamic = X[y_bin==1]
        X_static = X[y_bin==0]

        y_dynamic = np.argmax(self.dynamic_model.predict(X_dynamic), axis=1)
        y_static = np.argmax(self.static_model.predict(X_static), axis=1)

        y_dynamic = y_dynamic + 1
        y_static = y_static + 4

        output = np.zeros((X.shape[0]), dtype='int')
        output[np.where(y_bin==1)[0]] = y_dynamic
        output[np.where(y_bin==0)[0]] = y_static

        return output
```

In [23]:

```
# Loading saved models
predictactivity = PredictActivity()
predictactivity.loadModels('class_model.h5', 'Dynamic_class_model.h5', 'Static_class_model.h5')
```

```
WARNING:tensorflow:Large dropout rate: 0.65 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate
instead of keep_prob. Please ensure that this is intended.
WARNING:tensorflow:Large dropout rate: 0.6 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate
instead of keep_prob. Please ensure that this is intended.
```

In [24]:

```
# Checking and printing the accuracy score on validation Data
accuracy_score(Y_test, predictactivity.predict(X_test))
```

Out[24]:

```
0.9311163895486936
```

In [34]:

```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(Y_test, predictactivity.predict(X_test), labels=range(1,7)))
```

```
[[492   1   3   0   0   0]
 [  0 447  24   0   0   0]
 [  2   0 418   0   0   0]
 [  0   0   0 400  91   0]
 [  0   0   0  55 477   0]
 [  0   0   0   0  27 510]]
```

Observations

- Final model has 93.11% validation accuracy.
- Our Final model performs very good but it is having some issues while identifying some classes.
- But the overall performance is preety good as compare to all the models I have previously tried.

In [35]:

```
from prettytable import PrettyTable

t = PrettyTable()
t.field_names= ("Model Name", "Validation accuracy")
t.add_row(["2 class classifier", "100%"])
t.add_row(["Dynamic class model", "97.83%"])
t.add_row(["Static class model", "88.91%"])
t.add_row(["Divide & Conquer CNN - Final Model", "93.11%"])

print(t)
```

```
+------------------------------------+---------------------+
|             Model Name             | Validation accuracy |
+------------------------------------+---------------------+
|         2 class classifier         |         100%        |
|        Dynamic class model         |        97.83%       |
|         Static class model         |        88.91%       |
| Divide & Conquer CNN - Final Model |        93.11%       |
+------------------------------------+---------------------+
```

Procedure

Step - 1 : I have tried several architectures with LSTM but it was giving validation accuracy around 91-92 %.

Step - 2 : So as suggested I have tried Divide and Conquer CNN and I have achieved preety good results as compare to previous models. The steps are given below:

- So divide and Conqure is a stratergy in which we divide our program into smaller parts and after performing operations on smaller parts we combine them.
- Here for Human activity recognition too, we are first breaking our whole task into smaller tasks such as - Identifying Static class and Dyamic class. After identifying we are applying different models for both the classes.
- For the 2 class classifier I have achieved the validation accuracy as 100%.
- For the 2 Dynamic class model I have achieved the validation accuracy as 97.83%.
- For the 2 Static class model I have achieved the validation accuracy as 88.91%.
- After combing the final model gave the accuracy of 93.11%, which is very good because we have not taken any help from the experts and then also we are able to achieve this much accuracy.